Consistency-aware scheduling and load balancing in multi-server distributed virtual environments

Li, Yusen

2013

Li, Y. (2013). Consistency-aware scheduling and load balancing in multi-server distributed virtual environments. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/55064>

<https://doi.org/10.32657/10356/55064>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

**CONSISTENCY-AWARE SCHEDULING
AND LOAD BALANCING IN
MULTI-SERVER DISTRIBUTED
VIRTUAL ENVIRONMENTS**

**LI YUSEN
SCHOOL OF COMPUTER ENGINEERING
2013**

**CONSISTENCY-AWARE SCHEDULING
AND LOAD BALANCING IN
MULTI-SERVER DISTRIBUTED
VIRTUAL ENVIRONMENTS**

LI YUSEN

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2013

Abstract

A distributed virtual environment (DVE) is a kind of distributed interactive application which allows a group of participants connected via a network to interact with a shared application state. DVEs have been widely used in many areas such as military training, collaborative design, e-learning and networked multi-user games. Zone-based multi-server DVEs (MSDVEs) have been shown to have good scalability to support a large population of users. The fundamental goal of a DVE is to create a consistent view of the virtual world among participants. However, resource saturations may occur at servers due to huge resource demand and imbalanced workload distribution. Due to resource limitations, some state updates cannot be timely disseminated to the relevant participants, and thus inconsistency is likely to occur. The inconsistency can have a great influence on the experiences of clients and the problem will become even more serious as the scale of the DVE grows.

In this thesis, we study the position update scheduling and load balancing issues for improving consistency in MSDVEs with a set of servers constrained by upload bandwidth. In the update scheduling issue, we aim at investigating update schedules for minimizing the overall inconsistency in a MSDVE. Depending on how inconsistency is measured, we first formulate the update scheduling problems as inequality constrained optimization problems. Then, the update scheduling algorithms are investigated for minimizing the overall inconsistency in practical systems. The performance of the proposed algorithms is experimentally evaluated. The results show that our proposed algorithms generally outperform other algorithms under the same experimental settings.

In the load balancing issue, we consider zone mapping and client assignment to minimize the total time-space inconsistency in a MSDVE. Zone mapping refers to determining which zone is maintained by which server while client assignment refers to determining which client is connected to which server. Similarly, the zone mapping and client assignment problem is first formulated as a mix-integer programming problem. Then, a centralized algorithm and a distributed adaptive tuning algorithm

are proposed for minimizing the total inconsistency in practical systems. The performance of these algorithms is experimentally evaluated. The results show that the centralized algorithm normally performs close to the theoretical lower bound and the proposed algorithms significantly outperform other algorithms under the same experimental settings.

To evaluate the proposed algorithms in practical system, a multi-server Battle City Game has been implemented. Various experiments have been designed in the game. The results show high efficiency of the proposed algorithms for reducing the inconsistency of the game. The results also show that the proposed algorithms can greatly improve the playability of the game.

Acknowledgments

Foremost I would like to express my sincere gratitude to my supervisor Prof. Cai Wentong for his guidance, support, enthusiasm, and immense knowledge. He patiently provided the vision, encouragement and advice necessary for me to proceed through the doctoral program and complete my thesis. I gratefully acknowledge the scholarship towards my PhD from Nanyang Technological University. Also, I wish to thank my friends in PDCC and Singapore for giving me the laughter, joy, and support in the tough PhD life. I wish our friendship last forever. Last but not the least, I would like to thank my proud family, my parents, my elders, my brothers and my sisters for their infinite support throughout everything.

Contents

Abstract	i
Acknowledgments	iii
List of Figures	vii
List of Tables	ix
List of Notations	x
 1 Introduction	 1
1.1 Distributed Virtual Environment	1
1.2 Inconsistency in DVE	2
1.3 Aims, Scope and Contributions of this Thesis	4
1.4 Organization of the Thesis	7
 2 Related Work	 9
2.1 DVE Architectures	9
2.2 Consistency Maintenance Techniques in DVEs	11
2.2.1 Time Management Techniques	11
2.2.2 Information Management Techniques	13
2.3 Load Balancing in MSDVEs	19
2.3.1 Spatial Partition based Approach	19
2.3.2 Avatar/Client Partition based Approach	22
2.3.3 Client Assignment for Improving Interactivity	24
2.4 Summary	25
 3 System Model and Problem Definitions	 26
3.1 System Model	26
3.2 Problem Definitions	29
3.2.1 Problem Definitions of Update Scheduling	31
3.2.2 Problem Definition of Load Balancing	39

4	Update Schedules for Minimizing the Number of Replicas without QoS	43
4.1	Centralized Approximate Algorithm	44
4.2	Distributed Updating Strategies	47
4.2.1	Local Preference Update Strategy	47
4.2.2	Remote Preference Update Strategy	49
4.3	Simulation and Experiments	51
4.3.1	Experimental Settings	52
4.3.2	Experimental Results	54
4.4	Summary	61
5	Update Schedules for Minimizing Total Time-space Inconsistency	62
5.1	Lagrange Multipliers Based Updating Algorithm	64
5.1.1	Derivation of Heuristic	64
5.1.2	Measurement of Update Priority	68
5.1.3	Update Schedules in Practical System	72
5.2	Simulation and Experiments	74
5.2.1	Update Algorithms to Compare	74
5.2.2	Experimental Setup	75
5.2.3	Experimental Results	76
5.3	Summary	81
6	Zone Mapping and Client Assignment for Minimizing Total Time-space Inconsistency	83
6.1	Alternating Optimization	85
6.1.1	Solution of Problem $\mathcal{P}3-1$	86
6.1.2	Solution of Problem $\mathcal{P}3-2$	88
6.1.3	Theoretical Lower Bound of $\mathcal{P}3$	90
6.1.4	Computing Complexity	92
6.2	Adaptive Tuning in Dynamic Systems	93
6.3	Simulation and Experiments	95
6.3.1	Experiment 1: Ideal Movement Model	96
6.3.2	Experiment 2: Random and Clustered Mobility Models	101
6.4	Summary	103

7	Case Study	106
7.1	A Multi-server Battle City Game	106
7.2	Experimental Design	108
7.3	Experimental Results and Discussions	112
7.3.1	Evaluation of the LPUS and RPUS	112
7.3.2	Evaluation of the LMH	118
7.3.3	Evaluation of the HAA-AT	122
7.4	Summary	126
8	Conclusions and Future Work	127
8.1	Conclusions	127
8.2	Future Work	129
8.2.1	Fairness-aware Consistency Maintenance Issues	129
8.2.2	Consistency Maintenance in Other DVE Architectures	129
8.2.3	Exploring New DVE Infrastructures	130
	References	132

List of Figures

2.1	DVE architectures	11
2.2	Area of Interest	16
3.1	System model of zone-based MSDVE	26
4.1	Performance of with respect to upload bandwidth capacity	55
4.2	Performance with respect to network latency	56
4.3	Performance with respect to the variance of network latency	57
4.4	Performance with respect to probability p	59
4.5	Performance with respect to threshold value	60
5.1	Relations between t_{now} and $t_{last} + d$	69
5.2	The measurement of optimal μ	70
5.3	Performance with respect to regulating frequency	77
5.4	Performance with respect to upload bandwidth capacity	78
5.5	Performance with respect to network latency ($\nu=0.95$)	79
5.6	Performance with respect to network latency ($\nu=0.5$)	81
5.7	Performance with respect to probability p	82
6.1	Initial distributions of avatars in the virtual world	97
6.2	Performance with respect to upload bandwidth	98
6.3	Performance with respect to network latency	100
6.4	Performance with respect to the variance of network latency	101
6.5	Performance with respect to upload bandwidth capacity	102
6.6	Performance with respect to network latency	104
6.7	Performance with respect to the variance of network latency	105
7.1	A screenshot of a map (zone)	107
7.2	Experimental Setup	111
7.3	Performance with respect to upload bandwidth with imbalanced work- load distribution	112

7.4	Performance with respect to upload bandwidth with uniform workload distribution	113
7.5	Performance with respect to network latency with imbalanced workload distribution ($\nu=0.95$)	115
7.6	Performance with respect to network latency with uniform workload distribution ($\nu=0.95$)	115
7.7	Performance with respect to network latency with imbalanced workload distribution ($\nu=0.5$)	116
7.8	Performance with respect to threshold value with imbalanced workload distribution	117
7.9	Performance with respect to upload bandwidth with imbalanced work- load distribution	119
7.10	Performance with respect to upload bandwidth with uniform workload distribution	119
7.11	Performance with respect to network latency with imbalanced workload distribution ($\nu=0.95$)	120
7.12	Performance with respect to network latency with uniform workload distribution ($\nu=0.95$)	121
7.13	Performance with respect to network latency with imbalanced workload distribution ($\nu=0.5$)	122
7.14	Performance with respect to upload bandwidth with imbalanced work- load distribution	123
7.15	Performance with respect to upload bandwidth with uniform workload distribution	123
7.16	Performance with respect to network latency with imbalanced workload distribution ($\nu=0.95$)	124
7.17	Performance with respect to network latency with uniform workload distribution ($\nu=0.95$)	124
7.18	Performance with respect to network latency with imbalanced workload distribution ($\nu=0.5$)	125

List of Tables

2.1	Classification of load balancing approaches	19
4.1	Notations for $\mathcal{P}1$	44
4.2	The default values of some experimental parameters	53
5.1	Notations for $\mathcal{P}2$	63
6.1	Notations for $\mathcal{P}3$	84
6.2	Notations used in the heuristic algorithm	88
7.1	The default values of some experimental parameters	112

List of Notations

NZ	the number of zones in the DVE
z_i	the i th zone in the DVE
NS	the number of servers in the DVE
s_i	the i th server in the DVE
NA	the number of avatars in the DVE
NC	the number of clients in the DVE, $NC = NA$
c_i	the i th client in the DVE
a_i	the one-to-one corresponding avatar of c_i
NR	the number of replicas in the DVE
r_i	the i th replica in the DVE
B_i	the upload network bandwidth capacity of server s_i
$s(c)$	the server the client c is connected to
$s(a)$	the server maintaining avatar a
$c(r)$	the client where replica r resides
$a(r)$	the source avatar of replica r
$s_t(r)$	the target server of replica r
$s_c(r)$	the contact server of replica r
$ \cdot $	the function to map an item to index
$d(x, y)$	the network latency between node x and node y
d_k	the transmission delay of position update of replica r_k from r_k 's target server to the client where r_k resides
α	the upload bandwidth consumption for disseminating one position update
β	the upload bandwidth consumption for sending one forwarding request
u_j	the upload bandwidth consumption for relaying c_j 's input commands per frame if c_j is not connected to the server that maintains a_j
f_r	the updating frame length of server
NA_i	the number of avatars maintained by server s_i

$c_{i,k}$	the k th client connected to server s_i
$a_{i,j}$	the j th avatar maintained by server s_i , $1 \leq i \leq NS$, $1 \leq j \leq NA_i$
$r_{i,j}^k$	the number of $a_{i,j}$'s remote replicas without QoS. These replicas are maintained by clients connected to s_k
$\Phi(r_{i,j}^k)$	function to map $r_{i,j}^k$ to 0 or 1, which indicates whether there needs a forwarding request from s_i to s_k ($k \neq i$) for updating $a_{i,j}$'s remote replicas
x_i	variables used to represent how many replicas that have s_i as both target and contact server will be updated by s_i
$y_{i,j}^k$	a 0-1 variable to represent whether s_i will send relaying request to s_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
$z_{i,j}^k$	a 0-1 variable to represent whether s_k will relay the update packets for s_i to update $a_{i,j}$'s remote replicas maintained by the clients connected to s_k . 1 means will; whereas 0 means will not
p_k	the update period of r_k
$\Delta(r, t)$	the spatial difference between the position of r and the position of its source avatar at time t
$\Delta^*(\cdot)$	an increasing function such that $\Delta(r_k, t) = \Delta^*(r_k, t - (t_{last(k)} + d_k))$
$\mathcal{R}_i^{TC}$	the set of replicas who have s_i as both target server and contact server
$\mathcal{R}_i^T$	the set of replicas whose target server is s_i while contact server is not s_i
$\mathcal{R}_i^C$	the set of replicas whose contact server is s_i while target server is not s_i
Z	a zone mapping solution
C	a client assignment solution
$zs_{j,i}$	$zs_{j,i} = 1$ if zone z_j is mapped to server s_i or $zs_{j,i} = 0$ otherwise
$cs_{j,i}$	$cs_{j,i} = 1$ if client c_j is connected to server s_i or $cs_{j,i} = 0$ otherwise
$R(z_i)$	the set of replicas whose avatars are in zone z_j
$R(c_i)$	the set of replicas maintained by c_i
$NR(s_i)$	the total number of replicas that have s_i as the target server

Chapter 1

Introduction

This chapter provides some background knowledge of Distributed Virtual Environments (DVEs), which includes a high level description of DVE and the inconsistency in DVE systems. The research aims, scope, contributions and the organization of the thesis are also summarized.

1.1 Distributed Virtual Environment

In recent years, advances in high-speed networking technologies, computer graphics and computing power have enabled the rapid development of DVE. The aim of DVE is to deploy a virtual world on a group of computers connected via network, which allows multiple geographically distributed participants, also known as clients, to communicate and interact with each other through the shared virtual world. DVEs originated in the 1980's with military simulators and currently have been widely used in many different applications such as military training, e-learning and online games [12, 16, 24, 58, 79, 90].

Generally, each client is often represented by an entity, called an avatar, in the virtual world. A client can use his/her avatar to move or traverse in the virtual world and communicate and interact with other clients. In order to provide a continuous view of the world among clients, each client must maintain 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 network latency and processing delays etc, the perception and rendering of the virtual world can be different at each client participating in the DVE. Many complex strategies have been proposed to provide the illusion of a single, continuous virtual world state among clients. These complexities are part of what makes DVEs an active research area and the wide applications of DVEs is what makes the research on DVE valuable.

As the development of the DVE applications, the scale of DVE systems is becoming larger and larger. Like the most popular online game WoW [90], the peak number of simultaneous active users has exceeded 5 million [91]. 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 only interacts 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 [44], NetEffect [30], CittaTron [49] and many existing Massively Multiplayer Online Games (MMOGs) [30, 39, 44, 70].

1.2 Inconsistency in DVE

Ideally, all the clients in a DVE should see the same view of the shared virtual world at the same time independent of the DVE architecture and network condition. This is referred to as the absolute consistency which has been defined in [73, 97]. However, due to non-deterministic message transmission delays, absolute consistency is difficult to achieve and inconsistency is likely to occur in the system. Basically, the inconsistency in DVEs can be divided into two types: causal order violation and time-space inconsistency [97].

The first type of inconsistency is causal order violation. In the real world, things happen according to their causal orders. For instance, a tripping event may cause a

falling event. Without the tripping event, the falling event can never happen; and the tripping event must happen before the falling event. This causal order is deeply rooted in our minds, and if it is violated in the virtual world, clients will feel the virtual world unrealistic. For instance, in a First Person Shooter (FPS) game, player A shoots at player B, and player C watches this action. Player A first generates a message indicating it has fired. This message is received by player B, and player B then generates a message indicating that he is dead. Player C receives both messages, but the message indicating the fire event has been delayed in the network. This causes player C receiving the dead message from player B before the shoot message from player A. This obviously contradicts our real-life experiences. There are many studies on this issue in the Parallel Discrete Event Simulation (PDES) community [43].

Another type of inconsistency is called time-space inconsistency. In DVE systems such as MSDVEs, the state of the virtual world is typically maintained by servers [49, 70]. In order to improve the interactivity, each client maintains a local copy of the relevant part of the virtual world. When the state of the virtual world changes, state updates are delivered by the servers to the clients in real time to reflect the change. However, due to transmission delay and clock asynchrony, different clients may receive the same state update at different times. So, the states of the virtual world seen by the clients are different from that maintained by servers. The differences between the replicated copies of the virtual world maintained by clients and the primary copy of the virtual world maintained by servers are defined as time-space inconsistency [98]. It has been shown that the usability and overall quality of a DVE depends heavily on the time-space inconsistency [20, 21].

Different metrics have been proposed to measure time-space inconsistency in DVEs [36, 61, 98]. Diot et.al. [36] proposed a metric, called “drift distance”, to measure the time-space inconsistency in their MiMaze application. Drift distance is the spatial difference between the primary copy and the remote replicated copies of avatars at each time step. This measurement ignores the time duration of the inconsistency. On the other hand, Lui et.al. [61] proposed a measure which is based on time duration

only. The metric is known as “phase difference”, which is defined as the difference between the start times of rendering the same change to the virtual world at different clients. Zhou et.al. [98] proposed a method of measuring time-space inconsistency in a DVE in both the time and space domains. Taking a moving avatar into consideration, the metric is defined as the integration of the spatial difference between a replicated copy and the primary copy of the avatar over time. This metric has been shown effective in evaluating the time-space consistency property of a DVE [98].

Generally, causal order can be preserved if we sacrifice the real-time property of DVEs [43]. Moreover, the causal order is not as crucial in DVEs as in PDES scenarios [98]. However, time-space inconsistency cannot be avoided, since message transmission delay and clock asynchrony cannot be eliminated. In this thesis, we will concentrate on time-space inconsistency in DVEs and the metric proposed in [98] is adopted to measure the time-space inconsistency.

1.3 Aims, Scope and Contributions of this Thesis

At present, a large scale MSDVE like MMOGs can include millions of concurrent users spread across the world, which demands huge amount of computing and network bandwidth resources on various servers [91]. 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.

Facing huge resource demand, servers may get saturated with resources like CPU, memory, disk and network bandwidth. In the presence of resource limitations, some state updates may not be disseminated timely and time-space inconsistency is likely to be increased. This can have a great influence on the experiences of clients and the problem will become even more serious as the scale of the DVE grows. Therefore, how resources can be allocated efficiently to improve consistency is an important issue in the design of a scalable MSDVE.

Moreover, a DVE system is highly dynamic where 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 [83]. 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 [83]. The workload variability increases the necessity of an efficient load balancing algorithm in MSDVEs to efficiently assign the workload among different servers in the system. Otherwise, if the workload is not properly distributed, the saturations at servers will be worsened and the inconsistency will be increased.

In this thesis, we study the update scheduling and load balancing issues for improving consistency in MSDVEs (particularly focusing on zone based architecture) with a set of servers constrained by upload bandwidth (i.e., outgoing bandwidth). The update scheduling aims to schedule state updates according to their potential impacts on consistency. For example, the state update that may incur large inconsistency (e.g., the position update of fast moving avatar) will be disseminated with high priority. The existing works on update scheduling in DVEs all concentrate on the DVEs with single server [40, 87, 92]. In MSDVEs, the problem gets much more complicated due to the inter-server communications and the lacking of centralized control. To the best of our knowledge, there is no existing work that has been done on update scheduling in MSDVEs.

The load balancing in MSDVEs is generally achieved by zone mapping and client assignment algorithms. The existing work on load balancing in MSDVEs mainly aims to: (1) balance workload among servers; (2) reduce inter-server communication; and (3) improve the interactivity of the DVE (i.e., reduce round-trip latency for clients) [10, 62, 65, 66, 70]. However, none of the existing work has taken inconsistency as an explicit performance measure. In this thesis, we consider the load balancing issues in MSDVEs from a new perspective with the purpose of reducing inconsistency.

A large scale MSDVE system often requires a considerable upload bandwidth which would take a large part of budget [83]. Like the most popular online game

WoW, the peak number of simultaneous active users has exceeded 5 million and in total 34.5Gbps upload bandwidth is demanded from WoW data centers for gameplay [91]. In addition, unlike computing power which can be increased by adding more CPUs, the upload bandwidth cannot be increased without limit due to the constraint of backbone network capacity. Therefore, this thesis is primarily concerned with the upload bandwidth resource. However, the proposed techniques can be applied to manage other type of resources as well.

The contributions of this thesis are summarized as follows:

Update Scheduling Given a MSDVE with a set of servers with constraints of upload bandwidth, we investigate state update schedules to minimize the overall inconsistency of the DVE. We particularly focus on scheduling position update of avatars, that is because position update packets are sent most frequently and consume most of network bandwidth [87]. The overall inconsistency is defined in two different ways and two problems are formulated respectively.

In the first problem, the overall inconsistency of a DVE is measured by the total number of replicas without QoS. A replica of an avatar is defined as a replica without QoS if the time-space inconsistency between the replica and the primary copy of the avatar is larger than a predefined threshold. The number of replicas without QoS has been shown to be efficient in evaluating the overall inconsistency of a DVE system [59, 71]. Based on this definition, the first updating problem is formulated as an integer programming problem which is to minimize the number of replicas without QoS. A centralized algorithm and two distributed algorithms are investigated. Simulation results show that the distributed algorithms achieve similar performance as the centralized algorithm, which is quite close to a lower bound of the optimal solution.

In the second problem, the overall inconsistency in a given period is measured by the total time-space inconsistency of all replicas of avatars in a DVE over this period. The total time-space inconsistency has been widely used in evaluating the overall inconsistency of a DVE system [87, 98]. Based on the definition, the problem

of finding optimal update schedule to minimize the overall inconsistency is formulated as an inequality constrained problem in an ideal situation. Then, Lagrange Multipliers are used to derive a heuristic position update algorithm that can be used in practical systems. Simulation results show that the proposed algorithm significantly improves the overall consistency compared to other existing update algorithms.

Load Balancing A mathematical model is created for the load balancing problem to minimize the overall inconsistency for a given MSDVE. The overall inconsistency is defined as the total time-space inconsistency of all replicas of avatars in a DVE. The load balancing problem is first formulated as a mix-integer programming problem. Then, a centralized algorithm is investigated based on Alternate Optimization (AO) method. Moreover, a distributed adaptive tuning algorithm is proposed to adapt dynamic configuration changes in practical systems. Simulation results show that the performance of the proposed algorithms greatly outperforms other algorithms.

Case Study A multi-server Battle City Game is implemented and the proposed update scheduling and load balancing algorithms are applied to the game. Various experiments have been designed for evaluating the performance of the proposed algorithms in practical systems. The experimental results show that the proposed algorithms can reduce the inconsistency efficiently and improve the playability of the game substantially.

1.4 Organization of the Thesis

This thesis is organized into eight chapters.

- Chapter 2 introduces some related work on consistency maintenance techniques such as DVE architectures, consistency maintenance techniques and load balancing techniques in DVEs.
- The system model of the MSDVE which is concerned in this thesis is presented in Chapter 3. The problems addressed in the thesis are also formally defined in this chapter.

- In Chapter 4, the update schedules for minimizing the number of replicas without QoS are presented and evaluated by simulations.
- In Chapter 5, the update schedules for minimizing the total time-space inconsistency are presented and evaluated by simulations.
- The zone mapping and client assignment algorithms for minimizing total time-space inconsistency are presented and evaluated by simulations in Chapter 6.
- In Chapter 7, a case study is described, which evaluates all the proposed algorithms using a multi-server Battle City Game.
- In Chapter 8, the conclusion and future work are discussed.

Chapter 2

Related Work

In this chapter, we summarize the existing work that is related to our research issues. Firstly, the architectures to support a DVE system are discussed. Then, the consistency maintenance techniques commonly used in DVEs are summarized. At last, the load balancing techniques in MSDVEs are introduced.

2.1 DVE Architectures

Three types of architectures have been proposed to support DVEs: the client-server architecture, the peer-to-peer architecture and the multi-server architecture.

In the client-server architecture, the state of the virtual world is maintained by a single server. All clients are connected to the server. The state update is accomplished as follows: the initiating client sends the input commands to the server. The server performs the commands which may change the virtual world and result in a state update. Then, the server disseminates the update to other clients who need to know about this update so that they can update their local view of the virtual world. Systems such as MASSIVE-3 [47] and Community Place [55] are implemented in the client-server architecture. The client-server architecture allows easy handling of tasks like user's access control, management of the states of the virtual world, synchronization of clients etc. However, it does not scale well. If the number of clients is large, the demands for resources such as CPUs, network bandwidth will also be huge. Moreover, the single server is also a single point of failure.

In the peer-to-peer architecture, there is no central server for managing the state of the virtual world. Instead, each client is responsible for maintaining its own part

of the virtual world based on the input commands issued by other clients. Since the commands are transmitted directly from one client to another, the interaction time between clients in the peer-to-peer architecture is reduced compared to that in the client-server architecture. Moreover, peer-to-peer architecture is by nature more robust as there is no single point of failure. While the peer-to-peer architecture is promising, it has many challenging issues such as distributed storage, management and consistency of the virtual world state, neighbor discovery and protection against cheating etc. Systems such as DIVE [17] and Solipsis [42] are implemented in the peer-to-peer architecture.

The multi-server architecture is proposed by combining the advantages of the client-server and the peer-to-peer architectures and has become a popular platform for DVE systems [22, 29, 33, 76, 88]. In the multi-server architecture, the virtual world is maintained across a group of distributed servers. Each client connects to one server for sending user input commands and receiving state updates, while servers perform the virtual world state updates based on the operations issued by clients. There are two approaches for implementing the multi-server architecture: mirrored server architecture and zone based architecture.

In the mirrored server architecture [29], the entire virtual world is replicated at all servers in the system. A client may connect to its closest server (in terms of network delay) to reduce the communication delay. However, the state of the virtual world at each server need to be synchronized in real time to keep a consistent view among all servers, which may incur additional communication overhead between servers. Hence, the mirrored server architecture is not scalable, and is only suitable for small-scale virtual worlds with a few tens of clients for each server [28].

On the other hand, in the zone based architecture, the virtual world is usually spatially partitioned into several disjoint zones, with each zone managed by only one server. A client only interacts 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. Therefore, the zone based architecture is often used to deal with large-scale virtual environments with hundreds, or even

Figure 2.1: DVE architectures

thousands of clients interacting simultaneously such as in RING [44], NetEffect [30], CittaTron [49] and some existing Massively Multiplayer Online Games (MMOGs) [30, 39, 44, 70]. Three different architectures are shown in Figure 2.1.

2.2 Consistency Maintenance Techniques in DVEs

As has been discussed in Chapter 1, the absolute consistency is difficult to achieve in DVEs due to network delays. Causal order violation or time-space inconsistency is likely to occur in DVEs. The issues of how to maintain consistency in DVEs have been extensively studied. The proposed mechanisms can be classified into two general classes [34, 35]: time management techniques and information management techniques.

2.2.1 Time Management Techniques

A DVE is essentially a Distributed Discrete Event Simulation (DDES) [43]. A simulation is a system (a computer program) that represents or emulates the behavior of a physical system over time. There are three different notions of time when discussing a simulation. *Physical time* refers to the time in the physical system. *Wallclock time* refers to time during the execution of the simulation program. *Simulation time* is an abstraction used by the simulation to model physical time.¹ In a discrete simulation, the simulation model views the physical system as only changing state at

¹To better understand these notions, please refer to the examples given by [43]

discrete points in simulation time. According to the mechanism used by the simulation to advance simulation time, discrete simulations can be divided into two categories: *time-stepped* and *event-driven* simulations. In the time-stepped approach, simulation time is subdivided as a sequence of equal-sized time steps, and the simulation advances from one time step to the next. Generally, DVE adopts the time-stepped approach to advance the simulation time. The distributed characteristics of DVE also bring some time related issues such as clock asynchrony, causal order violation etc. Time management techniques aim to tackle these issues.

In an absolute consistent DVE all clients would render the same virtual world state at the “same (wall clock) time”. Therefore, all clocks of the participating computers must be completely synchronized. Several mechanisms have been proposed to synchronize the clocks, for example, by using the network time protocol [64] or GPS [26].

As has been discussed, in order to provide a realistic virtual world, events in DVEs such as state updates should be executed in causal order. Many techniques have been proposed to prevent causal order violations, which can be divided into two classes: conservative methods and optimistic methods. The conservative method prevents a client from advancing until it makes sure that the client will not receive any out-of-order event [44]. This mechanism never does speculative event processing and roll back can be avoided at the expense of response time. By contrast, the optimistic method executes each event as soon as it arrives. When a causal order violation occurs, it undoes all the events till the event which causes the violation (i.e., roll back) and starts execution again from that event [29, 52]. This mechanism takes risks by performing speculative event processing to save time. However, if an out-of-order event is received, the event processing must be rolled back.

Only to prevent out-of-order events is not sufficient to achieve absolute consistency in DVEs. In order to provide the same view at the same time to all clients, the same event should be rendered by all clients at the same time. However, due to network delays, events may arrive at different clients at different times. In order to execute the events at the same time, several mechanisms such as bucket synchronization [46]

and local lag [27] have been proposed to delay the execution of events at clients so that the same event can be executed at the same time. However, system response time will be affected due to the delayed execution.

Since the absolute consistency is not easy to achieve, Qin et.al. [73] proposed the concept of asynchronous consistency. The system is defined as asynchronously consistent as long as clients perceive the same virtual world (even at different times). Based on this definition, some mechanisms have been proposed to guarantee the asynchronous consistency in DVEs [73, 81].

2.2.2 Information Management Techniques

Information management techniques aim to reduce the amount of data that has to be managed or to improve the efficiency of processing and disseminating data so as to improve the consistency. In large scale DVEs, the state updates of the virtual world need to be disseminated to the clients in a timely manner, which requires huge amount of resources such as network bandwidth. Therefore, how to reduce the amount of data to be disseminated and how to efficiently manage the limited network bandwidth are important issues in such applications. Dead Reckoning (DR), Relevance Filtering (RF) and state update scheduling are widely used mechanisms in DVEs.

2.2.2.1 Dead Reckoning

DR is an important predictive contract agreement mechanism utilized by many distributed interactive applications. This approach filters entity state updates to remove redundant communication according to a distance threshold value. Consider the state update of an entity from server to clients. The server maintaining the primary copy of each entity also maintains a parametric model to predict avatar position. When the difference between the actual position and the predicted position is larger than a threshold value, a new update packet (the current position and the parametric model are included) is generated and transmitted to the clients. Clients use the information in the update packet to predict the entity's position until next update packet is received. The usefulness of DR is largely dependent on the threshold value employed.

The threshold value determines the number of update packets and also determines the maximum difference between the two consecutive update packets.

The DR technique has the following benefits: First, if the prediction is very accurate, the DR technique can substantially reduce the frequency of the update packets. Thereby, a lot of unnecessary communication can be saved. Second, the DR technique can provide QoS guarantee and limit the maximum divergence. Once the divergence is larger than the predefined threshold value, a new update packet will be sent. Third, the DR technique can adapt to the current network condition by controlling the frequency of sending update packet. If the network bandwidth capacity is limited, we can increase the threshold to reduce the frequency.

DR techniques have been widely used in DVEs [3, 15, 37, 95]. An auto-adaptive DR algorithm was proposed for Distributed Interactive Simulation (DIS) in [15]. A multi-level threshold scheme was used to replace the fixed threshold. The levels of threshold are adaptively adjusted based on the relative distance between entities during the simulation. The auto-adaptive DR algorithm can considerably reduce update packets without sacrificing accuracy in extrapolating entity's position according to the experimental results.

Duncan et.al. proposed a pre-reckoning algorithm in [37]. A copy of the predictive model is used to anticipate changes that will likely result in the DR threshold being exceeded. When this situation is detected, the server issues an entity state update immediately rather than waiting for the DR threshold to be exceeded. Pre-reckoning algorithm could eliminate foreseeable error while some unnecessary updates may be issued in cases where the entity reverts to the predictable behavior.

Traditionally, when a client receives an update packet, it puts the entity at the current position specified by the update and starts extrapolating the path of the entity from that point using the local clock. Due to clock asynchrony, there will be inaccuracy in the extrapolation at client side. In order to make DR maintain high consistency, Aggarwal et.al. [3] proposed Globally Synchronized DR (GS-DR), which synchronizes the physical clocks of all participating clients and servers in a system

and adds time stamps to the update packets. In this manner, the inaccuracy caused by clock asynchrony in DR can be eliminated.

When the DR threshold is exceeded at server side, server will correct the DR prediction model and then sends the new model (in the new update packet) to clients. Due to transmission delay, before the new update packet is received by clients, the server and the clients use different prediction models. Inaccuracy is likely to occur in the extrapolation at client side, which was defined as *before inconsistency* [95]. In order to reduce the *before inconsistency*, Zhang et.al. [95] proposed a method named Globally Synchronized DR with Local Lag (GS-DR-LL). By delaying the model correction at server side, GS-DR-LL can significantly reduce the *before inconsistency*.

2.2.2.2 Relevance Filtering

Another technique for network bandwidth reduction is relevance filtering. The basic idea is to reduce the packet traffic to individual clients by sending only state updates that are relevant to them. The main idea of relevance filtering is based on the concept of “area of interest” (AOI). In DVEs, each client has an AOI which is defined as the area in the virtual world where interactions between this client controlled avatar and other avatars may take place. If client B is within client A’s AOI (see Figure 2.2), state updates made by client B will “affect” client A. With relevance filtering, each state update is delivered only to the clients who are affected by the update. It has been shown that this technique can reduce the number of state update packets transmitted significantly [7].

The relevance filtering techniques have been widely studied in DVEs. Most relevance filtering schemes to date utilize multicast (e.g. NPSNET [63], Spline [6], MiMaze [45]). In NPSNET [63], the virtual world is broken into hexagons, each representing a multicast group. Each avatar sends state updates to a multicast group corresponding to the local hexagon it is residing. This approach works well when avatars are distributed evenly within the virtual world, but fails if all avatars are clumped within the same cell. In Spline [6], the virtual world is partitioned into “locals” which can be any size or shape. It allows the designer to partition the virtual world so as to avoid

Figure 2.2: Area of Interest

the clumping problem. However, clumping can still occur because the designer can never know in advance how many avatars will be in one place.

Howard Abrams et.al. [2] proposed a three-tiered approach of relevant filtering for large scale virtual environments. The first tier works similarly to NPSNET, where the world is broken up into manageable regions. The second tier uses the data from the first tier to create a protocol independent perfect match between a client’s AOI and the virtual environment. The third tier adds protocol dependence, allowing the client to receive only the data from the protocol it needs. By separating out the protocol from the core filtering mechanism, multiple protocols are allowed to simultaneously exist within the same environment, while using the same underlying filtering mechanism. However, it still depends on multicast which may be limited by the number of multicast groups. It will also depend on the availability of multicast communication on the Internet. Otherwise, we need to assume that both the clients and the servers are connected using a Virtual Private Network (VPN) and the benefits of using multicast would be greatly reduced.

Tang and Cai [86] proposed a concept of “potential interest region” (PIR) as a spatial extension of AOI. A PIR calculated for an AOI should satisfy two constraints: (1) the PIR completely encloses the AOI (coverage constraint); and (2) all other unsubscribed entities are located outside the PIR (avoidance constraint). PIRs are only updated when at least one constraint is violated. PIRs enable the filtering to take

advantage of the relative positions between entities and AOIs to save communication cost. The proposed PIR-based filtering protocol is able to maintain highly accurate entity-in-AOI knowledge with substantially lower communication overhead.

The relevance filtering techniques in peer-to-peer DVEs also have been studied [50, 51]. Hu et.al. [50, 51] used Voronoi diagram to construct a network (i.e., VON) of all participating clients in peer-to-peer manner. Neighbor discovery in the network can be realized through mutual peer collaborations and bandwidth consumption at each peer is bounded. VON is efficient in terms of latency and number of communication messages. However, it is not easy to be applied in reality because of highly frequent topology adjustment.

2.2.2.3 Update Scheduling in DVEs

A great deal of work on task scheduling issues has been studied in distributed high performance computing environments such as clusters and Grid (e.g., [5, 48]). Most of the work focuses on allocating efficiently the workload among the processors to minimize the overall computing time. In some real-time systems, the work focuses on scheduling tasks with strict completion deadlines and the goal is to minimize the fraction of events that miss their deadlines (e.g., [74, 96]).

Research has also been carried out on investigating update schedules in DVEs for improving consistency. In [40], the authors proposed a priority round-robin algorithm to reduce the expected error in entity state. The error-metric (such as visual error) is defined by users and the priorities are set with the purpose of minimizing the expected error. However, the network latency is not considered in this algorithm.

The work presented by Yu et.al. [92] considered state update optimization issues in client-server MMOG under mobile network environment. The inconsistency is defined as the distortion between avatar's actual location at the server side and the location interpreted by clients, assuming the bandwidth which is provided by the underlying wireless access points is constrained. In order to minimize the inconsistency between the servers and clients, they proposed a network aware bandwidth allocation

algorithm. However, only is the spatial inconsistency considered in their work while the temporal impact on the inconsistency is not considered.

Zhou et.al. [99] proposed a utility model to evaluate the relative importance of a simulation entity. The utility of an entity is determined by the number of entities on which this entity may have influence and the distance between this entity and the entities within its area of influence. Based on the utility model, some flexible updating mechanisms were devised for utilizing the bandwidth more efficiently. However, the calculation of utility is rather complicated and may incur considerable overhead.

The work presented in [87] focused on the state update schedules for improving consistency in client-server DVEs with a single centralized server. The proposed algorithm was used to schedule state updates to reduce total time-space inconsistency with constraint of network capacity in single server DVEs. However, in MSDVEs, the situation gets much more complicated due to world partitioning, client assignment, and inter-server communications. To the best of our knowledge, hardly any similar work has been done to address these issues.

In this thesis, we consider the scheduling issue at the application level for minimizing the overall inconsistency in MSDVE. There are existing techniques that have been proposed to address the problem of minimizing bandwidth utilization with constraints on the consistency of replicas in other application domains (e.g., distributed database and sensor networks [72, 80]). On the contrary, in this thesis, we consider the dual problem which is to minimize inconsistency with network bandwidth constraints and in the application domain of DVEs.

2.2.2.4 Data Compression and Packet Bundling

The compression techniques have been widely used in DVEs [7, 23]. The objective of data compression is to reduce the size of the transmitted packets. Packets should be compressed by the sender and then decompressed by the receiver. Although the size of the data to be transmitted is reduced, it increases the computing overhead for both the sender and the receiver.

Different from data compression, packet bundling is to reduce the number of packets to be transmitted. It involves assembling a number of individual packets into a target data unit and transmitting this new unit as a single packet [7].

2.3 Load Balancing in MSDVEs

In a distributed system it is possible for some computers to be heavily loaded while others are lightly loaded, which can lead to poor system performance. The goal of load balancing is to improve the performance by balancing the loads among computers.

There are two main categories of load balancing policies: static policies and dynamic policies. Static policies make their decision based on statistical information about the system. The current state of the system is not taken into consideration [11, 56, 77]. Dynamic policies make their decision based on the current state of the system. Despite the higher runtime complexity dynamic policies can lead to better performance than static polices [18, 78].

The load balancing problem in MSDVEs has been extensively studied and the proposed approaches are summarized in Table 2.1. Basically, the approaches can be classified into two categories: spatial partition based approach and avatar/client partition based approach.

Table 2.1: Classification of load balancing approaches

		Static	Dynamic
Spatial Partition	Uniform	[19]	[31], [38], [49], [57], [70]
	Variable	[32], [82]	[4], [10], [22], [65]
Client/Avatar Partition		[62], [66], [69], [84], [85], [93], [94]	[60], [67], [68]

2.3.1 Spatial Partition based Approach

In the spatial partition based approach, the virtual world is generally partitioned into several small size regions, which are also known as cells or zones. The workload is balanced through distributing and transferring the regions among servers.

As shown in Table 2.1, the partitioning methods can be further divided into two major categories: uniform and variable partitioning shapes. Uniform partitioning divides the virtual world into cells with same shape and size. The most commonly used shapes for uniform partition are: square/rectangular, triangle and hexagon [19, 31, 38, 49, 57, 70].

In [31] and [49], the virtual world is divided into cells of equal size. Each server manages a set of adjacent cells. Load balancing is achieved by transferring cells from an overloaded server to its neighboring servers (i.e., servers that maintain neighboring cells). However, if the workload of the system is highly skewed, these algorithms may not produce effective results. This is because an overloaded server may not be able to transfer workload to its neighbors if all of its neighbors are also overloaded. Moreover, the load balancing algorithm may result in frequent client migrations if a client always needs to connect to the server that maintains his/her avatar. In order to reduce client migrations, Duong et.al. [38] proposed a dynamic load sharing algorithm as well as an efficient client migration algorithm.

Ng et.al. [70] proposed an adaptive region partitioning scheme for a distributed virtual walkthrough system, called Cyber-Walk. They initially partition the virtual world into cells and each cell is managed by one server. If the workload of a server exceeds a certain threshold, the server performs an adaptive load balancing algorithm. The server finds a target server from its neighboring servers. Then, it iteratively transfer cells to the target server until it is no longer overloaded. However, if the overloaded servers are neighbored with each other, it does not effectively and timely resolve the overloaded servers.

Lee et.al. [57] proposed a scalable dynamic load distribution scheme for MSDVEs, where avatars are highly skewed rather than uniformly distributed over a virtual environment. In the proposed scheme, an overloaded server firstly selects a set of servers which will be involved in the load balancing. Then, the initiating server repartitions the regions maintained by the involved servers such that all the involved servers have roughly equal workload. The involved servers then migrate their workloads with each other in a peer-to-peer manner according to the result.

Chen et.al. [19] proposed a load balancing algorithm, which is aware of the spatial locality in the virtual world. Based on the localized information, the algorithm balances the load and reduces the cross server communication, while avoiding frequent reassignment of regions.

Uniform partition fixes the cell size, which facilitates the design of the load balancing algorithm. However, if the workload is skewed, it is difficult to balance the workload among servers. Therefore, variable partition approaches have been proposed [4, 10, 22, 32, 65, 82], which creates cells with different shape and size in the partition.

Min et.al. [65] proposed a load balancing algorithm for distributed client-server MMOGs based on a spatial partitioning strategy. The virtual world is divided vertically against the X-axis. Each game server manages a partition of the virtual environment. Dynamic load balancing is achieved by relocating the partition line along the X-axis. An overloaded server may transfer some of its load to its two neighboring servers (the servers that maintain the neighboring regions) only. However, if these two neighbors cannot help, cascading load migrations may occur, which may affect the interactivity of the application.

In [82], Steed et.al. considered the partitioning issue under the assumption that they already have known the behavior of avatars in a virtual world. Under this assumption, several techniques were investigated to partition the virtual world into regions that can be mapped to servers. When constructing a partitioning, they attempted to minimize the overhead of the partitioning with respect to network management, while maintaining a bound on the number of entities that are mapped to any particular server.

De Vleeschauwer et.al. [32] proposed to split the virtual world into several smaller cells called “microcells”. These cells can be dynamically assigned to a set of servers, thus allowing an even distribution of the workload. A set of algorithms were proposed to assign the microcells to the available servers. The idea is to transfer cells managed by overloaded servers to lightly loaded ones, in a way such that the maximum load experienced by a server is minimized. However, some of the algorithms proposed in [32] did not consider the communication overhead between servers.

In [22], Chertov et.al. investigated the architecture of a unified environment where the virtual world is not partitioned according to rigid boundaries, but according to the clients' AOIs. Based on client's AOIs, some rectangular regions can be constructed that divide avatars evenly among servers. Each server uses a *client_threshold* value to determine the number of clients it is willing to serve. If *client_threshold* is exceeded, the server attempts to migrate part of the load to a nearby server.

Ahmed et.al. [4] proposed a zone (also called microcell) oriented load balancing model. To balance the load, their algorithm first finds all clusters of microcells which are managed by the overloaded server. Then, the smallest cluster (in terms of the number of microcells) is selected. In the smallest cluster, the microcell with the lowest communication with other microcells managed by the same server is chosen. The chosen microcell is then transferred to the least loaded server.

Bezerra et.al. [10] proposed a balancing approach for distributed MMOG servers, which considers the upload bandwidth occupation of the server as the load unit. They also considered some important aspects such as the quadratic growth of the traffic when the avatars are close to another, and the distribution overhead when clients connected to different servers are interacting. The proposed balancing approach is divided into three phases and different algorithms are proposed for each phase.

2.3.2 Avatar/Client Partition based Approach

In the spatial partition based approaches, avatars in the virtual world are managed in regions. The avatars in the same region are managed by the same server. Instead of subdividing virtual worlds into smaller regions, the avatar/client partition based approach handles avatars individually. Any client/avatar may be managed by any server. By managing avatars individually instead of entire regions, load assigned to the involved servers is fine grained. Hence, workload can be much easily distributed among the servers [60, 62, 66, 67, 68].

Lu et.al. [60] proposed an approach to balance CPU usage among a cluster of servers. The load balancing is done by a load balancer, which distributes clients among the servers in a round-robin manner. Thus each server will manager the same

number of clients. In this approach, although it does balance the number of clients among servers, communication between servers is not considered.

Lui et.al. [62] proposed an efficient partitioning algorithm that addresses the scalability issue of designing a large scale DVE system. The main idea is to dynamically divide the clients into different groups and then efficiently assign these groups to different servers. As a result, each server will process approximately the same amount of workload. Another objective of the algorithm is to reduce the server-to-server communication overhead. However, the algorithm is centralized. It is not scalable and may incur large server-to-server communication overhead.

Morillo et.al. [66, 68] showed that the behavior of DVE systems is non-linear with the number of avatars in the system. The performance of DVE systems greatly decreases when any of servers reaches 100% of CPU utilization. Therefore, they proposed a partitioning method that is targeted to keep all the servers in the system below a certain threshold of CPU utilization, regardless of the amount of network traffic. In this approach servers manage the workload generated by avatars, not regions of the virtual world. Evaluation results show that the proposed partitioning method can improve the performance of DVE system, independent from both the movement pattern of avatars and the initial distribution of avatars in the virtual world. However, similarly to [62], the proposed algorithm is also centralized.

The proposed algorithm in [68] balances the workload generated by all of the avatars as much as possible. However, the algorithm entirely assigns the exceeding workload that causes the saturation of a given server to the least loaded server in the system. When the system is close to its saturation point, all servers are heavily loaded. Hence, this algorithm can cause the cascading effect. In their latter work [67], a fine grained method for solving the partitioning problem in DVE systems has been proposed. This method consists of a global load balancing strategy, involving all the servers in the system. The exceeding workload that causes the saturation of a given server is proportionally distributed among the least loaded servers in the system. In this manner, the cascading effect can be reduced greatly.

2.3.3 Client Assignment for Improving Interactivity

In addition to balancing the load among servers, another objective of partitioning clients is to improve the interactivity by reducing the round-trip latency [69, 84, 85, 93, 94].

D. Ta et.al. [85] studied client-to-server assignment issues in zone-based MSDVEs. They noticed that due to the fact that clients in DVEs are geographically distributed and the heterogeneous nature of the Internet, a client in a zone may have large network delays to the server hosting that zone. Thus the interactivity of the DVE for that client may be greatly affected. Therefore, they proposed a two-phase client assignment algorithm to enhance the interactivity of the DVE. In the initial assignment phase, the zones of the virtual world are assigned to servers. Then, in the refined assignment phase, a client is assigned to its contact server (the server that the client will be assigned to) according to different scenarios. In the virtual location based scenario, the client will be assigned to the server that maintains the client's zone, i.e., the target server of the client. In the greedy assignment scenario, the assignment takes into account network delay from the client to its target server when selecting contact server for the client. If the round-trip delay of a client in a zone is high, the interactivity of the DVE for all clients in that zone may be affected. Therefore, in their latter work [84], two simple yet effective algorithms are proposed to reduce the round-trip client-to-server delay. The algorithms are based on the greedy heuristics that are used to solve the Terminal Assignment problem [54], which is known to be NP-complete.

Morillo et.al. [69] presented a partitioning method that presents a latency below a threshold to the maximum number of clients in DVE systems. The partitioning approach searches various assignments of avatars in order to find the one that represents the best trade-off among system latency, system throughput, and partitioning efficiency.

The work proposed in [93] focuses on the client assignment problem for enhancing the interactivity of Distributed Interactive Applications (DIAs). The authors formulated the problem as a combinational optimization problem on graphs and proved

that it is NP-complete. Several heuristic algorithms were proposed for fast computation of good client assignments. The algorithms have been shown to perform close to the optimal assignment. In their later work [94], they focused on continuous DIAs that change states not only in response to user operations but also due to the passing of time. They analyzed the minimum achievable interaction time for the continuous DIAs to preserve consistency so as to provide fairness among clients, and once again they formulated the client assignment problem as a combinatorial optimization problem. Four heuristic assignment algorithms were proposed and evaluated using real Internet latency data.

2.4 Summary

In summary, the existing update scheduling techniques mainly focus on single server DVEs. To our best of knowledge, the update scheduling issues in MSDVEs have not been studied. The existing load balancing techniques in MSDVEs mainly focus on: (1) balance workload among servers; (2) reduce inter-server communication; and (3) reduce round-trip latency for clients. However, none of the existing work considers inconsistency explicitly as a performance measure. In this thesis, we address the update scheduling and load balancing issues by considering them from a new perspective that aims to reduce the inconsistency of a MSDVE.

Chapter 3

System Model and Problem Definitions

In this chapter, the system model of zone-based MSDVE which we focus on in this thesis is described in detail. The scheduling and load balancing problems are formally defined.

3.1 System Model

Figure 3.1: System model of zone-based MSDVE

In this thesis, we focus on MSDVEs which in particular refers to zone based MSDVEs². In MSDVEs, as shown in Figure 3.1, the virtual world is assumed to

²In the rest of the thesis, MSDVE refers to the zone-based MSDVE

be partitioned into zones and each zone is maintained by one server. For instance, in Figure 3.1, 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, the AOI of an avatar is assumed to be the whole zone 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 users' input commands as well as state updates need to be forwarded by the connected servers thus the communications between servers are required. In our system model, a client is allowed to connect to any server, which is not necessarily the server that maintains his avatar or the closest server. The client assignment issue will be studied from a new perspective, which is presented in Chapter 6.

In this thesis, we focus on the position update of avatars. Based on the above system model, the position update in a MSDVE is accomplished as follows. Actions such as movement of an avatar are generated at client side according to user's input commands and then sent to the server that maintains the avatar. If the client is not connected to the server that maintains the avatar, the input commands need to be forwarded by the server that is connected to the client. As has been mentioned in Section 2.2.1, DVEs adopt time-stepped mechanism to advance the simulation time. At each time step, also known as *frame* or *update frame* in DVE, the server maintaining the avatar updates the position of the avatar according to the user's input commands and 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., in Figure 3.1, the replicas maintained by client c_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., in Figure 3.1, the replicas maintained by client c_3 and c_4), 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.

Consider a special case, as shown in Figure 3.1, avatar “circle” are replicated both at clients c_3 and c_4 , the two replicas share the same target server s_1 and contact server s_2 . Position updates of “circle” to the replicas need to be forwarded by s_2 . It seems that the forwarding requests for the two replicas from s_1 to s_2 can be merged for saving more bandwidth. However, in practice, the transmission delay of position update from the target server to the clients where the replicas are may be different. This therefore results in different update schedules. So, in our system model, the merge of forwarding requests in the above situation is not considered. However, the analysis for the current system model (without the merge of forwarding requests) can easily be applied to the system model that considers the merge of forwarding requests. We only need to let the replicas that share the same forwarding request have the same update period in the problem definition.

For a replica r , let $\Delta(r, t)$ be the spatial difference between the position of the replica at the client side and the position of its source avatar at the target server side at time t , the time-space inconsistency between the replica and the avatar over a

period $[t_{start}, t_{end}]$ is defined by [98]:

$$\int_{t_{start}}^{t_{end}} \Delta(r, t) dt \quad (3.1)$$

Suppose the transmission delay of the position update from replica r 's target server to replica r is d , due to the transmission delay, position update will not take effect until the client receives the update d time later. Then, after the replica receives an update, the growth of time-space inconsistency between replica r and its source avatar is given by:

$$\int_{t_{last}+d}^{t_{now}} \Delta(r, t) dt \quad (3.2)$$

where t_{now} is the current time and t_{last} is the time of the update last sent by r 's target server before $t_{now} - d$.

In our system model, we do not consider the situation that the servers are connected via dedicated lines. The traffic between servers needs to go through Internet. In this manner, the upload bandwidth consumption at each server can be divided into four parts. The first part is used for sending position updates to the replicas who have the server as both target server and contact server. The second part is used for sending forwarding requests to other servers. The third part is used for sending position updates based on the forwarding requests received from other servers. The last part is used for forwarding user's input commands if a client is connected to the server but the avatar of the client is maintained by another server.

To simplify the problem definitions, the wall clock times among all participating servers and clients in the system are assumed to be synchronized by protocols such as NTP [64].

3.2 Problem Definitions

Based on the system model described in the last section, three problems are formally defined. In the first two problems, we try to investigate update scheduling algorithms to minimize inconsistency in a MSDVE with a set of servers constrained by upload

bandwidth. In the third problem, we study the impact of the zone mapping and client assignment on the total inconsistency and investigate solutions to minimize the total inconsistency.

In order to formally define the problems, some common notations are defined first:

- NZ - the number of zones in the DVE
- $z_1, \dots, z_{NZ}$ - zones in the DVE
- NS - the number of servers in the DVE
- $s_1, \dots, s_{NS}$ - servers in the DVE
- NA - the number of avatars in the DVE
- $a_1, \dots, a_{NA}$ - avatars in the DVE
- NC - the number of clients in the DVE, $NC = NA$
- $c_1, \dots, c_{NC}$ - clients in the DVE
- NR - the total number of replicas of all avatars in the DVE
- $r_1, \dots, r_{NR}$ - replicas in the DVE
- $B_1, \dots, B_{NS}$ - server side upload bandwidth capacity per frame
- $s(c)$ - the server that client c is connected to
- $s(a)$ - the server that maintains avatar a
- $z(a)$ - the zone that avatar a is in
- $c(r)$ - the client where replica r resides
- $a(r)$ - the source avatar of replica r
- $s_t(r)$ - the target server of replica r , i.e., $s(a(r))$
- $s_c(r)$ - the contact server of replica r , i.e., $s(c(r))$

- $|\cdot|$ - the function to map an item to an index, e.g., $|a(r)|$ is the index of avatar $a(r)$ in the list of avatars
- $d(x, y)$ - the network latency between x and y , where x, y are either client or server. If $x = y$, $d(x, y) = 0$
- d_k - the transmission delay of position update of replica r_k from r_k 's target server to the client where r_k resides, $1 \leq k \leq NR$
- α - the upload bandwidth consumption for disseminating one position update
- β - the upload bandwidth consumption for sending one forwarding request
- u_j - the upload bandwidth consumption for relaying c_j 's input commands per frame if c_j is not connected to the server that maintains a_j , $1 \leq j \leq NC$
- f_r - the frame length of each server

3.2.1 Problem Definitions of Update Scheduling

In this section, the first two problems we are concerned in this thesis are formally defined. Consider a MSDVE with a set of servers constrained by upload bandwidth, the position update packets may not be disseminated timely and thus, the consistency of the virtual world cannot be guaranteed. At each update frame, if the available upload bandwidth cannot afford to disseminate or relay position updates to all the replicas, scheduling algorithms need to be developed to improve the consistency.

We defined two metrics to measure the overall inconsistency of a MSDVE: the total number of replicas without QoS and the total time-space inconsistency of all replicas over a time period. Both of the metrics have been used to measure the inconsistency of a DVE [71, 87]. A replica is defined *replica without QoS* if the time-space inconsistency between the replica (maintained by a client) and its avatar (maintained by the target server) is larger than a predefined threshold value. Based on the two metrics, two problems of minimizing the overall inconsistency are formulated. In the first problem, we try to investigate update schedules to minimize the number of replicas without

QoS in each update frame. In the second problem, the objective is to investigate update schedules to minimize the total time-space inconsistency over a period.

3.2.1.1 Minimizing the Number of Replicas without QoS

In this problem, we take the number of replicas without QoS as the metric to evaluate the overall inconsistency in each update frame. We try to investigate update schedules to minimize the number of replicas without QoS in every frame.

The following notations will be used in this problem definition:

- NA_i : the number of avatars maintained by server s_i , $1 \leq i \leq NS$
- $a_{i,j}$: the j th avatar maintained by server s_i , $1 \leq i \leq NS$, $1 \leq j \leq NA_i$
- $c_{i,k}$: the k th client connected to server s_i , $1 \leq i \leq NS$
- $r_{i,j}^k$: the number of $a_{i,j}$'s remote replicas without QoS. These replicas are maintained by clients connected to s_k
- $\Phi(r_{i,j}^k)$: function to map $r_{i,j}^k$ 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,

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

- x_i : a variables whose value indicates that how many replicas that have s_i as both target and contact server are updated by s_i , $1 \leq i \leq NS$.
- $y_{i,j}^k$: 0-1 variables which are used to represent whether s_i will send forwarding 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.
- $z_{i,j}^k$: 0-1 variables which are valid if and only if $y_{i,j}^k = 1$. It is used to represent whether s_k will send the update packets for s_i ($i \neq k$) according to the forwarding request 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, the problem is defined as the following integer programming problem (referred to as $\mathcal{P}1$). The objective function is:

$$\max \sum_{i=1}^{NS} (x_i + \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} y_{k,j}^i * z_{k,j}^i * r_{k,j}^i) \quad (3.3)$$

with the following constraint for each $1 \leq i \leq NS$:

$$\alpha x_i + \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} y_{i,j}^k + \alpha \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} y_{k,j}^i * z_{k,j}^i * r_{k,j}^i + \sum_l u_l \leq B_i \quad (3.4)$$

where $l \in \{j | 1 \leq j \leq NC, s(c_j) = s_i, s(a_j) \neq s_i\}$.

In the problem definition, x_i , $y_{k,j}^i$ and $z_{k,j}^i$ for all i, j, k are unknown variables to be determined. Other notations are given parameters. Since the aim of $\mathcal{P}1$ is to minimize the number of replicas without QoS in the DVE system, we need to update as many as possible replicas without QoS at each update frame. This is represented by Equation 3.3. We assume that updating a replica always brings it back to with QoS. Otherwise, if the replica is always without QoS no matter how frequently it is updated, the system is not playable. x_i represents the number of replicas updated by s_i directly. $\sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} y_{k,j}^i * z_{k,j}^i * r_{k,j}^i$ represents the number of replicas updated by s_i according to other servers' forwarding requests. The sum of these two parts is the total number of replicas without QoS updated by s_i . The first part in constraint (3.4) represents the upload bandwidth consumption on sending update packets directly to the clients connected to s_i itself. The second part represents the upload bandwidth consumption of s_i on sending forwarding requests to other servers. The third part represents the upload bandwidth consumption of s_i on sending update packets based on the forwarding request received from other servers. Suppose client c_l is connected to s_i . If the avatar of c_l (i.e., a_l) is not maintained by s_i , s_i needs to relay the input commands for c_l . Let u_l denote the upload bandwidth consumption for relaying c_l 's input commands. The total upload bandwidth consumption for relaying users' input commands at s_i can be represented by the last part of (3.4), i.e., $\sum_l u_l$, $l \in \{j | 1 \leq j \leq NC, s(c_j) = s_i, s(a_j) \neq s_i\}$. When the configurations of the DVE (e.g., client assignment, zone mapping, user's input commands rate etc) are fixed for a given DVE system, $\sum_l u_l$ is fixed as well.

$\mathcal{P}1$ is an integer programming problem and is NP hard [8]. It is difficult to get an optimal solution in polynomial time and especially hard in distributed manner. The solution of $\mathcal{P}1$ and the update schedules for minimizing the number of replicas without QoS are presented in Chapter 4.

3.2.1.2 Minimizing the Total Time-space Inconsistency

In this problem, we try to investigate update schedules to minimize the total time-space inconsistency, which reflects a long-term and overall situation of the inconsistency of a DVE [98].

In a MSDVE, suppose the upload bandwidth is limited at each server, only a given number of position updates are allowed to be disseminated. Inconsistency is likely to occur between replicas and the primary copy of avatars. The objective of this problem is to investigate position update schedules to minimize the sum of time-space inconsistency of all replicas in the DVE.

The following notations will be used in the problem definition.

- $\mathcal{R}_i^{TC}$ - the set of replicas who have s_i as both target server and contact server,
 $1 \leq i \leq NS$
- $\mathcal{R}_i^T$ - the set of replicas whose target server is s_i while contact server is not s_i ,
 $1 \leq i \leq NS$
- $\mathcal{R}_i^C$ - the set of replicas whose contact server is s_i while target server is not s_i ,
 $1 \leq i \leq NS$

In order to simplify the problem, we make the following assumption.

Assumption 3.1 *For each replica r_k ($1 \leq k \leq NR$), after each position update is received, the spatial difference between the replica and its avatar, i.e., $\Delta(r_k, t)$ grows in the same manner following an increasing function $\Delta^*(\cdot)$.*

According to this assumption, it follows that $\Delta(r_k, t) = \Delta^*(r_k, t - t_{last(k)})$, where $t_{last(k)}$ is the time of the update last sent by target server before t . It is plausible to

assume $\Delta^*(\cdot)$ to be increasing. Otherwise, if $\frac{d\Delta^*(r_k, t)}{dt} \leq 0$, which means the difference between replica r_k and its avatar either keeps unchanged or becomes smaller between two successive updates, position update would be unnecessary.

Suppose the network delay between target server and the client where the r_k resides is d_k . In the presence of transmission delays, a position update sent by target server at time t would not take effect in the client's view until time $t + d_k$. Therefore, the spatial difference is given by $\Delta(r_k, t) = \Delta^*(r_k, t - t_{last(k)})$, where $t_{last(k)}$ is the time of the update last sent by target server before $t - d_k$.

With this assumption, Equation (3.2) can be rewritten as:

$$\int_d^{t_{now} - t_{last}} \Delta^*(r, t) dt \quad (3.5)$$

Under the Assumption 3.1, for a single server DVE, the following is proven to be true in [87]:

Lemma 3.1 *In single server DVEs, given a fixed number of updates allowed in a period for a replica, periodical update with equal interval results in the minimum time-space inconsistency between this replica and its source avatar over this period.*

Therefore, in order to minimize total time-space inconsistency over all replicas in a single server DVE over a period, only does the optimal update period need to be determined for each replica.

In MSDVEs, we first define *optimal update schedule* over a period as follows:

Definition 3.1 *For a given set of servers with limited upload bandwidth over a period, an update schedule is optimal for this period if it: (1) minimizes total time-space inconsistency over all replicas; and (2) has the minimum upload bandwidth consumption compared to other update schedules that also minimize total time-space inconsistency.*

Based on the above definition, we can then have the following lemma:

Lemma 3.2 *For a replica in MSDVEs, if target server and contact server of the replica are different, updates for this replica should be forwarded by contact server without delay in any optimal update schedule.*

Proof Consider a replica whose target server and contact server are different. Thus, its updates need to be forwarded by the contact server. Suppose the average rate of generating updates for this replica at the target server is R_{gen} and these updates are forwarded by the contact server at rate R_{fwd} . In an optimal update schedule, $R_{fwd} = R_{gen}$ should be satisfied.

Firstly, it is obvious that $R_{fwd} \leq R_{gen}$. Assume there is an optimal update schedule and $R_{fwd} < R_{gen}$. This means that for the same period of time, the target server will generate more updates than the contact server can forward. The waiting time to forward the update at the contact server will be increased continuously and indefinitely as time proceeds. So, eventually, the update will be dropped when the contact server receives the next update for the same replica, since only the most recent update needs to be forwarded. This violates condition 2 of Definition 3.1. So, we have a contradiction. Therefore, $R_{fwd} = R_{gen}$ should be satisfied for an optimal update schedule. It implies that whenever an update generated by the target server is received by the contact server, there will be enough bandwidth to forward the update at the contact server. Thus, in an optimal update schedule, the state updates will not be delayed due to bandwidth limitations at the contact server and the lemma is proved. $\square$

According to Lemma 3.1, Definition 3.1, and Lemma 3.2, it follows Theorem 3.2.

Theorem 3.2 *In MSDVEs, for a given set of servers with limited upload bandwidth over a period, every replica should be updated periodically with equal interval in an optimal update schedule.*

Proof Consider a replica in a MSDVE, if its target server and contact server are the same, updates for this replica are directly disseminated by its target server to the client where the replica is, just like in a single server DVE. Therefore, according to Lemma 3.1, this replica should be updated periodically with equal interval.

Otherwise, if its target server and contact server are different, position updates need to be relayed by the contact server. According to Lemma 3.2, the update will be

forwarded without delay in an optimal update schedule. Hence, the situation can be also considered as a single server DVE but with a longer transmission delay. Again, according to Lemma 3.1, in order to minimize total time-space inconsistency, the update for the replica should be disseminated periodically with equal interval. $\square$

Definition 3.2 *Given a MSDVE, suppose the total time-space inconsistency over a period is only determined by the update period of replicas. All other configurations and system properties of the DVE which would influence the total time-space inconsistency, such as zone mapping, client assignment, server side bandwidth capacity, transmission delays etc, are assumed to keep unchanged over this period.*

Based on Theorem 3.2, to minimize total time-space inconsistency over all replicas, we just need to determine the optimal update period for each replica. We begin by analyzing the problem under an ideal situation defined by Definition 3.2. The analytical results then will be used to guide the design of the update scheduling algorithm used in practical systems.

For every replica r_k , $1 \leq k \leq NR$, suppose the update period of r_k is p_k , then, the time-space inconsistency between two consecutive received position updates is given by:

$$\int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \quad (3.6)$$

Firstly, we define the objective function to be minimized, which is the total time-space inconsistency over all replicas. Over a period T , the number of updates for r_k can be approximated by $\frac{T}{p_k}$, the total time-space inconsistency between r_k and its source avatar over T is given by:

$$\frac{T}{p_k} \cdot \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \quad (3.7)$$

Therefore, the total time-space inconsistency over all replicas is given by:

$$\sum_{k=1}^{NR} \frac{T}{p_k} \cdot \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \quad (3.8)$$

Then, we analyze the constraint of upload bandwidth for each server. For each server s_i , $1 \leq i \leq NS$, it needs to send position updates directly to the replicas who have s_i as both target server and contact server. The bandwidth consumption on this part over T is given by:

$$\alpha \cdot \left(\sum_j \frac{T}{p_j} \right) \quad (3.9)$$

where $j \in \{k | 1 \leq k \leq NR, r_k \in \mathcal{R}_i^{TC}\}$.

In addition, s_i needs to send forwarding requests for the replicas whose target server is s_i but contact server is not s_i . The bandwidth consumption on this part over T is given by:

$$\beta \cdot \left(\sum_m \frac{T}{p_m} \right) \quad (3.10)$$

where $m \in \{k | 1 \leq k \leq NR, r_k \in \mathcal{R}_i^T\}$.

Moreover, s_i receives forwarding requests from other servers and send updates to the destinations accordingly. The bandwidth consumption on this part over T is given by:

$$\alpha \cdot \left(\sum_l \frac{T}{p_l} \right) \quad (3.11)$$

where $l \in \{k | 1 \leq k \leq NR, r_k \in \mathcal{R}_i^C\}$. As has been mentioned, each server has to relay input commands for the client that is connected to the server but the avatar of the client is maintained by another server. For server s_i , the total upload bandwidth consumption for relaying users' input commands can be represented by $\sum_n u_n \cdot \frac{T}{f_r}$, $n \in \{j | 1 \leq j \leq NC, s(c_j) = s_i, s(a_j) \neq s_i\}$. Suppose the available upload bandwidth of s_i at each frame is constrained by B_i , the total upload bandwidth over T will be constrained by $B_i \cdot \frac{T}{f_r}$. Therefore, we have:

$$\alpha \cdot \left(\sum_j \frac{T}{p_j} \right) + \beta \cdot \left(\sum_m \frac{T}{p_m} \right) + \alpha \cdot \left(\sum_l \frac{T}{p_l} \right) + \sum_n u_n \cdot \frac{T}{f_r} \leq B_i \cdot \frac{T}{f_r} \quad (3.12)$$

Finally, the problem can be defined as the following inequality constrained problem (referred to as $\mathcal{P}2$), the objective is to minimize:

$$f(p) = \sum_{k=1}^{NR} \frac{T}{p_k} \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \quad (3.13)$$

subject to

$$g_i(p) \leq 0, \quad i = 1, \dots, NS \quad (3.14)$$

where $p = [p_1, \dots, p_{NR}]$ and $g_i(p)$ is defined as:

$$g_i(p) = \alpha \cdot \left(\sum_j \frac{T}{p_j} \right) + \beta \cdot \left(\sum_m \frac{T}{p_m} \right) + \alpha \cdot \left(\sum_l \frac{T}{p_l} \right) + \sum_n u_n \cdot \frac{T}{f_r} - B_i \cdot \frac{T}{f_r} \quad (3.15)$$

$\mathcal{P}2$ is an inequality constrained problem. The solution of $\mathcal{P}2$ and the updating algorithms for minimizing the total time-space inconsistency in a MSDVE are presented in Chapter 5.

3.2.2 Problem Definition of Load Balancing

In the problem definition of $\mathcal{P}2$, the configurations and system properties in a DVE such as zone mapping, client assignment, upload bandwidth capacity etc are assumed to be fixed in the examined period. Under that assumption, the total time-space inconsistency is only determined by the update periods of replicas. In this problem, we study the impact of zone mapping and client assignment on the total time-space inconsistency and investigate zone mapping and client assignment algorithms to minimize the total time-space inconsistency in a MSDVE with a set of servers constrained by upload bandwidth.

Except for zone mapping and client assignment, suppose other DVE configurations and system properties that would influence the inconsistency keep unchanged in the examined time period. Under this assumption, for a MSDVE with a set of servers constrained by upload bandwidth, the total time-space inconsistency over the examined time period is determined only by the zone mapping, client assignment and position update schedules used. This is because: (1) Different zone mapping and client assignment lead to different transmission delay between a replica and its target server. This could result in different time-space inconsistency; and (2) Different update schedules result in different intervals between two consecutive position updates. This could also result in different total time-space inconsistency. Therefore, to determine the zone

mapping and client assignment aiming at minimizing total time-space inconsistency, position update schedules also need to be considered.

Let a zone mapping solution is defined by the matrix Z , $Z = [zs_{j,i}]$, where the decision variables $zs_{j,i} = 1$ if zone z_j is mapped to server s_i or $zs_{j,i} = 0$ otherwise. Let a client assignment solution is defined by the matrix C , $C = [cs_{j,i}]$, where the decision variables $cs_{j,i} = 1$ if client c_j is connected to s_i or $cs_{j,i} = 0$ otherwise.

For every $1 \leq k \leq NR$, let p_k denote the update period of replica r_k . Assume the transmission delay from target server to r_k is d_k , the time-space inconsistency between two consecutive received position updates is given by

$$\int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \quad (3.16)$$

The total time-space inconsistency over period T is represented by:

$$\frac{T}{p_k} \cdot \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \quad (3.17)$$

Therefore, the total time-space inconsistency over all replicas over period T is given by:

$$\sum_{k=1}^{NR} \frac{T}{p_k} \cdot \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \quad (3.18)$$

According to the definition of d_k , $d_k = d(s_t(r_k), s_c(r_k)) + d(s_c(r_k), c(r_k))$. By using the variables in Z and C , it can be represented by:

$$d_k = \sum_{i=1}^{NS} \sum_{l=1}^{NS} d(s_i, s_l) \cdot zs_{|z(r_k)|,i} \cdot cs_{|c(r_k)|,l} + \sum_{i=1}^{NS} d(s_i, c(r_k)) \cdot cs_{|c(r_k)|,i} \quad (3.19)$$

where $z(r_k)$ is short for $z(a(r_k))$ (i.e., the zone where the source avatar of r_k is residing).

Consider replica r_k , if the position update of r_k does not need to be forwarded, the upload bandwidth consumption at r_k 's target server is $\frac{\alpha \cdot T}{p_k}$. Otherwise, if the position update needs to be forwarded, the upload bandwidth consumption at r_k 's target server is $\frac{\beta \cdot T}{p_k}$ and the upload bandwidth consumption at r_k 's contact server is represented by $\frac{\alpha \cdot T}{p_k}$.

For server s_i , the total upload bandwidth consumption can be divided into four parts. The first part is used for updating the replicas whose target server and contact server both are s_i , it can be represented by:

$$\sum_k z_{S|z(r_k)|,i} \cdot c_{S|c(r_k)|,i} \cdot \frac{\alpha \cdot T}{p_k} \quad (3.20)$$

where $1 \leq k \leq NR$. The second part is used for sending forwarding request for the replicas whose target server is s_i but contact server is not s_i . The bandwidth consumption can be represented by:

$$\sum_l \sum_k z_{S|z(r_k)|,i} \cdot c_{S|c(r_k)|,l} \cdot \frac{\beta \cdot T}{p_k} \quad (3.21)$$

where $1 \leq l \leq NS, l \neq i, 1 \leq k \leq NR$. The third part is used for forwarding position updates for the replicas whose contact server is s_i but target server is different, which can be represented by:

$$\sum_l \sum_k z_{S|z(r_k)|,l} \cdot c_{S|c(r_k)|,i} \cdot \frac{\alpha \cdot T}{p_k} \quad (3.22)$$

where $1 \leq l \leq NS, l \neq i$, and $1 \leq k \leq NR$. The last part is used for relaying users' input commands for the clients who are connected to s_i while the avatars of the clients are maintaining by other servers. It can be represented by:

$$\sum_l \sum_j z_{S|z(a_j)|,l} \cdot c_{S,j,i} \cdot u_j \cdot \frac{T}{f_r} \quad (3.23)$$

where $1 \leq l \leq NS, l \neq i, 1 \leq j \leq NC$ and u_j denotes the upload bandwidth consumption at $s(c_j)$ for relaying c_j 's input commands if $s(c_j) \neq s(a_j)$.

Suppose the upload bandwidth at s_i is constrained by B_i , the total available bandwidth over T is given by $B_i \cdot \frac{T}{f_r}$. It follows that

$$\begin{aligned} & \sum_k z_{S|z(r_k)|,i} \cdot c_{S|c(r_k)|,i} \cdot \frac{\alpha \cdot T}{p_k} + \sum_l \sum_k z_{S|z(r_k)|,i} \cdot c_{S|c(r_k)|,l} \cdot \frac{\beta \cdot T}{p_k} + \\ & \sum_l \sum_k z_{S|z(r_k)|,l} \cdot c_{S|c(r_k)|,i} \cdot \frac{\alpha \cdot T}{p_k} + \sum_l \sum_j z_{S|z(a_j)|,l} \cdot c_{S,j,i} \cdot u_j \cdot \frac{T}{f_r} \leq B_i \cdot \frac{T}{f_r} \end{aligned}$$

Therefore, the problem can be defined as the following mixed integer programming problem (referred to as $\mathcal{P3}$):

$$\min_{p,C,Z} f(p, C, Z) = \sum_{k=1}^{NR} \frac{T}{p_k} \cdot \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \quad (3.24)$$

constrained by

$$g_i \leq 0, 1 \leq i \leq NS \quad (3.25)$$

where

$$\begin{aligned} g_i = & \sum_k z_{S|z(r_k)|,i} \cdot c_{S|c(r_k)|,i} \cdot \frac{\alpha \cdot T}{p_k} + \sum_l \sum_k z_{S|z(r_k)|,i} \cdot c_{S|c(r_k)|,l} \cdot \frac{\beta \cdot T}{p_k} + \\ & \sum_l \sum_k z_{S|z(r_k)|,l} \cdot c_{S|c(r_k)|,i} \cdot \frac{\alpha \cdot T}{p_k} + \sum_l \sum_j z_{S|z(a_j)|,l} \cdot c_{Sj,i} \cdot u_j \cdot \frac{T}{f_r} - B_i \cdot \frac{T}{f_r} \end{aligned}$$

The mix-integer programming problem is NP hard [8]. The solution of $\mathcal{P}3$ and the zone mapping and client assignment algorithms for minimizing the total time-space inconsistency are presented in Chapter 6.

Chapter 4

Update Schedules for Minimizing the Number of Replicas without QoS

In this chapter, we investigate the update scheduling algorithms for minimizing the number of replicas without QoS. The problem has been formulated as $\mathcal{P}1$ in Chapter 3.

The objective function of $\mathcal{P}1$ is defined as follows:

$$\max \sum_{i=1}^{NS} (x_i + \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} y_{k,j}^i * z_{k,j}^i * r_{k,j}^i) \quad (4.1)$$

which is constrained by

$$\alpha x_i + \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} y_{i,j}^k + \alpha \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} y_{k,j}^i * z_{k,j}^i * r_{k,j}^i + \sum_l u_l \leq B_i \quad (4.2)$$

for all $1 \leq i \leq NS$, where $l \in \{j | 1 \leq j \leq NC, s(c_j) = s_i, s(a_j) \neq s_i\}$.

The definitions of the notations are shown again in Table 4.1. x_i represents the number of replicas updated by s_i directly. $\sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} y_{k,j}^i * z_{k,j}^i * r_{k,j}^i$ represents the number of replicas updated by s_i by relaying other servers' forwarding requests. The sum of these two parts is the total number of replicas without QoS updated by s_i . The left part of 4.2 represents the total upload bandwidth consumption at server s_i . The first item in 4.2 represents the upload bandwidth consumption on sending update packets directly to the clients connected to s_i itself. The second item represents the upload bandwidth consumption of s_i on sending forwarding requests to other servers. The third item represents the upload bandwidth consumption of s_i on sending the

Table 4.1: Notations for $\mathcal{P}1$

x_i	a variables whose value indicates that how many replicas that have s_i as both target and contact server are updated by s_i , $1 \leq i \leq NS$.
$y_{i,j}^k$	0-1 variables which are used to represent whether s_i will send forwarding 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.
$z_{i,j}^k$	0-1 variables which are valid if and only if $y_{i,j}^k = 1$. It is used to represent whether s_k will send the update packets for s_i ($i \neq k$) according to the forwarding request to update $a_{i,j}$'s remote replicas maintained by the clients connected to s_k . 1 means will; whereas 0 means will not.
$r_{i,j}^k$	the number of $a_{i,j}$'s remote replicas without QoS. These replicas are maintained by clients connected to s_k .
u_l	the upload bandwidth consumption for relaying c_l 's input commands per frame if c_l is not connected to the server that maintains a_l .

update packets based on the forwarding requests from other servers. $\sum_l u_l$ is the upload bandwidth consumption at s_i for relaying users' input commands.

The problem $\mathcal{P}1$ is an integer programming problem which is NP hard. It is difficult to get an optimal solution in polynomial time and especially hard in distributed manner. Therefore, we investigate approximate solutions. Firstly, a centralized algorithm to get an approximation solution of $\mathcal{P}1$ is proposed. Then, two fully distributed update scheduling algorithms are investigated. The proposed algorithms are theoretically analyzed and evaluated by wide range of experiments.

4.1 Centralized Approximate Algorithm

We first propose a centralized algorithm (denoted 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 of $\mathcal{P}1$ as the solutions which can update the same number of replicas without QoS. We also define a failed forwarding request as the one which is received by a server but not relayed by the server (i.e., $y_{i,j}^k = 1$, $z_{i,j}^k = 0$). We can prove that for any solution with failed forwarding request, we

can always find an equivalent solution without any failed forwarding request. For instance, we assume there is a solution A with $y_{i,j}^k = 1$ and $z_{i,j}^k = 0$. We can propose another solution B and the only difference between A and B is that $y_{i,j}^k$ is set to 0 in B. Therefore, the failed forwarding request does not exist in B. Due to $y_{i,j}^k * z_{i,j}^k$ 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 forwarding request, i.e., $z_{i,j}^k = y_{i,j}^k$ is always satisfied for $1 \leq i, k \leq NS$ and $1 \leq j \leq NA_i$. Therefore, if we can determine the values of all $y_{i,j}^k$, we can then get x_i for server s_i by the following equation according to (4.2):

$$x_i = \frac{1}{\alpha} (B'_i - \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} y_{i,j}^k - \alpha \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} y_{k,j}^i * z_{k,j}^i * r_{k,j}^i)$$

where $z_{k,j}^i = y_{k,j}^i$ and $B'_i = B_i - \sum_l u_l$.

Algorithm 1 Description of CAA

```

1: Phase One
2: for Each  $y_{i,j}^k$  where  $1 \leq i \leq NS$ ,  $1 \leq j \leq NA_i$  and  $1 \leq k \leq NS$ ,  $k \neq i$  do
3:   Set  $y_{i,j}^k = z_{i,j}^k = \Phi(r_{i,j}^k)$ 
4: end for
5: for  $1 \leq i \leq NS$  do
6:   Set  $x_i = \frac{1}{\alpha} (B'_i - \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k) - \alpha \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} \Phi(r_{k,j}^i) * r_{k,j}^i)$ 
7: end for
8: Phase Two
9: for  $1 \leq i \leq NS$  do
10:  for  $1 \leq j \leq NA_i$  do
11:    for  $1 \leq k \leq NS$ ,  $k \neq i$  do
12:      if  $y_{i,j}^k = 1$  and  $r_{i,j}^k < \sum_{j=1}^{NA_k} r_{k,j}^k - x_k$  then
13:        Set  $z_{i,j}^k = y_{i,j}^k = 0$ 
14:        Set  $x_k = x_k + r_{i,j}^k$ 
15:      end if
16:    end for
17:  end for
18:  Set  $x_i = \text{Min}(x_i + \frac{\beta}{\alpha} (\sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} \Phi(r_{i,j}^k) - \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} y_{i,j}^k), \sum_{j=1}^{NA_i} r_{i,j}^i)$ 
19: end for
    
```

Algorithm 1 shows the description of the CAA. There are two phases in the CAA. In the first phase, we make an initial solution by setting all $z_{i,j}^k = y_{i,j}^k = \Phi(r_{i,j}^k)$ and $x_i = \frac{1}{\alpha} (B'_i - \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k) - \alpha \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} \Phi(r_{k,j}^i) * r_{k,j}^i)$ (lines 2 to 7). Remember that $\Phi(r_{i,j}^k)$ indicates whether there needs a forwarding request from s_i

to s_k ($k \neq i$) for updating $a_{i,j}$'s remote replicas. $\Phi(r_{i,j}^k) = 1$ if $r_{i,j}^k > 0$, otherwise, $\Phi(r_{i,j}^k) = 0$. Therefore, $\beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k) + \alpha \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} \Phi(r_{k,j}^i) * r_{k,j}^i$ represents the upload bandwidth consumption at s_i for sending all the forwarding requests to other servers as well as sending position updates according to received forwarding requests to replicas. Here we assume the total upload bandwidth requirement on sending forwarding requests and sending updates according to received forwarding requests is always small compared to the bandwidth capacity at each server, i.e., $B'_i > \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k) + \alpha \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} \Phi(r_{k,j}^i) * r_{k,j}^i$ is always satisfied. This is reasonable because sending forwarding requests and sending updates according to received forwarding requests only happens when a client is connected to the server not maintaining his/her avatar. If a client is connected to the server that does not maintain his/her avatar, it implies that the network delay from the target server to the client is larger than the sum of the network delay from the target server to the contact server and the contact server to the client (i.e., triangle inequality violation (TIV) exists among the target server, the contact server and the client). Otherwise, just connecting the client to the target server directly will result in smaller inconsistency. According to the evidence from various real world latency data sets, only around 5% of triples of nodes have TIVs [89]. Therefore, such clients (whose target server and contact server are different) are the minority in the DVE system. Thus, the forwarding related upload bandwidth requirement should be small compared to the bandwidth capacity.

The objective of phase two is to save on the upload bandwidth used in sending forwarding requests and to increase the number of local updates. Lines 9 to 19 in Algorithm 1 show the refinement process. For each $y_{i,j}^k = 1$, we check whether $r_{i,j}^k < \sum_{j=1}^{NA_k} r_{k,j}^k - x_k$ is satisfied. If it is satisfied, $y_{i,j}^k$ can be reset to 0. This means that s_i does not need to send the forwarding request $r_{i,j}^k$ to s_k . This is because s_k has more than $r_{i,j}^k$ replicas without QoS connected directly to itself, which have not been updated (denoted by $\sum_{j=1}^{NA_k} r_{k,j}^k - x_k$). s_k can update the same number of replicas without QoS instead of disseminating $r_{i,j}^k$ update packets for server s_i . Therefore, the forwarding 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 should be updated (lines 13 to 15). The initialization of x_i at line 6 means that the remaining bandwidth (after being used for sending forwarding request and sending updates according to received forwarding requests) is all used for updating the replicas that have s_i as both target server and contact server. However, the bandwidth requirement for updating such replicas (represented by $\sum_{j=1}^{NA_i} r_{i,j}^i$) may be smaller than the bandwidth saved from sending forwarding request. In this case, x_i should be set to $\sum_{j=1}^{NA_i} r_{i,j}^i$ (line 18).

4.2 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 centralized control. We first give a local preference update strategy, then we propose a more efficient remote preference update strategy.

4.2.1 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 forwarding request from other servers will be processed at the end of each frame. A 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 forwarding request to other servers. The received forwarding requests from other servers will be processed at the end of each frame.

Algorithm 2 shows the description of the LPUS for server s_i . For avatar $a_{i,j}$, $1 \leq j \leq NA_i$, s_i tries to send the forwarding 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 1-20). After that, if the network bandwidth is still available, s_i will process the forwarding request from other servers (lines 21-29).

Algorithm 2 Description of LPUS for s_i

```

1: for Each  $a_{i,j}$ ,  $1 \leq j \leq NA_i$  do
2:   for Each  $r_{i,j}^k > 0$ ,  $1 \leq k \leq NS$  do
3:     if  $k \neq i$  then
4:       if  $B'_i - \beta > 0$  then
5:         Send forwarding request to  $s_k$ 
6:          $B'_i = B'_i - \beta$ 
7:       else
8:         return
9:       end if
10:    else
11:      if  $B'_i - \alpha * r_{i,j}^i > 0$  then
12:        Send position updates to  $r_{i,j}^i$  local replicas without QoS
13:         $B'_i = B'_i - \alpha * r_{i,j}^i$ 
14:      else
15:        Send position updates to  $\frac{1}{\alpha} * B'_i$  local replicas
16:      return
17:      end if
18:    end if
19:  end for
20: end for
21: for Each received forwarding request  $r_{k,j}^i$ ,  $1 \leq k \leq NS, k \neq i$  do
22:   if  $B'_i - \alpha r_{k,j}^i > 0$  then
23:     relay update packets to  $r_{k,j}^i$  replicas
24:      $B'_i = B'_i - \alpha r_{k,j}^i$ 
25:   else
26:     relay update packets to  $\frac{1}{\alpha} B'_i$  replicas
27:   return
28:   end if
29: end for

```

4.2.2 Remote Preference Update Strategy

4.2.2.1 Update Strategy Description

We notice that, the 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 as RPUS, for each server. Different from the LPUS, the forwarding requests of the local avatars and the forwarding requests from other servers are processed at the beginning of each frame. After that, if the network bandwidth is still 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 the RPUS for s_i .

Algorithm 3 Description of RPUS for s_i

```

1: Send forwarding request to all  $s_k$  for all  $r_{i,j}^k > 0$ ,  $1 \leq j \leq NA_i$ ,  $1 \leq k \leq NS$ , and  $k \neq i$ ;
2: Receive forwarding request  $r_{k,j}^i$  from all  $s_k$ ,  $1 \leq k \leq NS$ ,  $k \neq i$  and  $1 \leq j \leq NA_k$ ;

3:  $B'_i = B'_i - \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k)$ ;
4: for each non-zero  $r_{k,j}^i$  where  $1 \leq k \leq NS$ ,  $k \neq i$  and  $1 \leq j \leq NA_k$  do
5:   if  $B'_i - \alpha r_{k,j}^i > 0$  then
6:     relay update packets to  $r_{k,j}^i$  replicas
7:      $B'_i = B'_i - \alpha r_{k,j}^i$ 
8:   else
9:     relay update packets to  $\frac{1}{\alpha} B'_i$  replicas;
10:  return
11: end if
12: end for
13: if  $B'_i - \alpha \sum_{j=1}^{NA_i} r_{i,j}^i > 0$  then
14:   update  $\sum_{j=1}^{NA_i} r_{i,j}^i$  local replicas without QoS;
15: else
16:   select  $\frac{1}{\alpha} B'_i$  local replicas to update;
17: end if

```

Actually, the RPUS is just the distributed version of the first phase of the CAA. In the RPUS, first, s_i sends out all the forwarding requests of the local avatars to other servers and receives the forwarding requests from other servers (lines 1-3). Then, s_i tries to relay the update packets to the replicas for all the forwarding 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).

4.2.2.2 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 entire virtual environment.
- **Optimal Solution:** the total number of replicas without QoS updated by any optimal update strategy, denoted by N_{OPT} .

We first discuss the performance of s_i in the optimal strategy. We use $N_{OPT}^{s_i}$ to represent the number of replicas without QoS updated by s_i in the optimal strategy. Firstly, for any strategy, we have:

$$\begin{aligned} x_i + \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} y_{k,j}^i * z_{k,j}^i * r_{k,j}^i &\leq \frac{1}{\alpha} (B'_i - \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} y_{i,j}^k) \\ &\leq \frac{1}{\alpha} B'_i \end{aligned} \quad (4.3)$$

according to inequality (4.2). This means the number of replicas without QoS updated by s_i in any strategy should be smaller than $\frac{1}{\alpha} B'_i$. It follows

$$N_{OPT}^{s_i} \leq \frac{1}{\alpha} B'_i \quad (4.4)$$

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

$$B'_i - \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k) \quad (4.5)$$

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:

$$\alpha \left(\sum_{j=1}^{NA_i} r_{i,j}^i + \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} r_{k,j}^i \right) \quad (4.6)$$

Thus, if we use $N_{RPUS}^{s_i}$ to define the number of replicas without QoS updated by s_i in the RPUS, we have:

$$N_{RPUS}^{s_i} = \min\left(\sum_{j=1}^{NA_i} r_{i,j}^i + \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} r_{k,j}^i, \frac{1}{\alpha}(B'_i - \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k))\right) \quad (4.7)$$

according to (4.5) and (4.6). If $N_{RPUS}^{s_i}$ is equal to

$$\frac{1}{\alpha}(B'_i - \beta \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k))$$

we have:

$$N_{OPT}^{s_i} - N_{RPUS}^{s_i} \leq \frac{\alpha}{\beta} \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k) \quad (4.8)$$

according to (4.4). Otherwise, if $N_{RPUS}^{s_i}$ is equal to

$$\sum_{j=1}^{NA_i} r_{i,j}^i + \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} r_{k,j}^i$$

it means that all the replicas without QoS whose contact server is s_i are updated by s_i . Since $N_{OPT}^{s_i} \leq \sum_{j=1}^{NA_i} r_{i,j}^i + \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_k} r_{k,j}^i$ is always satisfied, it follows:

$$N_{OPT}^{s_i} - N_{RPUS}^{s_i} \leq 0 \quad (4.9)$$

Based on (4.8) and (4.9), we can conclude that for the whole DVEs, it follows:

$$N_{OPT} - N_{RPUS} \leq \sum_{i=1}^{NS} \frac{\alpha}{\beta} \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k) \quad (4.10)$$

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

4.3 Simulation and Experiments

We have implemented a event-driven simulator to simulate the MSDVE. The simulator is implemented by Java thus it is platform independent. The proposed algorithms were implemented in the simulator and a wide range of experiments were conducted to evaluate the performance. In this section, we will present our experimental results.

4.3.1 Experimental Settings

A MSDVE with a virtual world of size 5000x5000 distance units is simulated. The virtual world is partitioned into 100 equal sized zones and the number of servers is 10. There are 3000 simulated avatars, which move around the virtual world with two mobility models: random and clustered models. In the random model, avatars move randomly in the virtual world with random waypoint model [14]. In this model, each avatar randomly selects one location in the virtual world as the destination. It then travels towards this destination with a given constant velocity. Upon reaching the destination, the avatar chooses another random destination and moves towards it without any wait. In the clustered model, avatars are divided into several groups ($G_1, \dots, G_N$) and the avatars in the same group (e.g., G_i) move towards the same destination. Once an avatar in G_i reaches the destination, the destination will be changed to another randomly selected point in the virtual world. Each time an avatar reaches the border of the virtual world (i.e., next hop to the destination is outside of the virtual world), a new destination is randomly generated and the avatar moves towards to the new destination. If the next hop to the new destination is still outside of the virtual world, the avatar will stay and select another new destination in the next frame. The AOI of each avatar is the whole zone where the avatar is residing. The avatars in an avatar's AOI are replicated at its client side. Avatar positions are disseminated by servers in each frame to the relevant replicas. The speed of an avatar is randomly assigned from a uniform distribution between [0.1,10] distance units/s.

The network delay between two nodes (either server or client) is modeled by a shifted exponential distribution with a probability density function $f(x) = \lambda e^{-\lambda(x-\tau)}$ ($x \geq \tau$) [25]. To get a mean network delay d with a variance factor ν ($0 < \nu < 1$), τ and λ should be set to $\nu * d$ and $\frac{1}{(1-\nu)*d}$ respectively. Note that, if a position update needs to be forwarded, the transmission delay is the sum of network delay from the target server to the contact server and the network delay from the contact server to the client.

In this experiment, zones are randomly mapped to servers. Avatars are initially randomly distributed in the virtual world. Each client is initially connected to the

Table 4.2: The default values of some experimental parameters

Parameter	Default Value
Moving Speed	[0.1,10] distance units/s
Threshold to define Replicas without QoS	2.5 units* s
Upload Bandwidth Capacity	200 bandwidth units/frame
Network Latency	0.1s
Network Latency Variance	0.95
Probability p	0.8
Avatar Group Number (Clustered Model)	10
Frame Length (f_r)	0.025s
Upload Bandwidth Consumption	$\alpha=\beta=1$ unit
User Input Commands (u_i)	1 unit/frame

server that maintains its avatar with a probability p (i.e., with a probability $1-p$ to connect to other servers). If an avatar moves from a zone to another zone which is maintained by a different server, the avatar will be migrated to the new server that maintains the new zone. The associated client will be migrated to the new server with a probability p or stay unchanged otherwise.

Due to resource limitations, not all the replicas without QoS can be updated at each update frame. The remaining replicas without QoS which cannot be updated due to the resource limitation at each frame is denoted by R . It will be used 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 (4.10), we can get

$$R_{OPT} \geq N_{Total} - N_{RPUS} - \sum_{i=1}^{NS} \frac{\alpha}{\beta} \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k)$$

Therefore, we define

$$R^* = \text{Max}(0, N_{Total} - N_{RPUS} - \sum_{i=1}^{NS} \frac{\alpha}{\beta} \sum_{k=1, k \neq i}^{NS} \sum_{j=1}^{NA_i} \Phi(r_{i,j}^k))$$

as the lower bound of the number of remaining replicas without QoS.

A variety of simulations were conducted to evaluate the performance of all update algorithms on the impact of server side upload bandwidth, network latency, the

probability p and the threshold of replicas without QoS. For each experiment, the simulation runs 10 minutes after an initial warm-up period and the average value of all the examined metrics are calculated after repeating the same experiment several times³. The default values of the parameters are shown in Table 4.2.

4.3.2 Experimental Results

4.3.2.1 Performance with respect to Upload Bandwidth

First, we examined the impact of the upload bandwidth capacity. We decreased bandwidth from 1000 to 100 update units per frame per server with a step of 100 and calculated the average value of R for each update strategy.

Figure 4.1(a) and Figure 4.1(b) show the results with random and clustered mobility models. As can be seen, the number of remaining replicas without QoS increases as the bandwidth capacity decreases in both two mobility models. The RPUS can reduce the number of remaining replicas without QoS substantially compared to the LPUS with clustered model. This is because the main difference between the RPUS and the LPUS is the processing sequence of the forwarding request. In the RPUS, we first send out all the forwarding requests. The aim for this action is to transfer the workload to other servers. In the clustered model, the load balance cannot always be guaranteed among servers. When one server is overloaded, other servers may be under-utilized. Therefore, the RPUS performs better than the LPUS with clustered mobility model.

4.3.2.2 Performance with respect to Network Delay

Next, we examined the impact of network delay. For each experiment, we let the average network latency (d) change in the range from 0 to 0.5s and other parameters are set as shown in Table 4.2. As can be seen from Figures 4.2(a) and 4.2(b), for both random and clustered mobility models, the number of remaining replicas without QoS gets larger as the average latency increases. This is because time-space inconsistency

³Confidence interval is not shown because the results from different simulation runs with the same experimental setting vary little.

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 4.1: Performance of with respect to upload bandwidth capacity

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 4.2: Performance with respect to network latency

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 4.3: Performance with respect to the variance of network latency

is affected by network delays and superlinear increases with the delays. The RPUS achieves better performance than the LPUS with clustered mobility model.

4.3.2.3 Performance with respect to the Variance of Network Latency

Then, the performance of the proposed algorithms with respect to the variance of network latency was examined. For each experiment, we let the variance of network latency (ν) change in the range from 1 to 0.5 and other parameters are set as shown in Table 4.2. As can be seen from Figures 4.3(a) and 4.3(b), for both random and clustered mobility models, the number of remaining replicas without QoS increases as the variance of the network latency increases⁴. This is because time-space inconsistency is increasing superlinear with network latency. Large network latency as the result of large variance of network delay may result in large time-space inconsistency.

4.3.2.4 Performance with respect to Probability p

The impact of the probability p is also examined. For each experiment, we varied the probability p in the range from 1 to 0.5 and examined the performance of the proposed algorithms for both the random and clustered mobility models. Other parameters are set as shown in Table 4.2.

Intuitively, as the probability p decreases, more clients will be connected to servers that do not maintain their avatars. More position updates need to be forwarded through contact servers, which makes the bandwidth contention more serious and thus results in the increase of the remaining replicas without QoS as shown in Figure 4.4(a) and Figure 4.4(b). Moreover, the RPUS updates more replicas without QoS than the LPUS with clustered model.

4.3.2.5 Performance with respect to Threshold Value

At last, the impact of the time-space threshold value to define replica without QoS is examined. The threshold value was varied from 0.5 to 4.5 and the results are shown in Figure 4.5(a) and Figure 4.5(b). As can be seen, as the threshold value increases, the number of remaining replicas without QoS decreases for both random and clustered

⁴Note that small value of variance factor ν results in large variance of network delay (d)

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 4.4: Performance with respect to probability p

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 4.5: Performance with respect to threshold value

mobility models. This is because given fixed upload bandwidth allocations, the number of replicas can be updated keeps unchanged. Thus, the total number of replicas without QoS is decreased. Moreover, the RPUS performs much better than LPUS with clustered model.

4.4 Summary

In this chapter, the solution of $\mathcal{P}1$ is discussed. A centralized approximate algorithm (CAA) and two distributed update algorithms (RPUS and LPUS) are investigated. The proposed algorithms are validated by theoretical analysis and wide range of experiments.

For various experimental settings, the RPUS achieves better performance than the LPUS. However, the performance difference is more significant for the clustered mobility model than the random mobility model. This implies that the RPUS is able to allocate the limited resources more efficiently than the LPUS when the workload is imbalanced among servers.

Chapter 5

Update Schedules for Minimizing Total Time-space Inconsistency

In this chapter, we investigate update scheduling algorithms for minimizing the total time-space inconsistency in MSDVE. The problem has been formulated to $\mathcal{P}2$ in Chapter 3 for an ideal situation defined by Definition 3.2. Based on $\mathcal{P}2$, in this chapter, we derive the updating algorithms that can be used in practical systems.

The $\mathcal{P}2$ has been defined as an inequality constrained problem in Chapter 3 as follows:

$$\min f(p) = \sum_{k=1}^{NR} \frac{T}{p_k} \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \quad (5.1)$$

subject to

$$g_i(p) \leq 0, \quad i = 1, \dots, NS \quad (5.2)$$

where $p = [p_1, \dots, p_{NR}]$ and $g_i(p)$ is defined as:

$$g_i(p) = \alpha \cdot \left(\sum_{r_j \in \mathcal{R}_i^{CT}} \frac{T}{p_j} \right) + \beta \cdot \left(\sum_{r_m \in \mathcal{R}_i^T} \frac{T}{p_m} \right) + \alpha \cdot \left(\sum_{r_l \in \mathcal{R}_i^C} \frac{T}{p_l} \right) + \sum_n u_n \cdot \frac{T}{f_r} - B_i \cdot \frac{T}{f_r} \quad (5.3)$$

where $n \in \{j | 1 \leq j \leq NC, s(c_j) = s_i, s(a_j) \neq s_i\}$.

The notations in the problem definition are listed again in Table 5.1. The objective function $f(p)$ represents the total time-space inconsistency of all replicas over a period T . $g_i(p) \leq 0$ is the upload bandwidth constraint at server s_i for $1 \leq i \leq NS$. $\alpha \cdot \left(\sum_{r_j \in \mathcal{R}_i^{CT}} \frac{T}{p_j} \right)$ represents the bandwidth consumption at s_i for sending updates to the replicas whose target server and contact server both are s_i . $\beta \cdot \left(\sum_{r_m \in \mathcal{R}_i^T} \frac{T}{p_m} \right)$

is the bandwidth consumption at s_i for sending forwarding request for the replicas whose target server is s_i and contact server is different. $\alpha \cdot \left(\sum_{r_l \in \mathcal{R}_i^C} \frac{T}{p_l} \right)$ represents the bandwidth consumption for disseminating position updates for the replicas whose contact server is s_i but target server is different. $\sum_n u_n \cdot \frac{T}{f_r}$ is the upload bandwidth consumption for relaying user's input commands. The total upload bandwidth is constrained by $B_i \cdot \frac{T}{f_r}$.

 Table 5.1: Notations for $\mathcal{P}2$

p_k	update period of r_k , $1 \leq k \leq NR$.
d_k	transmission delay of the position update from r_k 's target server to r_k (may be forwarded by other server).
$\mathcal{R}_i^{TC}$	the set of replicas that have s_i as both target server and contact server.
$\mathcal{R}_i^T$	the set of replicas whose target server is s_i while contact server is not s_i .
$\mathcal{R}_i^C$	the set of replicas whose contact server is s_i while target server is not s_i .

The optimal update periods of a given problem instance of $\mathcal{P}2$ can be obtained by many existing gradient descent methods [8]. However, $\mathcal{P}2$ is defined for the ideal situation that the configurations and system properties are not changed. In practical systems, the DVE configurations and system properties are changing over time. Therefore, the solution of $\mathcal{P}2$ (optimal update periods in the ideal situation) cannot be directly used in practical systems.

In this thesis, instead of pursuing an optimal solution of $\mathcal{P}2$, we derive a heuristic update algorithm for the practical systems. The development of the algorithm consists of three steps:

- (i) Lagrange Multipliers are used to derive the conditions when the $\mathcal{P}2$ is minimized. Based on the analytical results, for the practical systems, a global updating priority is defined for each replica according to long-term benefit in terms of inconsistency reduction.
- (ii) Since the update priority measured in centralized manner may incur too much computational and communication overhead, a local regulating algorithm is proposed to estimate the update priority for each replica in a distributed manner.

- (iii) Based on the update priority of each replica, a distributed update algorithm is developed to minimize total time-space inconsistency.

5.1 Lagrange Multipliers Based Updating Algorithm

5.1.1 Derivation of Heuristic

To solve $\mathcal{P}2$, we can use Lagrange Multipliers and the Lagrange function is given by:

$$L(p, \mu) = f(p) + \sum_{i=1}^{NS} \mu_i \cdot g_i(p) \quad (5.4)$$

where $\mu = [\mu_1, \dots, \mu_{NS}]$ are Lagrange Multipliers. For problem $\mathcal{P}2$, according to the optimization theory [8], we have the following theorem:

Theorem 5.3 *Let p^* be such that $g_i(p^*) \leq 0$, for $1 \leq i \leq NS$. p^* is a local minimum for $\mathcal{P}2$ **iff** there exists Lagrange Multipliers vector $\mu^* = [\mu_1^*, \dots, \mu_{NS}^*]$ such that*

$$\nabla_p L(p^*, \mu^*) = 0, \text{ and} \quad (5.5)$$

$$\mu_i^* \geq 0, \quad \mu_i^* \cdot g_i(p^*) = 0, \quad \forall i = 1, \dots, NS \quad (5.6)$$

Proof: For necessity, it is an obvious conclusion according to the optimization theory [8].

For sufficiency, according to the optimization theory [8], to prove p^* is a local minimum, we just need to prove $\nabla_{pp}^2 L(p^*, \mu^*) > 0$.

For every r_k , $1 \leq k \leq NR$, if $s_t(r_k) = s_c(r_k)$,

$$\begin{aligned} \frac{\partial L}{\partial p_k} &= \frac{d \left(\frac{T}{p_k} \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \right)}{dp_k} + \alpha \cdot \mu_{|s_t(r_k)|} \cdot \left(-\frac{T}{p_k^2} \right) \\ &= \frac{T}{p_k} \cdot \Delta^*(r_k, p_k + d_k) - \frac{T}{p_k^2} \cdot \left(\int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \right) - \alpha \cdot \mu_{|s_t(r_k)|} \cdot \frac{T}{p_k^2} \\ &= \frac{T}{p_k^2} \cdot \left(p_k \cdot \Delta^*(r_k, p_k + d_k) - \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt - \alpha \cdot \mu_{|s_t(r_k)|} \right) \end{aligned} \quad (5.7)$$

Otherwise, if $s_t(r_k) \neq s_c(r_k)$,

$$\begin{aligned}
 \frac{\partial L}{\partial p_k} &= \frac{d\left(\frac{T}{p_k} \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt\right)}{dp_k} + (\beta \cdot \mu_{|s_t(r_k)|} + \alpha \cdot \mu_{|s_c(r_k)|}) \cdot \left(-\frac{T}{p_k^2}\right) \\
 &= \frac{T}{p_k} \cdot \Delta^*(r_k, p_k + d_k) - \frac{T}{p_k^2} \cdot \left(\int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt\right) - (\beta \cdot \mu_{|s_t(r_k)|} + \alpha \cdot \mu_{|s_c(r_k)|}) \cdot \frac{T}{p_k^2} \\
 &= \frac{T}{p_k^2} \cdot \left(p_k \cdot \Delta^*(r_k, p_k + d_k) - \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt\right. \\
 &\quad \left. - (\beta \cdot \mu_{|s_t(r_k)|} + \alpha \cdot \mu_{|s_c(r_k)|})\right)
 \end{aligned} \tag{5.8}$$

If we define $\varphi(r_k, \mu)$ as:

$$\varphi(r_k, \mu) = \begin{cases} \alpha \cdot \mu_{|s_t(r_k)|}, & \text{if } s_t(r_k) = s_c(r_k) \\ \beta \cdot \mu_{|s_t(r_k)|} + \alpha \cdot \mu_{|s_c(r_k)|}, & \text{if } s_t(r_k) \neq s_c(r_k) \end{cases} \tag{5.9}$$

$\frac{\partial L}{\partial p_k}$ can then be re-written as:

$$\frac{T}{p_k^2} \cdot \left(p_k \cdot \Delta^*(r_k, p_k + d_k) - \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt - \varphi(r_k, \mu)\right) \tag{5.10}$$

Thus,

$$\begin{aligned}
 \frac{\partial^2 L}{\partial p_k^2} &= \frac{-2T}{p_k^3} \cdot \left(p_k \cdot \Delta^*(r_k, p_k + d_k) - \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt - \varphi(r_k, \mu)\right) \\
 &\quad + \frac{T}{p_k^2} \cdot \left(\frac{d(p_k \cdot \Delta^*(r_k, p_k + d_k))}{dp_k} - \frac{d(\int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt - \varphi(r_k, \mu))}{dp_k}\right) \\
 &= \frac{-2}{p_k} \cdot \frac{\partial L}{\partial p_k} + \frac{T}{p_k} \cdot \frac{d\Delta^*(r_k, p_k + d_k)}{dp_k}
 \end{aligned} \tag{5.11}$$

Since $\Delta^*(\cdot)$ is an increasing function⁵, we have:

$$\frac{d\Delta^*(r_k, p_k + d_k)}{dp_k} > 0 \tag{5.12}$$

So, with Equation (5.5), we have:

$$\frac{\partial^2 L}{\partial p_k^2} = \frac{T}{p_k} \cdot \frac{d\Delta^*(r_k, p_k + d_k)}{dp_k} > 0 \tag{5.13}$$

It is obvious that, for all $1 \leq k, l \leq NR$, $k \neq l$,

$$\frac{\partial^2 L}{\partial p_k \partial p_l} = 0 \tag{5.14}$$

Therefore, $\nabla_{pp}^2 L(p^*, \mu^*) > 0$. □

⁵ $\Delta^*(\cdot)$ is defined in Chapter 3. We assume after each position update is received, the spatial difference between the replica and its avatar, i.e., $\Delta(r_k, t)$ grows in the same manner following an increasing function $\Delta^*(\cdot)$

Note that, here p^* is a local minimum and it can be also proven to be a global minimum due to the convex property of the $\mathcal{P}2$ ⁶. Now, let us assume for any server s_i , there exists at least one replica whose target server and contact servers both are s_i ⁷.

Assumption 5.4 For $\forall s_i, 1 \leq i \leq NS$, $\{r_k | 1 \leq k \leq NR, |s_t(r_k)| = |s_c(r_k)| = i\} \neq \emptyset$ is true.

Suppose the above assumption holds, based on Theorem 5.3, the following lemma can be proven to be true.

Lemma 5.3 If p^* is a local minimum for $\mathcal{P}2$, assume Assumption 5.4 holds, it follows that there exists $\mu^* = [\mu_1^*, \dots, \mu_{NS}^*]$ such that

$$\begin{aligned} \nabla_p L(p^*, \mu^*) &= 0, \text{ and} \\ \mu_i^* &> 0 \text{ and } g_i(p^*) = 0, \quad \forall i = 1, \dots, NS \end{aligned} \quad (5.15)$$

Proof: To prove this lemma, we just need to prove (5.15) is true. Under Assumption 5.4, for $\forall s_i, 1 \leq i \leq NS$, we can always find an r_k ($1 \leq k \leq NR$) satisfying $|s_t(r_k)| = |s_c(r_k)| = i$. Based on Theorem 5.3, there exists μ^* such that (p^*, μ^*) satisfies (5.5) and (5.6). Hence, according to (5.5), (5.9) and (5.10), it follows:

$$p_k^* \cdot \Delta^*(r_k, p_k^* + d_k) - \int_{d_k}^{p_k^* + d_k} \Delta^*(r_k, t) dt - \alpha \cdot \mu_i^* = 0 \quad (5.16)$$

Define function $\eta(t)$ as: $\eta(t) = t \cdot \Delta^*(r_k, t + d_k) - \int_{d_k}^{t+d_k} \Delta^*(r_k, s) ds$. For $t > 0$, $\frac{d\eta(t)}{dt} = t \cdot \frac{d\Delta^*(r_k, t+d_k)}{dt} > 0$. This means $\eta(t)$ is increasing with t . Since $\eta(0) = 0$, it follows that $\eta(p_k) > 0$, i.e., $p_k \cdot \Delta^*(r_k, p_k + d_k) - \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt > 0$. So, according to (5.16), $\mu_i^* > 0$. Finally, according to (5.6), it follows $g_i(p^*) = 0$. $\square$

Based on Theorem 5.3 and Lemma 5.3, when Assumption 5.4 holds, we have the following theorem:

⁶A proof of the convex property is presented in Chapter 6.

⁷Note that this assumption is not overly-restrictive and can be easily satisfied in DVE by migrating clients to different servers.

Theorem 5.5 *Let p^* be such that $g_i(p^*) \leq 0$, for $1 \leq i \leq NS$, and also assume Assumption 5.4 holds. p^* is a local minimum for $\mathcal{P}2$ **iff** there exists Lagrange Multipliers vector $\mu^* = [\mu_1^*, \dots, \mu_{NS}^*]$ such that for $1 \leq k, l \leq NR$ and $k \neq l$,*

$$\begin{aligned} & \frac{1}{\varphi(r_k, \mu^*)} \cdot \left(p_k^* \cdot \Delta^*(r_k, p_k^* + d_k) - \int_{d_k}^{p_k^* + d_k} \Delta^*(r_k, t) dt \right) \\ &= \frac{1}{\varphi(r_l, \mu^*)} \cdot \left(p_l^* \cdot \Delta^*(r_l, p_l^* + d_l) - \int_{d_l}^{p_l^* + d_l} \Delta^*(r_l, t) dt \right) \end{aligned} \quad (5.17)$$

and

$$\mu_i^* > 0 \quad \text{and} \quad g_i(p^*) = 0, \quad \forall i = 1, \dots, NS \quad (5.18)$$

Proof: For necessity, if p^* is a local minimum, according to Lemma 5.3, there exists μ^* satisfying (5.5) and (5.15) (which is the same as (5.18)). Then, according to (5.5) and (5.10), for every $1 \leq k \leq NR$, it follows $p_k^* \cdot \Delta^*(r_k, p_k^* + d_k) - \int_{d_k}^{p_k^* + d_k} \Delta^*(r_k, t) dt - \varphi(r_k, \mu^*) = 0$, that is,

$$\frac{1}{\varphi(r_k, \mu^*)} \cdot \left(p_k^* \cdot \Delta^*(r_k, p_k^* + d_k) - \int_{d_k}^{p_k^* + d_k} \Delta^*(r_k, t) dt \right) = 1 \quad (5.19)$$

Therefore, for $k \neq l$, (5.17) can be proven to be true.

For sufficiency, let p^* be such that $g_i(p^*) = 0$. Suppose there exists $\mu^* > 0$ satisfying (5.17). Let $\frac{1}{\varphi(r_k, \mu^*)} \cdot (p_k^* \cdot \Delta^*(r_k, p_k^* + d_k) - \int_{d_k}^{p_k^* + d_k} \Delta^*(r_k, t) dt) = \phi$, $k = 1, \dots, NR$, and $\mu' = \mu^* \cdot \phi$. From (5.9), it follows $\varphi(r_k, \mu') = \phi \cdot \varphi(r_k, \mu^*)$. Thus, we have $p_k^* \cdot \Delta^*(r_k, p_k^* + d_k) - \int_{d_k}^{p_k^* + d_k} \Delta^*(r_k, t) dt - \varphi(r_k, \mu') = 0$, $k = 1, \dots, NR$. According to (5.10), it follows $\nabla_p L(p^*, \mu') = 0$. For every $1 \leq i \leq NS$, $\mu'_i \cdot g_i(p^*) = 0$ is an obvious conclusion. Therefore, according to Theorem 5.3, p^* is a local minimum. $\square$

We define a vector μ satisfying Theorem 5.5 as an optimal μ for $\mathcal{P}2$. Theorem 5.5 implies that to minimize total time-space inconsistency of all replicas, we need to find a Lagrange Multipliers vector μ and an update schedule that satisfy both (5.17) and (5.18).

The above analysis is only for the ideal situation that the configurations of the DVE and system properties keep unchanged. In practical systems, the configurations of a DVE and the system properties could vary over time. Let $\mu(t_{now})$ denote the

optimal μ for $\mathcal{P}2$ at time t_{now} . For each replica r_k , we replace the update period by the time elapsed since its last update, the left-hand side of equation (5.17) can be rewritten as:

$$\frac{1}{\varphi(r_k, \mu(t_{now}))} \cdot \left((t_{now} - t_{last(k)}) \cdot \Delta(r_k, t_{now} + d_k) - \int_{t_{last(k)} + d_k}^{t_{now} + d_k} \Delta(r_k, t) dt \right) \quad (5.20)$$

where $t_{last(k)}$ is the time of last position update of replica r_k sent by r_k 's target server before t_{now} .

According to (5.17), in order to obtain a local minimum for $\mathcal{P}2$, the update schedules should attempt to minimize the differences of (5.20) over all replicas at times when servers update them. In most cases, it is reasonable to assume that the configurations of a DVE do not change rapidly relative to the time scale at which updates occur. Since the optimal μ is determined by the configurations of the DVE, it implies that the optimal μ changes little during two consecutive updates. According to (5.9), we have $\varphi(r_k, \mu(t)) \approx \varphi(r_k, \mu(t_{now}))$ for all $t_{last(k)} \leq t \leq t_{now}$. Under this reasonable approximation, the term (5.20) can be proven to be monotonically increasing with time between two successive updates. So, in order to minimize the differences (5.20) between different replicas, once there is available bandwidth, the replica with the highest value of (5.20) should be updated first. Therefore, the value of (5.20) can be defined as the *update priority* for replica r_k at time t_{now} . Note that $\Delta^*(\cdot)$ has been replaced by $\Delta(\cdot)$ in (5.20), which implies that we have dropped the Assumption 3.1.

5.1.2 Measurement of Update Priority

Consider replica r , to calculate the update priority at time t_{now} (i.e., the value of (5.20)), the values of $\Delta(r, t_{now} + d)$, $\int_{t_{last} + d}^{t_{now} + d} \Delta(r, t) dt$ and $\varphi(r, \mu(t_{now}))$ should be measured by r 's target server.

5.1.2.1 How to Measure $\Delta(r, t_{now} + d)$

For a replica of an avatar, $\Delta(r, t_{now} + d)$ refers to the spatial divergence between the position of the replica at client side and the position of the primary copy of the avatar

at server side at time $t_{now} + d$, where d is the transmission delay of the position update from target server to the replica. At time t_{now} , to measure $\Delta(r, t_{now} + d)$, we need to know future positions (positions at $t_{now} + d$) of the replica and the avatar. Let $\vec{P}_a(t)$ denote the position of the avatar at time t at server side and $\vec{P}_r(t)$ denote the position of the replica at time t at client side. Then, $\Delta(r, t_{now})$ is represented by $||\vec{P}_a(t_{now}), \vec{P}_r(t_{now})||$, which is Euclidean distance between two positions. After each position update is received, the client maintains the latest position of the avatar till the next update. It implies that $\vec{P}_r(t_{now}) = \vec{P}_a(t_{last})$, where t_{last} is the time of the update last sent by the server before $t_{now} - d$. As a result, $\vec{P}_r(t_{now} + d)$ is equal to $\vec{P}_a(t_{last})$ which can be collected by target server. Many prediction models can be used to estimate $\vec{P}_a(t_{now} + d)$ according to the current state of the avatar. For instance, if first-order prediction model is adopted, $\vec{P}_a(t_{now} + d)$ can be estimated by $\vec{P}_a(t_{now} + d) = \vec{P}_a(t_{now}) + d \cdot \frac{d\vec{P}_a(s)}{ds}|_{s=t_{now}}$. So far, the values of $\vec{P}_r(t_{now} + d)$ and $\vec{P}_a(t_{now} + d)$ both can be collected or predicted by target server at time t_{now} , thus, $\Delta(r, t_{now} + d)$ can be measured by $||\vec{P}_a(t_{now} + d), \vec{P}_r(t_{now} + d)||$.

5.1.2.2 How to Measure $\int_{t_{last}+d}^{t_{now}+d} \Delta(r, t) dt$

Figure 5.1: Relations between t_{now} and $t_{last} + d$

To calculate $\int_{t_{last}+d}^{t_{now}+d} \Delta(r, t) dt$, all values of $\Delta(r, t)$ between $[t_{last} + d, t_{now} + d]$ need to be collected. If $t_{now} > t_{last} + d$ (as shown in Figure 5.1(a)), the values of $\Delta(r, t)$ from $t_{last} + d$ to t_{now} can be collected by target server. Since the target server maintains the source avatar, the target server should know the past positions of every replica. While

the values of $\Delta(r, t)$ from t_{now} to $t_{now} + d$ can be estimated using the method described in Section 5.1.2.1. Otherwise, if $t_{now} \leq t_{last} + d$ (as shown in Figure 5.1(b)), which means the latest update sent at time t_{last} has not been received by the replica at time t_{now} , in this case, all the values of $\Delta(r, t)$ from $t_{last} + d$ to $t_{now} + d$ need to be estimated using the method that has been presented in Section 5.1.2.1. $\int_{t_{last}+d}^{t_{now}+d} \Delta(r, t) dt$ can be calculated using many existing techniques like Romberg method [75].

5.1.2.3 How to Measure $\varphi(r, \mu(t_{now}))$

Figure 5.2: The measurement of optimal μ

For replica r , $\varphi(r, \mu(t_{now}))$ refers to a constant value, where $\mu(t_{now})$ is the optimal μ of the DVE at time t_{now} . As mentioned above, $\mu(t_{now})$ can be approximated through many gradient descent methods. However, the optimal μ is only for a specific configuration setting. The configurations of the DVE vary over time in a practical system. The gradient descent algorithms to find $\mu(t_{now})$ are usually executed in a centralized manner and global knowledge is required. So, it is impractical to calculate an optimal μ and broadcast it to all servers at each frame due to large communication overhead. Since it is reasonable to assume that the configurations of a DVE do not change rapidly relative to the time scale at which updates occur, the optimal μ should not change abruptly. Thus, the measurement of the optimal μ can be divided into two steps. As shown in Figure 5.2, in the first step, an optimal μ is measured in a

centralized manner through gradient descent methods. This is executed only once at the start of the update scheduling algorithm. After that, in the second step, each μ_i is regulated periodically by server s_i in a distributed manner to adapt to the gradual change of DVE configurations. After each regulation, servers exchange the new values to each other.

The distributed regulation algorithm is inspired by the first-order Lagrange iterative method. Based on the previous conclusions and the optimization theory, we have the following theorem.

Theorem 5.6 *Suppose p^* is a local minimum and μ^* is the corresponding optimal μ , there exists $\theta > 0$, the sequence (p^k, μ^k) generated by (5.21) and (5.22) converges at least linear to (p^*, μ^*) .*

$$p^{k+1} = p^k - \theta \nabla_p L(p^k, \mu^k) \quad (5.21)$$

$$\mu^{k+1} = \mu^k + \theta \nabla_\mu L(p^k, \mu^k) \quad (5.22)$$

Proof: According to the optimization theory [8], we need to prove that: (1) $\nabla_p L(p^*, \mu^*) = 0$; (2) $z' \nabla_{pp}^2 L(p^*, \mu^*) z > 0, \forall z \in R^n$; (3) the gradients $\nabla g_1(p^*), \dots, \nabla g_{NS}(p^*)$ are linearly independent. Based on the previous conclusions, (1) and (2) can easily be proven and we just need to prove condition (3). $\nabla g_i(p)$ is a vector with dimension $1 \times NR$. When Assumption 5.4 holds, for every $g_i(p)$, $1 \leq i \leq NS$, we can always find a r_j whose target server and contact server are the same and it follows that

$$\frac{\partial g_i(p)}{\partial p_j} \Big|_{p_j^*} = \frac{-\alpha}{p_j^{*2}} \quad (5.23)$$

which will appear at the j th column in vector $\nabla g_i(p^*)$. p_j refers to the update period of replica r_j . Position update of r_j does not need to be forwarded by other servers and no bandwidth is required from other servers for updating r_j . Therefore, p_j will not appear in any other $g_k(p)$ ($k \neq i$), which means the elements in the j th column of other $\nabla g_k(p^*)$ are 0. If so, $\nabla g_i(p^*)$ can not be linear represented by other vectors. Therefore, $\nabla g_1(p^*), \dots, \nabla g_{NS}(p^*)$ are linearly independent and the theorem is proved. $\square$

For each μ_i , $1 \leq i \leq NS$, Equation (5.22) can be written as $\mu_i^{k+1} = \mu_i^k + \theta g_i(p^k)$. Based on this conclusion, the distributed regulation algorithm is designed as follows: Let μ_i^{last} denote the value of μ_i before regulation and $p(t_{now})$ denote the estimated value of p at time t_{now} . In the regulation at time t_{now} , the new value of μ_i (denoted as μ_i^{new}) is calculated by

$$\mu_i^{new} = \mu_i^{last} + \theta g_i(p(t_{now})) \quad (5.24)$$

The update period of a replica whose target server or contact server is s_i at time t_{now} can be estimated by s_i based on the last few position updates of the replica. Therefore, $g_i(p(t_{now}))$ can be estimated according to the local information at s_i . So, the regulation is fully distributed. Instead of calculating the optimal μ by using global knowledge, the distributed regulation makes the μ move towards the optimal value gradually. After each run, server s_i ($1 \leq i \leq NS$) exchanges new values of μ_i with other servers so that the new μ can be used in the scheduling. The exchanged value can be easily piggybacked with update messages without incurring too much communication overhead.

This approximate measurement is based on the reasonable assumption that optimal μ does not change rapidly over time. If the DVE is reconfigured for some reasons (e.g., workload re-balance), the gradient descent algorithm in the first step of measurement (see Figure 5.2) needs to be executed to re-initialize the optimal μ .

5.1.3 Update Schedules in Practical System

Based on the update priority we have defined for each replica, we have developed an update algorithm to be used in practical systems. Ideally, if global knowledge is available, a centralized heuristic update algorithm (referred to as C-LMH) can be easily developed as follows: each time the replica with the highest update priority is selected to send update; and if its target server does not have enough bandwidth to perform the update or the contact server does not have enough bandwidth to forward the update, then the replica with second highest update priority is selected instead, and so on. However, centralized algorithm requires global knowledge, which

is infeasible in practice. So, we proposed a simple and effective algorithm named Lagrange Multiplier Based Heuristic Algorithm (LMH) to schedule updates. It runs on each server in a distributed manner.

Algorithm 4 Description of LMH

```

1: At each update frame, for  $\forall s_i, 1 \leq i \leq NS$ 
2: if  $s_i$  has enough bandwidth to send and forward position updates then
3:   Send position updates according to forwarding requests received from other
   servers
4:   Disseminate position updates to the replicas whose target server and contact
   server both are  $s_i$ 
5:   Send forwarding request for the replicas whose target server is  $s_i$  and contact
   server is  $s_j, j \neq i$ 
6: else
7:   Calculate update priority for every replica whose target server is  $s_i$  according
   to (5.20)
8:   Re-estimate update priority for every forwarding request received from other
   servers
9:   while Network capacity is available do
10:    Select the item with highest update priority
11:    if Selected item is a forwarding request then
12:      Send position update to the replica based on forwarding request
13:    else
14:      if Position update of the selected replica needs to be forwarded then
15:        Send forwarding request to the contract server
16:      else
17:        Send position updates to the replica directly
18:      end if
19:    end if
20:  end while
21: end if

```

As shown in Algorithm 4, if a server has enough upload bandwidth, all position updates are disseminated or forwarded straightforwardly (lines 2-5). Otherwise, each server calculates update priority for replicas whose target server is this server and re-estimate update priority for every received forwarding request (lines 7-8). Then, the items with higher update priority, which could be a direct update or update based on forwarding request, are selected to allocate bandwidth (lines 9-20).

Note that for each server s_i , the update decisions are made based only on the update priority ranking of the replicas whose target server is s_i according to the local knowledge of the server. Therefore, the LMH may produce different schedules

from the C-LMH, where the decision to update a replica is made based on the global rankings of update priorities and the global knowledge of all the servers. Because of this, when a forwarding request is received by a contact server, it may not be selected to allocate bandwidth immediately since the update priority of the forwarding request calculated by replica's target server may not be high enough in the local rankings of update priorities at the contact server. As the time increases, update priority of the held forwarding request will be increased because time-space inconsistency increases with time. Therefore, at each update frame, contact server needs to re-estimate the update priority for those held forwarding requests (line 8). For a replica whose position update needs to be forwarded, contact server is supposed to know $\Delta(\cdot)$ of the replica and the last update sent to the replica. At time t_{now} , if a forwarding request of this replica is being detained at contact server, the new update priority of the forwarding request can be approximately measured by contact server according to (5.20), where $t_{last(k)}$ should be replaced by the time the update, last forwarded by the contact server, was generated at the target server.

5.2 Simulation and Experiments

5.2.1 Update Algorithms to Compare

In order to show the performance of the LMH, some other updating strategies were also investigated for MSDVEs. These include the TIME and the SPACE algorithms. In addition, the priority calculation used in the TotalTS [87], which is the optimal update strategy for minimizing total time-space inconsistency in single server DVEs, was also adopted in an implementation of an update strategy for MSDVEs. All those update algorithms have similar procedures to the LMH as shown in Algorithm 4. However, they calculate update priority in different ways. The details are shown as follows.

5.2.1.1 TIME Algorithm

The TIME takes the time since the last update as the update priority. For replica r , at time t_{now} , the update priority is directly set by $t_{now} - t_{last}$, where t_{last} is the time

of last update of replica r before t_{now} .

5.2.1.2 SPACE Algorithm

Different from the TIME, replicas with larger spatial difference between the replica and source avatar has higher priority in the SPACE algorithm. Since the update sent by target server at time t_{now} would not affect the replica in the client's view until time $t_{now} + d$, the SPACE algorithm estimates the expected spatial difference between the replica and its source avatar at time $t_{now} + d$, i.e., $\Delta(r, t_{now} + d)$, where d is the transmission delay of the position update from target server to the replica.

5.2.1.3 TotalTS Algorithm

The TotalTS update algorithm is the optimal updating algorithm for minimizing total time-space inconsistency over all replicas in single server DVEs [87]. To apply the TotalTS to MSDVEs, the update priority of each replica is given by $(t_{now} - t_{last}) \cdot \Delta(r, t_{now} + d) - \int_{t_{last}+d}^{t_{now}+d} \Delta(r, t) dt$, where t_{last} is the time of the last update sent before t_{now} .

The update priority of forwarding request in the TIME, SPACE and TotalTS also needs to be re-estimated at contact server. The methods are the same as that used in the LMH, which has been discussed in Section 5.1.3.

5.2.1.4 RANDOM Algorithm

In order to show the importance of update schedules, we implemented an update algorithm without any scheduling, which is referred to as RANDOM. In the RANDOM, at each update frame, replicas are randomly selected to allocate bandwidth. By assuming the global knowledge is available, we also implemented the centralized algorithm C-LMH as described in Section 5.1.3. Since the replicas are updated according to the global update priority, the C-LMH gives a lower bound of the LMH.

5.2.2 Experimental Setup

We modified the event-driven simulator that has been mentioned in Section 4.3 to simulate the MSDVE and implemented all the update algorithms described in the

previous section. The basic experimental settings are the same as those described in Section 4.3.1, Chapter 4.

There are total 3000 avatars simulated in a virtual world with size 5000×5000 distance units. The virtual world is partitioned into 100 fixed zones which are maintained by 10 servers. The avatars move around the virtual world with random and clustered mobility models. A client is initially connected to the server that maintains its avatar with probability p . If an avatar moves from a region to another one which is maintained by a different server, the associated client will be migrated to the new server with a probability p or stay unchanged otherwise. The speed of an avatar is randomly assigned from a uniform distribution between $[0.1, 10]$ distance units/s.

In this experiment, zones are randomly mapped to servers. Avatars are initially randomly distributed in the virtual world. The network delay between two nodes is still modeled by the shifted exponential distribution with a probability density function (see Section 4.3.1).

A variety of simulations were conducted to evaluate the performance of all update algorithms on the impact of network latency, server upload bandwidth, and the probability p . For each experiment, the average value of the total time-space inconsistency over a period of 10 minutes after an initial warm-up period was calculated after repeating the same experiment several times. The default values of the parameters are shown in Table 4.2.

5.2.3 Experimental Results

5.2.3.1 Determine Distributed Regulating Frequency

Firstly, we try to determine an appropriate frequency for regulating optimal μ in our simulation. For each experiment, the frequency of regulating is varied in a large range from 1 to 10000 frames per-time. The other parameters are set as shown in Table 4.2. As can be seen from Figure 5.3, when avatars move with random mobility model, the impact of regulating frequency is insignificant. This is because the workload is distributed among servers uniformly and the optimal μ changes very little over time. By contrast, when avatars move with clustered mobility model, the total inconsistency

Figure 5.3: Performance with respect to regulating frequency

increases obviously when the regulating interval is larger than 100 frames. It implies that when the workload is imbalanced among servers, frequent regulation results in less inconsistency. Based on the results, in the following experiments, the regulating interval is set to 100 frames.

5.2.3.2 Performance with respect to Upload Bandwidth

The impact of server side upload bandwidth is examined. For each experiment, the average upload bandwidth is set in a range from 100 to 1000 units per-server per-frame while other parameters are set as shown in Table 4.2.

Figure 5.4(a) shows the results of the total time-space inconsistency as a function of different upload bandwidth for all update algorithms with random mobility model. As can be seen, lower upload bandwidth makes larger inconsistency. This is because the bandwidth constraints are more serious with lower network bandwidth. The tested updating algorithms perform similarly when the upload bandwidth is more than 1000. This is because all replicas can be updated frequently in this case.

We also examined the performance of the proposed algorithms with clustered mobility model. As can be seen from Figure 5.4(b), compared to the situation of random mobility model, the LMH achieves more benefits against other algorithms. It implies that when the workload is varied among servers, the LMH can allocate network bandwidth more efficiently than other algorithms. The C-LMH performs

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 5.4: Performance with respect to upload bandwidth capacity

better than LMH when the upload bandwidth is varied. Since each server selects replicas to update according to its local rankings of update priority in the LMH, when workload is imbalanced amongst servers, the replicas selected to update by the LMH and the C-LMH at each update frame may be different.

In all the experiments, the RANDOM results in very large time-space inconsistency compared to other updating algorithms with clustered mobility model. This

indicates the necessity and effectiveness of the update schedules on the reduction of the inconsistency for a DVE.

5.2.3.3 Performance with respect to Network Delays

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 5.5: Performance with respect to network latency ($\nu=0.95$)

Next, we examined the impact of network delay and the variance of network delay on the performance of the proposed algorithms. For each experiment, we let the

average network latency (d) change in the range from 0 to 0.5s and other parameters are set as shown in Table 4.2.

To simulate a network with a small variance of latency, we set $\nu = 0.95$ first. As can be seen from Figures 5.5(a) and 5.5(b), the total inconsistency gets larger as the average latency increases. This is because due to network delays, when position update arrives at the client side, avatars' actual positions at the server side would have already changed. Therefore, time-space inconsistency is affected by network delays and generally increases with the delays. We also notice that the most affected algorithm is the SPACE. This is because the calculation of the update priority in the SPACE, i.e., $\Delta(r, t_{now} + d)$ is highly dependent on the network latency.

We then set $\nu = 0.5$ to simulate a large variance of network latency. Figures 5.6(a) and 5.6(b) show the performance of the proposed algorithms. The time-space inconsistency achieved by each algorithm becomes larger compared to the results with $\nu = 0.95$ for the same network latency setting. This is because time-space inconsistency is increasing superlinear with network latency. Large network latency as the result of large variance of network delay may result in large time-space inconsistency.

5.2.3.4 Performance with respect to Probability p

At last, the impact of the probability p was examined. For each experiment, we varied the probability p in the range from 1 to 0.5 and examined the performance of the proposed algorithms both for random and clustered mobility models. Other parameters are set as shown in Table 4.2.

Intuitively, as the probability p decreases, more clients will be connected to servers that do not maintain their avatars. More position updates need to be forwarded through contact servers, which makes the bandwidth contention more serious and thus results in the increase of the inconsistency as shown in Figure 5.7(a) and Figure 5.7(b). Moreover, the LMH can allocate network bandwidth more efficiently in terms of inconsistency reduction compared to other distributed algorithms for clustered mobility model.

(a) Random Mobility Model

(b) Clustered Mobility Model

 Figure 5.6: Performance with respect to network latency ($\nu=0.5$)

5.3 Summary

In this Chapter, we investigate the updating algorithms for minimizing the total time-space inconsistency. We analyze the problem $\mathcal{P}2$ by using Lagrange Multipliers for a DVE in an ideal situation. The results are then used to develop the updating algorithm in practical systems.

From all the experiments, the LMH always performs much better than other up-

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 5.7: Performance with respect to probability p

dating algorithms for clustered mobility model. The experimental results imply that the LMH can allocate the limited resources much more efficiently than other updating algorithms for reducing time-space inconsistency.

Chapter 6

Zone Mapping and Client Assignment for Minimizing Total Time-space Inconsistency

In this chapter, we investigate the zone mapping and client assignment algorithms for minimizing total time-space inconsistency in MSDVEs. The problem has been formulated as $\mathcal{P3}$ in Chapter 3 under the assumption that except for zone mapping and client assignment, other DVE configurations and system properties keep unchanged.

The problem $\mathcal{P3}$ has been defined in Chapter 3 as the following mixed integer programming problem:

$$\min_{p, C, Z} f(p, C, Z) = \sum_{k=1}^{NR} \frac{T}{p_k} \cdot \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt \quad (6.1)$$

constrained by

$$g_i \leq 0, 1 \leq i \leq NS \quad (6.2)$$

g_i is defined as follows:

$$\begin{aligned} & \sum_k z_{S|z(r_k)|,i} \cdot c_{S|c(r_k)|,i} \cdot \frac{\alpha \cdot T}{p_k} + \sum_l \sum_k z_{S|z(r_k)|,i} \cdot c_{S|c(r_k)|,l} \cdot \frac{\beta \cdot T}{p_k} + \\ & \sum_l \sum_k z_{S|z(r_k)|,l} \cdot c_{S|c(r_k)|,i} \cdot \frac{\alpha \cdot T}{p_k} + \sum_l \sum_j z_{S|z(a_j)|,l} \cdot c_{Sj,i} \cdot u_j \cdot \frac{T}{f_r} \leq B_i \cdot \frac{T}{f_r} \end{aligned} \quad (6.3)$$

where $1 \leq l \leq NR, l \neq i, 1 \leq k \leq NR, 1 \leq j \leq NC$.

The notations in the problem definition are listed again in Table 6.1. The objective function $f(p, C, Z)$ represents the total time-space inconsistency of all replicas

over a period T for a given zone mapping solution Z , client assignment solution C , and update periods of all replicas. $g_i(p) \leq 0$ is the upload bandwidth constraint at server s_i for $1 \leq i \leq NS$. $\sum_k z_{s|z(r_k)|,i} \cdot cs_{|c(r_k)|,i} \cdot \frac{\alpha \cdot T}{p_k}$ represents the upload bandwidth consumption at s_i for updating the replicas whose target server and contact server both are s_i . $\sum_l \sum_k z_{s|z(r_k)|,i} \cdot cs_{|c(r_k)|,l} \cdot \frac{\beta \cdot T}{p_k}$ is the upload bandwidth consumption at s_i for sending forwarding request for the replicas whose target server is s_i and contact server is different. $\sum_l \sum_k z_{s|z(r_k)|,l} \cdot cs_{|c(r_k)|,i} \cdot \frac{\alpha \cdot T}{p_k}$ represents the upload bandwidth consumption for sending position updates according to the received forwarding requests. $\sum_l \sum_j z_{s|z(a_j)|,l} \cdot cs_{j,i} \cdot u_j \cdot \frac{T}{f_r}$ is the upload bandwidth consumption for relaying user's input commands. The total upload bandwidth is constrained by $B_i \cdot \frac{T}{f_r}$.

 Table 6.1: Notations for $\mathcal{P3}$

p_k	update period of r_k , $1 \leq k \leq NR$.
d_k	transmission delay of the position update from r_k 's target server to r_k (may be forwarded by other server).
Z	a matrix with variables $[zs_{j,i}]$ that represents a zone mapping solution.
C	a matrix with variables $[cs_{j,i}]$ that represents a client assignment solution.
$z(r_k)$	short for $z(a(r_k))$, which represents the zone where the source avatar of r_k is residing.
$c(r_k)$	the client that maintains r_k .
$z(a_j)$	the zone where avatar a_j is residing.
u_j	the upload bandwidth consumption for relaying c_j 's input commands.
$zs_{j,i}$	$zs_{j,i} = 1$ if zone z_j is mapped to server s_i or $zs_{j,i} = 0$ otherwise.
$cs_{j,i}$	$cs_{j,i} = 1$ if client c_j is connected to server s_i or $cs_{j,i} = 0$ otherwise.
$ \cdot $	the function to map an item to index.

Based on $\mathcal{P3}$, the zone mapping and client assignment algorithms for the practical systems are investigated by two stages. In the first stage, an initial (C, Z) is determined by a centralized algorithm at the beginning of the DVE system running. Then, in the second stage, an adaptive tuning algorithm is used to make change on (C, Z) during execution.

The initial (C, Z) is obtained by solving $\mathcal{P3}$, which is to minimize the time-space inconsistency of the DVE at the moment of system starts. A straightforward method for solving it is to enumerate all possible values for C and Z , and to find the optimal

p for each case. However, when the number of avatars is very large, the enumeration is impossible. A branch and bound method, or other global optimization technique, can be used to speed up the computation of the globally optimal solution. But the worst case complexity of any method that computes the global optimal solution is exponential to the size of C and Z [13]. For this reason, we need to consider heuristics for solving the problem approximately. In this thesis, the original problem is decomposed into two sub-problems by using Alternating Optimization (AO) method. For each of the sub-problems, the corresponding solution is proposed.

6.1 Alternating Optimization

Alternating Optimization is a generic methodology for locating the solution of an optimization problem by partitioning and treating independently the design variables [9]. The principle advantage of AO is that it replaces the optimization of the objective function with a sequence of easier optimizations involving different partitions of the design variables.

In this thesis, the idea of AO is used to solve problem $\mathcal{P}3$ and the unknown variables are divided into two subsets: (C, Z) and p . It generates two easier sub-problems. The first problem (referred to as $\mathcal{P}3-1$) is defined by:

$$\min_p f(p, (C, Z)) \quad (6.4)$$

constrained by (6.2), where (C, Z) is fixed. The second problem (referred to as $\mathcal{P}3-2$) is defined by:

$$\min_{C, Z} f(p, (C, Z)) \quad (6.5)$$

constrained by (6.2), where p is fixed.

The AO algorithm for solving the $\mathcal{P}3$ is described by Algorithm 5. At the beginning, an initial (C, Z) is selected and denoted as (C^0, Z^0) (line 1). Then, the $\mathcal{P}3-1$ is optimized by fixing (C, Z) at (C^0, Z^0) and the obtained global minimum is denoted as p^0 (line 2). Taking $(p^0, (C^0, Z^0))$ as the starting point, the AO algorithm alternates

Algorithm 5 Alternating Optimization Algorithm

- 1: Pick an initial (C, Z) as (C^0, Z^0) . Set a termination tolerance $\epsilon = 1e - 6$ and iterative step $t = 0$.
 - 2: $p^0 = \arg \min_p f(p, (C^0, Z^0))$
 - 3: $(C^{t+1}, Z^{t+1}) = \arg \min_{C, Z} f(p^t, (C, Z))$
 - 4: $p^{t+1} = \arg \min_p f(p, (C^{t+1}, Z^{t+1}))$
 - 5: If $(f(p^t, (C^{t+1}, Z^{t+1})) - f(p^{t+1}, (C^{t+1}, Z^{t+1}))) < \epsilon$ or $(t > MaxStep)$ goto (7)
 - 6: $t = t + 1$, goto step (3)
 - 7: Stop.
-

between optimizing the $\mathcal{P3-2}$ and the $\mathcal{P3-1}$ (lines 3 to 4). The algorithm stops when either the total decrease in $f(p, (C, Z))$ is less than or equal to ϵ or the number of iterative steps exceeds a threshold value $MaxStep$ (line 5).

Suppose the global minimums of the $\mathcal{P3-1}$ and the $\mathcal{P3-2}$ in the iterations (lines 3-4) can be always obtained. Algorithm 5 can be proven to converge at a local minimum of the $\mathcal{P3}$ [9]. We establish the above for $\epsilon = 0$, however, $\epsilon = 1e - 6$ is generally used in practice. Suppose Algorithm 5 converges at $t = k$ step for $\epsilon = 0$, it follows that

$$f(p^{k+1}, (C^{k+1}, Z^{k+1})) = f(p^k, (C^{k+1}, Z^{k+1})) \quad (6.6)$$

On one hand, line 4 implies that for the fixed (C^{k+1}, Z^{k+1}) , p^{k+1} is a global minimum of the $\mathcal{P3-1}$. Since $f(p^{k+1}, (C^{k+1}, Z^{k+1})) = f(p^k, (C^{k+1}, Z^{k+1}))$, when Algorithm 5 converges, it follows that p^k is also a global minimum of the $\mathcal{P3-1}$. On the other hand, according to line 3, (C^{k+1}, Z^{k+1}) is a global minimum of the $\mathcal{P3-2}$ when p is fixed at p^k . Therefore, $(p^k, (C^{k+1}, Z^{k+1}))$ has the following property for the $\mathcal{P3}$, i.e.,

$$\begin{aligned} f(p^k, (C^{k+1}, Z^{k+1})) &= \min_{C, Z} f(p^k, (C, Z)) \\ &= \min_p f(p, (C^{k+1}, Z^{k+1})) \end{aligned} \quad (6.7)$$

According to the definition of local minimum [9], $(p^k, (C^{k+1}, Z^{k+1}))$ is a local minimum of the $\mathcal{P3}$. The technology to pursue a global minimum of the $\mathcal{P3}$ is beyond the scope of this thesis.

6.1.1 Solution of Problem $\mathcal{P3-1}$

$\mathcal{P3-1}$ aims to determine the optimal update period of each replica for minimizing total inconsistency for fixed DVE configurations. We shall show that $\mathcal{P3-1}$ is a convex

optimization problem with inequality constraints. With this property, the global optimum of $\mathcal{P}3-1$ can be approximated efficiently by the gradient descent methods such as Interior Point (IP) method [8].

In order to prove the convex property of $\mathcal{P}3-1$, Equation (6.1) and (6.2) should be convex functions for the fixed (C, Z) . For every $1 \leq k \leq NR$, let $q_k = \frac{1}{p_k}$, Equation (6.1) can be rewritten as:

$$f(q, (C, Z)) = \sum_{k=1}^{NR} T \cdot q_k \cdot \int_{d_k}^{\frac{1}{q_k} + d_k} \Delta^*(r_k, t) dt \quad (6.8)$$

and Equation (6.3) can be rewritten as:

$$\begin{aligned} g_i = & \sum_k z_{S|z(r_k)|,i} \cdot c_{S|c(r_k)|,i} \cdot \alpha \cdot T \cdot q_k + \sum_l \sum_k z_{S|z(r_k)|,i} \cdot c_{S|c(r_k)|,l} \cdot \beta \cdot T \cdot q_k + \\ & \sum_l \sum_k z_{S|z(r_k)|,l} \cdot c_{S|c(r_k)|,i} \cdot \alpha \cdot T \cdot q_k + \sum_l \sum_j z_{S|z(a_j)|,l} \cdot c_{Sj,i} \cdot u_j \cdot T \leq B_i \cdot \frac{T}{f_r} \end{aligned} \quad (6.9)$$

For every $1 \leq k \leq NR$, it follows:

$$\begin{aligned} \frac{\partial f(q, (C, Z))}{\partial q_k} &= T \cdot \left(\int_{d_k}^{\frac{1}{q_k} + d_k} \Delta^*(r_k, t) dt + q_k \cdot \Delta^*(r_k, \frac{1}{q_k} + d_k) \cdot \frac{-1}{q_k^2} \right) \\ &= T \cdot \left(\int_{d_k}^{\frac{1}{q_k} + d_k} \Delta^*(r_k, t) dt - \Delta^*(r_k, \frac{1}{q_k} + d_k) \cdot \frac{1}{q_k} \right) \end{aligned} \quad (6.10)$$

Hence,

$$\begin{aligned} \frac{\partial^2 f(q, (C, Z))}{\partial q_k^2} &= \frac{T \cdot \partial(\int_{d_k}^{\frac{1}{q_k} + d_k} \Delta^*(r_k, t) dt)}{\partial q_k} - \frac{T \cdot \partial(\Delta^*(r_k, \frac{1}{q_k} + d_k) \cdot \frac{1}{q_k})}{\partial q_k} \\ &= T \cdot \Delta^*(r_k, \frac{1}{q_k} + d_k) \cdot \frac{d(\frac{1}{q_k} + d_k)}{dq_k} - T \cdot \left(\frac{1}{q_k} \right)' \cdot \Delta^*(r_k, \frac{1}{q_k} + d_k) \\ &\quad - T \cdot \frac{1}{q_k} \cdot \frac{d\Delta^*(r_k, \frac{1}{q_k} + d_k)}{dq_k} \\ &= -T \cdot \frac{1}{q_k} \cdot \frac{d\Delta^*(r_k, \frac{1}{q_k} + d_k)}{d(\frac{1}{q_k} + d_k)} \cdot \frac{d(\frac{1}{q_k} + d_k)}{dq_k} \\ &= T \cdot \frac{1}{q_k^3} \cdot \left(\frac{d\Delta^*(r_k, \frac{1}{q_k} + d_k)}{d(\frac{1}{q_k} + d_k)} \right) \geq 0 \end{aligned} \quad (6.11)$$

Then, for $\forall m, k \neq m$, it is obvious that

$$\frac{\partial^2 f(q, (C, Z))}{\partial q_k \partial q_m} = 0 \quad (6.12)$$

Therefore, we have $\nabla_{qq}^2 f(q, (C, Z)) \geq 0$ and $f(q, (C, Z))$ is a convex function over q with fixed C and Z . So, the $\mathcal{P3-1}$ is a convex optimization problem and each local minimum is the global minimum. In this thesis, the gradient descent Interior Point (IP) method is used to solve $\mathcal{P3-1}$ in our experimental evaluations.

6.1.2 Solution of Problem $\mathcal{P3-2}$

$\mathcal{P3-2}$ is a 0-1 Integer Programming Problem, which aims to determine (C, Z) for minimizing total inconsistency under the assumption that p has been obtained in Algorithm 5. Enumeration and dynamic programming can be used to get a global minimum. However, the computational complexity is very high especially for very large scale DVEs. Therefore, heuristic algorithm is developed to get an approximate solution. There are two main parts in the heuristic algorithm. In the first part, a greedy algorithm is used to obtain an initial solution of C and Z . Then, a local search algorithm is developed to further improve the solution. The notations will be used in the heuristic algorithm are defined in Table 6.2.

Table 6.2: Notations used in the heuristic algorithm

Zone related notations		Client related notations	
Notation	Definition	Notation	Definition
$R(z_j)$	set of replicas whose avatar is in z_j	$R(c_j)$	set of replicas maintained by c_j
w_j	upload bandwidth consumption for updating $r_k \in R(z_j)$	$w(c_j)$	upload bandwidth consumption for updating all the replicas $r_k \in R(c_j)$
$v_{j,i}$	sum of the time-space inconsistency of all $r_k \in R(z_j)$	$v(c_j, s_i)$	total time-space inconsistency of all the replicas $r_k \in R(c_j)$ when c_j is connected to s_i

For each z_j , let $R(z_j)$ represent the set of replicas whose avatars are in zone z_j , that is, $R(z_j) = \{r_k | z(r_k) = z_j, 1 \leq k \leq NR\}$. Define V as matrix $V = [v_{ji}]$, where v_{ji} is the sum of the time-space inconsistency of all $r_k \in R(z_j)$ (i.e., $\sum_{r_k \in R(z_j)} \frac{T}{p_k} \cdot \int_{d_k}^{p_k+d_k} \Delta^*(r_k, t) dt$) if z_j is assigned to s_i .

Algorithm 6 Greedy Assign Algorithm (GAA)

```

1: Let  $\hat{c}_i$  denote the available upload bandwidth at  $s_i$ ,  $1 \leq i \leq NS$ . Initially,  $\hat{c}_i = B_i$ 
2: Calculate  $w_j$  for  $1 \leq j \leq NZ$ 
3: Calculate  $v_{ji}$  for  $1 \leq j \leq NZ, 1 \leq i \leq NS$ 
4: for each assignment iteration do
5:   Denote  $\frac{v_{j^*i^*}}{w_{j^*}}$  as the minimum among all  $\frac{v_{ji}}{w_j}$ s such that: (1)  $z_j$  has not been
      assigned; (2)  $\hat{c}_i \geq w_j$ ;
6:   Assign  $z_{j^*}$  to  $s_{i^*}$ 
7:    $\hat{c}_{i^*} = \hat{c}_{i^*} - w_{j^*}$ 
8: end for
9: Let  $z_{L[1]}, \dots, z_{L[n]}$  denote the zones that have not been assigned
10: for Each  $z_{L[k]}$  from  $1 \leq k \leq n$  do
11:   Assign  $z_{L[k]}$  to  $s_i$  with maximum  $\hat{c}_i$  for  $1 \leq i \leq NS$ 
12:    $\hat{c}_i = \hat{c}_i - w_{L[k]}$ 
13: end for

```

In the greedy algorithm, we let each client be connected to the server that maintains the zone where the client's avatar is in, i.e., $zs_{|z(r_k)|,i} = cs_{|c(r_k)|,i}$ for all r_k . In this manner, no matter which server z_j is mapped to, the upload bandwidth consumption for updating $r_k \in R(z_j)$ is the same which is denoted w_j . The problem of mapping zones can be formulated to the following variant of multiple knapsack problem [53]:

$$\min \sum_{j=1}^{NZ} \sum_{i=1}^{NS} zs_{j,i} \cdot v_{j,i} \quad (6.13)$$

subject to

$$\sum_{j=1}^{NZ} w_j \cdot zs_{j,i} \leq B_i, \quad 1 \leq i \leq NS \quad (6.14)$$

$$\sum_{i=1}^{NS} zs_{j,i} \leq 1, \quad zs_{j,i} = 0 \text{ or } 1, \quad 1 \leq j \leq NZ \quad (6.15)$$

The greedy algorithm to assign zones is shown in Algorithm 6. Zones are assigned in an increasing order of the $\frac{v_{j,i}}{w_j}$. In each iteration, the decision to assign a zone z_j to a server s_i is made according to the following conditions (line 5): (1) z_j is not assigned yet; (2) z_j can be assigned to s_i ; (3) z_j and s_i incur the minimum value of $\frac{v_{j,i}}{w_j}$ among all possible zones and servers satisfying (1) and (2). Due to the upload bandwidth constraints, some zones may not be assigned to any server during the iterations. For

Suppose the HAA terminates at $(\bar{p}, (\bar{C}, \bar{Z}))$. Define $q = \frac{1}{p}$, according to the decomposition theory [41], the variables can be divided into two subsets: q and (C, Z) and the problem can be partitioned into two subproblems. The primary problem (referred to as $\mathcal{PP}1$) can be written as follows:

$$\min_q f(q, (\bar{C}, \bar{Z})) \quad (6.16)$$

subject to

$$g_i(q, (\bar{C}, \bar{Z})) \leq 0, i = 1, \dots, NS \quad (6.17)$$

The $\mathcal{PP}1$ has been proven to be a convex optimization problem in Section 6.1.1 and the global optimum of the $\mathcal{PP}1$ can be obtained using gradient descent methods. Suppose the optimum of the $\mathcal{PP}1$ is q^* and let μ^* represent the column vector of Lagrange Multipliers corresponding to the inequality constraints g . Then, the master problem (referred to as $\mathcal{MP}1$) can be stated as follows:

$$\min_{\substack{C, Z \\ \mu_B}} \mu_B \quad (6.18)$$

subject to

$$\begin{aligned} L^*(C, Z; \mu^*) &\leq \mu_B \\ g_i &\leq 0 \end{aligned}$$

where $L^*(C, Z; \mu^*)$ is represented by

$$f(q^*, (C, Z)) + (\mu^*)^T g(q^*, (C, Z)) \quad (6.19)$$

According to [41], the global minimum of the $\mathcal{MP}1$ provides a lower bound for the original problem $\mathcal{P}3$. However, it is still difficult to solve the $\mathcal{MP}1$, thus, some relaxations are further carried out.

The 0-1 integer variables in C and Z are relaxed to continuous variables from 0 to 1, i.e., $0 \leq z_{s_j, i}, c_{s_j, i} \leq 1$. For each $1 \leq k \leq NR$, d_k is redefined as $d_k = \min\{d(s_1, c(r_k)), \dots, d(s_{NS}, c(r_k))\}$.

After these relaxations, the $\mathcal{MP1}$ is rewritten as another problem denoted $\mathcal{RMP1}$. It is obvious that the global minimum of the $\mathcal{RMP1}$ is smaller than that of the $\mathcal{MP1}$. For the $\mathcal{RMP1}$, we use the decomposition theory again and divide the variables (i.e., (C, Z)) into two sets: C and Z . It can be easily proven that for the fixed Z to optimize C or vice versa, the $\mathcal{RMP1}$ is convex optimization problem⁸. Suppose Z is fixed at $\bar{Z}$. For the $\mathcal{RMP1}$, the primal problem (referred to as $\mathcal{PP2}$) takes the form:

$$\min_{\substack{C \\ \mu_B}} \mu_B \quad (6.20)$$

subject to

$$\begin{aligned} L^*(C, \bar{Z}; \mu^*) &\leq \mu_B \\ g_i &\leq 0 \end{aligned}$$

Suppose the optimum of the $\mathcal{PP2}$ is μ_B^* and corresponding Lagrange Multipliers is λ^* . The master problem (referred to as $\mathcal{MP2}$) can be defined as:

$$\min_{\substack{C \\ \mu_C}} \mu_C \quad (6.21)$$

subject to

$$\begin{aligned} \mu_B^* + (\lambda^*)^T [(f(q^*, C, \bar{Z})) + (\mu^*)^T g(q^*, C, \bar{Z}) - \mu_B^*] &\leq \mu_C \\ g_i &\leq 0 \end{aligned}$$

The global minimum of the $\mathcal{MP2}$ should be a lower bound of the $\mathcal{RMP1}$, which is also a lower bound of the $\mathcal{P3}$. Since the $\mathcal{MP2}$ is a convex optimization problem (similar to $\mathcal{RMP1}$), the minimum of the $\mathcal{MP2}$ can be achieved through many gradient descent algorithms like Interior Point method. The minimum of the $\mathcal{MP2}$ is taken as the lower bound of the $\mathcal{P3}$ in our experiments.

6.1.4 Computing Complexity

In this paper, problem $\mathcal{P3-1}$ is solved by interior point methods, which specifically refers to the barrier method. The basic idea of the barrier method is based on solving

⁸The proof of the convex property is similar to the proof for $\mathcal{P3-1}$.

the problem by applying Newton's method to a sequence of equality constrained problems. According to [13], the upper bound of the total number of Newton steps in the barrier method is

$$\left\lceil \frac{\log(m/(t^{(0)}\epsilon))}{\log \mu} \right\rceil \left(\frac{m(\mu - 1 - \log \mu)}{\gamma} + c \right) \quad (6.22)$$

In (6.22), m is the number of inequality constraints and we have $m = NS$ for our problem. ϵ is the accuracy rate which guarantees that we have an ϵ -suboptimal solution of the original problem. $t^{(0)}$, μ , γ and c are some constants [13]. For each Newton step, the computing complexity is $O(NR)$ for our problem. Therefore, the computing complexity of the barrier method in our problem is $O(NR \cdot NS \cdot \log NS)$.

In Algorithm 6, the computing complexity of calculating w_j is $O(NS)$. If the inconsistency of each replica to each server is calculated, v_{ji} can be collected accordingly. Therefore, the computing complexity of calculating v_{ji} is $O(NR \cdot NS)$. The zone to be assigned can be selected with a complexity of $O(\log(NZ \cdot NS))$. The number of iterations in line 4 is at most NZ . Therefore, the computing complexity of the for-loop (lines 4-8) is $O(NZ \cdot \log(NZ \cdot NS))$. Intuitively, the computing complexity of the for-loop from lines 10 to 13 is $O(NZ \cdot \log(NS))$. Therefore, the overall computing complexity of Algorithm 6 is $O(NR \cdot NS) + O(NZ \cdot \log(NZ \cdot NS))$. It is not difficult to get the computing complexity of Algorithm 3 is $O(NC \cdot NS)$.

Suppose the number of iterations of HAA is bounded by M . Moreover, we have $NR \gg NC > NZ > NS$ in the practical system. Therefore, the computing complexity of the HAA can be simplified to $O(M \cdot NR \cdot NS \cdot \log NS)$.

6.2 Adaptive Tuning in Dynamic Systems

The initial (C, Z) obtained by the HAA algorithm in the first stage is only optimal for the moment when the DVE system starts. Since DVE is highly dynamic, the configurations and system properties vary over time in the execution. To keep an efficient mapping, one method is to run the HAA algorithm frequently during system running. However, the HAA algorithm is executed in centralized manner and global

knowledge is required, which makes it impractical to run very frequently due to large computation and communication overhead. Therefore, in the second stage, we investigate an efficient adaptive tuning algorithm to discover the potential improvements and make change on (C, Z) during execution.

The basic idea of the adaptive tuning is to discover the potential zone and client migrations which can reduce the inconsistency. The algorithm is executed in a distributed manner and can be activated at any server based on a threshold. In particular, each server periodically checks the bandwidth “overload” and the algorithm is activated when the overload is larger than 1. The bandwidth overload at server s_i is evaluated by the upload bandwidth demand over the bandwidth capacity, i.e.,

$$\frac{NR(s_i) \cdot \alpha}{B_i} \quad (6.23)$$

, where $NR(s_i)$ is the total number of replicas that have s_i as the target server. If the algorithm is activated, s_i firstly communicates with a subset of (possibly all) servers and selects the server s_k who has the smallest overload. If the overload at s_k is larger than s_i , nothing will be done. Otherwise, three steps will be carried out at s_i .

Let the notations of $R(z_j)$, v_{ji} , $R(c_j)$, $w(c_j)$ and $v(c_j, s_i)$ denote the same meaning as has been defined in Table 6.2. In the first step, s_i attempts to migrate a zone to s_k . In order to do that, s_i randomly selects a zone z_j maintained by s_i . If the condition $v_{jk} < v_{ji}$ is satisfied, i.e., migrating z_j to s_k will achieve less inconsistency, z_j is migrated to s_k and let all the clients whose avatars are in z_j connect to s_k as well.

If z_j is not migrated in the first step, s_i attempts to exchange z_j with a zone maintained by s_k in the second step. Let z_l denote the zone that has the fewest avatars among the zones maintained by s_k . If the condition $v_{ji} + v_{lk} \geq v_{jk} + v_{li}$ is satisfied, i.e., exchanging z_j and z_l will achieve less inconsistency, z_j and z_l are migrated to s_k and s_i respectively. Meanwhile, let the clients whose avatars are in z_j connect to s_k and the clients whose avatars are in z_l connect to s_i .

In the third step, s_i attempts to migrate a client to another server. Let $d(c)$ denote the transmission delay from a client c to the server maintaining the avatar

of c . The client to be migrated is denoted c_j , where $d(c_j)$ is the largest among all the clients connected to s_i . Let s_m denote the server that achieves the smallest $d(c_j)$ if c_j is connected to s_m . If the condition $v(c_j, s_i) > v(c_j, s_k)$ is satisfied, i.e., the inconsistency is decreased if c_j is connected to s_k , c_j is migrated to s_k .

We shall show that the communication overhead (besides migrations) in the three steps is $O(1)$. Suppose the latencies from any client to any server are maintained by each server (the knowledge can be collected from the coordinator when the HAA is executed). For the first step, the value of v_{ji} can be estimated by s_i locally. In order to estimate v_{jk} , we need to know the update periods of the replicas in $R(z_j)$ after z_j is migrated to s_k . We simply assume all the replicas share the same update period, denoted by p , it approximately follows that

$$(NR(s_k) + NR(z_j)) \cdot \alpha \cdot \frac{T}{p} = \frac{T}{f_r} \cdot B_k \quad (6.24)$$

, where $NR(z_j)$ is the number of replicas whose avatars are in z_j . Therefore, only the values of $NR(s_k)$ and B_k are needed to be transmitted from s_k and p can be estimated at s_i by $(NR(s_k) + NR(z_j)) \cdot \alpha \cdot f_r / B_k$. So, the total communication overhead is $O(1)$. Similarly, for the second step, only the values of B_i , B_k , $NR(s_i)$ and $NR(s_k)$ are transmitted and the values of v_{ji} , v_{jk} and v_{lk} , v_{li} can be estimated at s_i and s_k respectively. The communication overhead is $O(1)$. For the third step, the values of $v(c_j, s_i)$ and $v(c_j, s_k)$ can be estimated at s_i . Only the values of B_k and $NR(s_k)$ need to be transmitted, which incurs $O(1)$ communication overhead.

Based on the above analysis, the zone mapping and client assignment can be achieved in the practical systems by the following algorithm (denoted as HAA-AT): (1) When the DVE system starts, the initial (C, Z) is determined by the HAA; (2) During the execution, the adaptive tuning algorithm is used to make change on (C, Z) .

6.3 Simulation and Experiments

In this section, we present experiments for evaluating the performance of the proposed algorithms. Firstly, we simulate a virtual world with ideal movement model

for avatars. Avatars are set to move around circles. Then, in the second experiment, avatars move with more realistic mobility models: random mobility model and clustered mobility model. We modified the event-driven simulator that has been mentioned in Section 4.3 to simulate the MSDVE and implemented all the proposed algorithms.

For both experiments, the basic experimental settings are as follows. The simulated virtual world is with size 5000×5000 distance units and there are 3000 avatars simulated in the virtual world. There are 100 equal sized zones which are maintained by 10 servers. We still consider the position update of avatars and the position update is scheduled by the LMH algorithm (presented in Chapter 5) if there is not enough upload bandwidth to update all the replicas. The network delays between two nodes are still modeled by the shifted exponential distribution as defined in Section 4.3.1.

6.3.1 Experiment 1: Ideal Movement Model

In this experiment, the avatars are set to move around circles. The radii of the circles are randomly assigned from a uniform distribution between 0.1 and 10 distance units. The rotational speeds of avatars are set to be 0.5π per second. Thus, the actual moving speed of an avatar is the product of the radius of the circle and its rotational speed. Using the above mobility model, the value of $\Delta^*(r, t)$ can be easily calculated according to $a(r)$'s trajectory.

Two different methods are used to generate the initial position of each avatar in the virtual world. The methods are (as shown in Figure 6.1):

- Uniform Distribution: The avatars are uniformly distributed in the virtual world.
- Clustered Distribution: Given the square virtual world defined by the two coordinates $[0,0]$ and $[X,Y]$. The avatars are generated around 10 clusters. Firstly, we randomly generate 10 points $(x_1, y_1), \dots, (x_{10}, y_{10})$ which are uniformly distributed in the virtual world. Then, we divide the avatars into 10 equal-sized groups, denoted $G_1, G_2, \dots$, and G_{10} . For each avatar in group G_i , its position

Figure 6.1: Initial distributions of avatars in the virtual world

(x,y) is uniformly distributed within the circle centered on (x_i,y_i) with radius of 1000 distance units. If the generated (x,y) is out of the virtual world (i.e., $x < 0$ or $x > X$ or $y < 0$ or $y > Y$), following adjustments are carried out:

$$x = \begin{cases} 0 & \text{if } x < 0 \\ X & \text{if } x > X \end{cases} \quad y = \begin{cases} 0 & \text{if } y < 0 \\ Y & \text{if } y > Y \end{cases} \quad (6.25)$$

The upload bandwidth at each server is assumed to be equal, which is varied in the range from 100 to 1000 bandwidth units/frame in the experiments. A random mapping algorithm is also implemented for comparison, which is referred to as RANDOM. In the RANDOM, the zones are randomly assigned to servers and clients are connected to servers in the virtual position manner. The zone mapping and client assignment algorithms mentioned in the related work are not compared in our experiments. This is because the comparison may not be meaningful since the objective in our problem definition is totally different. In each experiment, the length of the examined period, i.e., T is set to be 10 minutes after a warm-up period. The total time-space inconsistency is collected as the metric for evaluating the proposed algorithms. The default values of the parameters are shown in Table 4.2.

Since avatars move regularly, the configurations and system properties which could influence the inconsistency can be approximately assumed unchanged in the examined period. In this case, the optimal zone mapping and client assignment of $\mathcal{P3}$ are not

changed in the examined period. The adaptive tuning algorithm in HAA-AT needs not to be executed. For the ideal movement model, a theoretical lower bound of the total time-space inconsistency in the examined period can be obtained by using Lagrange Relaxation method as shown in Section 6.1.3.

6.3.1.1 Performance with respect to Upload Bandwidth

(a) Uniform Distribution

(b) Clustered Distribution

Figure 6.2: Performance with respect to upload bandwidth

Firstly, the impact of upload bandwidth was examined. For each experiment, the

available upload bandwidth is set in a range from 100 to 1000 bandwidth units/frame at each server while other parameters are set as shown in Table 4.2. Figures 6.2(a) and 6.2(b) show the performance of the HAA-AT for two distributions of avatars. As can be seen, lower upload bandwidth makes larger inconsistency. This is because the bandwidth constraints are more serious with lower bandwidth capacity. For the uniform distribution, the performance of the HAA-AT and the RANDOM is close. This is because avatars are uniformly distributed in the virtual world, which results in balanced network traffic among servers. When the avatars are crowded, i.e., with the clustered distribution, more communication workload will be generated by the zones with the crowded avatars. Thus, random assignment will cause imbalanced network traffic among servers. The HAA-AT is able to distribute the network traffic more efficiently among servers. Thus, it outperforms the RANDOM significantly with the clustered distribution.

6.3.1.2 Performance with respect to Network Latency

Then, the impact of network delay was examined on the performance of the proposed algorithms. For each experiment, the average network latency (d) ranges from 0 to 0.5s and other parameters are set as shown in Table 4.2. As can be seen from Figures 6.3(a) and 6.3(b), the total inconsistency gets larger as the average latency increases. This is because time-space inconsistency is affected by transmission delays and generally increases with the delays. Moreover, when the latency becomes larger, the advantage of the HAA-AT becomes more obvious. This is because the mapping of zones and the assignment of clients determines the transmission delays between target server and replicas. The HAA-AT is more powerful to reduce the impact of transmission delays on time-space inconsistency when the average network latency increases.

6.3.1.3 Performance with respect to the Variance of Network Latency

Next, the performance with respect to the variance of network latency was examined of the proposed algorithms. For each experiment, the variance of network latency

(a) Uniform Distribution

(b) Clustered Distribution

Figure 6.3: Performance with respect to network latency

ranges from 1 to 0.5 and other parameters are set as shown in Table 4.2. As can be seen from Figures 6.4(a) and 6.4(b), the total inconsistency gets larger as the variance of network latency increases. However, the growth is not quite obvious and the impact of the variance is insignificant for both uniform and clustered workload distributions. The HAA-AT always performs better than other distributed algorithms for all variance settings.

(a) Uniform Distribution

(b) Clustered Distribution

Figure 6.4: Performance with respect to the variance of network latency

6.3.2 Experiment 2: Random and Clustered Mobility Models

In this experiment, the avatars move around the virtual world with random and clustered mobility models, which are more close to the practical situations. In the experiments, we examined the impact of the upload bandwidth and network latency on the performance of the proposed algorithms. The RANDOM algorithm is also

implemented, where zones are randomly mapped among servers at the beginning and clients are connected to servers in the virtual position manner. No adaptive tuning algorithm is executed in the RANDOM. For each experiment, we run the simulation for 10 minutes after a warm-up and collected the total time-space inconsistency as the metric for evaluating the proposed algorithms.

6.3.2.1 Performance with respect to Upload Bandwidth

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 6.5: Performance with respect to upload bandwidth capacity

Then, the impact of upload bandwidth on the performance of the HAA-AT algorithm was examined. Figure 6.5(a) shows the total time-space inconsistency against the upload bandwidth with random mobility model. Figure 6.5(b) shows the results with clustered mobility model. The total time-space inconsistency gets larger as the available upload bandwidth decreases. This is because the replicas cannot be updated frequently with low upload bandwidth. The HAA-AT algorithm outperforms the RANDOM algorithm greatly for the clustered mobility model.

6.3.2.2 Performance with respect to Network Latency

The impact of network delay on the performance of the HAA-AT was also examined. For each experiment, the average network latency (d) between the simulated clients and servers ranges from 0 to 0.5s with a variance of 0.95. As can be seen from Figures 6.6(a) and 6.6(b), for both random mobility model and clustered mobility model, the total time-space inconsistency becomes larger as the latency increases. Similarly, the HAA-AT outperforms the RANDOM greatly for the clustered mobility model.

6.3.2.3 Performance with respect to the Variance of Network Latency

At last, the performance with respect to the variance of network latency was examined of the proposed algorithms. For each experiment, we let the variance of network latency range from 1 to 0.5 and other parameters are set as shown in Table 4.2. As can be seen from Figures 6.7(a) and 6.7(b), the total inconsistency also increases as the variance of network latency increases. The impact of the variance of network latency on the total inconsistency is also insignificant for both random and clustered mobility models. Moreover, the HAA-AT still performs better than the RANDOM algorithms for all variance settings.

6.4 Summary

In this chapter, the zone mapping and client assignment algorithms for minimizing the total time-space inconsistency are investigated. By setting some assumptions, we

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 6.6: Performance with respect to network latency

formally formulate the problem as a mix integer programming problem. Then, the zone mapping and client assignment algorithms are investigated.

From the above two sets of experiments, we can draw the conclusion that the performance of the HAA-AT with ideal movement model is quite close to the global minimum of $\mathcal{P}3$, which should be between the lower bound and the performance of the HAA-AT. From the second experiment we can see that, the HAA-AT still can

(a) Random Mobility Model

(b) Clustered Mobility Model

Figure 6.7: Performance with respect to the variance of network latency

achieve good performance with more realistic mobility models.

Chapter 7

Case Study

A small scale multi-server Battle City Game was implemented for evaluating the updating and load balancing algorithms proposed in this thesis. Various experiments have been designed for evaluating the proposed algorithms on the following two aspects: the performance on reducing inconsistency and the performance on improving the playability of the game. The impact of upload bandwidth, network latency, the threshold to define the replica without QoS, et.al., on the performance of the proposed algorithms has been studied.

7.1 A Multi-server Battle City Game

Battle City, also known as Tank 1990, is a popular multi-directional shooter video game produced and published in 1985 by Namco. Many of us had played the game in our childhood, and some people even often play it on computer now.

The player, controlling a tank, must destroy enemy tanks in each level, which enter the playfield from the top of the screen. For each level, the playfield is represented by a map that consists of empty space, rivers, grass, steel walls and brick walls. The tank cannot move through rivers or walls, but it can destroy a brick wall by shooting at it. A brick wall will be turned into empty spaces after it is destroyed. However, if your shot hits a steel wall, there will be no damage to the wall. For the player controlled tank, the player can choose to move to a neighboring (4 directions, UP, DOWN, LEFT and RIGHT) empty space, or shoot in one of the four directions without a move. The shot will go ahead in that direction, until it goes out of the map or hits a wall. If the

shot hits a brick wall, the wall will disappear. The enemy tanks attempt to destroy the player's base (represented on the map as a bird, eagle or Phoenix), as well as the player controlled tank itself. A level is completed when the player destroys all 20 enemy Tanks, but the game ends if the player's base is destroyed or the player loses all available lives.

The traditional Battle City Game is stand-alone version and the game iteration is controlled by a single process. We have implemented a different edition of this game based on multi-server architecture. In the multi-server Battle City Game, a large size playfield is simulated, which is composed by 9 maps (arranged in 3x3). Three servers runs on three different machines for maintaining the maps, which are distributed and connected via network. Each map represents one zone and each server maintains a subset of zones. A separate administrative server provides the service for player's log in/log out and runs the scheduling and load balancing algorithms.

Figure 7.1: A screenshot of a map (zone)

Multiple players are allowed to participate in the game simultaneously. Each player

is connected to one server for sending commands and receiving updates of other tanks. Each tank can move around in the playfield and get across from one zone to another zone. The AOI of each tank is the zone that it is currently residing in. More details regarding the gameplay will be presented in the experimental design section. Figure 7.1 shows a screenshot of a map of the multi-server Battle City Game.

7.2 Experimental Design

The multi-server Battle City Game is a typical zone-based MSDVE system we have focused on in this thesis. The system model and update schema of the multi-server Battle City Game are the same as those that have been described in Chapter 3. The consistent state of virtual world is maintained by servers. Servers receive player's commands and disseminate new state updates to players after executing the commands. Each player maintains a copy of the map and the replicas of other tanks in the same zone. The player receives state updates from servers for updating the replicas and renders the new display of the game. The HAA-AT algorithm is used to determine which server a player is connected to.

In the experiments, we focus on the tank position updates and consider the inconsistency between replicas and the primary copy of tanks. In addition to the players, some enemy tanks are also simulated, which are controlled by emulated players. Each enemy tank keeps moving until it changes direction or when it touches the walls, rivers or the boundary of the virtual world. It changes direction every 250 frames. For each change, the new direction is selected randomly. The moving speed of each enemy tank is uniformly distributed in $[1,5]$ distance units per frame.

Since a tank can move freely in the virtual world, if a tank moves to another zone maintained by a different server during the gameplay, the tank will be migrated to the new server accordingly. Suppose the 9 zones are denoted as $z_0, \dots, z_8$ and 3 servers are denoted as s_0, s_1, s_2 respectively. Two methods are used to generate different workload distributions among servers. Firstly, in order to generate a uniformly distributed workload, each zone is assigned the same number of 10 enemy tanks and zones are

uniformly distributed among servers. A tank can move freely from one zone to another zone during the gameplay. In this manner, each server will maintain almost the same number of tanks and the workload, in terms of upload bandwidth, is uniformly distributed among servers. Then, in order to generate an imbalanced workload distribution, each zone is assigned an entering probability. During the gameplay, if a tank is attempting to cross a zone boundary, the tank is allowed to enter the new zone with the entering probability of the new zone. Otherwise, the tank will stay in the old zone and change its moving direction. The set of entering probabilities is as follows: {32%, 16%, 16%, 8%, 8%, 8%, 4%, 4%, 4%}, which is randomly assigned to in zones every 2 minutes to change the workload distribution. In the initialization, the number of tanks assigned to a zone is calculated by $100 \times ep$, where ep is the entering probability has been assigned to this zone.

In order to simulate the upload bandwidth limitations, only a given number of replicas is allowed to be updated in each frame for each server. The position updates are scheduled by the proposed scheduling algorithms. In order to make the game more smooth, the tank positions between two consecutive position updates are predicted by the first-order prediction model⁹ according to last two positions. In the experiments, the upload bandwidth in terms of number of position updates were varied widely for examining the impact of different upload bandwidth on the performance of the proposed algorithms.

In the previous sections, time-space inconsistency and the number of replicas without QoS were defined as two metrics for evaluating the proposed algorithms in the simulations. In this case study, we defined two game scenario settings. Under different game scenario settings, different performance metrics are used.

Scenario A In scenario A, we want to examine the efficiency of the proposed algorithms on reducing the inconsistency of the game. The experiment is designed as follows. We let a player control a tank to play against the enemy tanks. All the tanks just move around the virtual world and are not allowed to attack other tanks. In this

⁹The first-order prediction has been introduced in section 5.1.2 of Chapter 5.

manner, the number of tanks will keep unchanged in the examined period. The total number of replicas without QoS and the total time-space inconsistency of replicas are used as the metrics to evaluate the proposed algorithms.

Moreover, since first-order prediction is used for each replica to predict positions at client side, there will be errors if the position predicted is different from the true position. The prediction errors have great influence on the player's experience. In the presence of upload bandwidth limitations, the update interval of replicas will be increased. It means that more positions of replicas need to be predicted by clients and more prediction errors are likely to occur. Therefore, whether the upload bandwidth is allocated efficiently can be reflected by the number of prediction errors in the examined period. So, the number of prediction errors occurred at the client side of players is used as another metric for evaluating the proposed algorithms in scenario A.

Scenario B In scenario B, we want to examine how the proposed algorithms affect the gameplay. We let a player control a tank to play the game against enemy tanks controlled by emulated players. The player controlled tank can shoot and destroy the enemy tanks. The task of the player is to destroy all the enemy tanks in the virtual world as quickly as possible. The time taken to complete the game is highly dependent on the playability of the game, which is much related to the overall inconsistency [98]. If the playability is not good (e.g., the game screen is not smooth), the player will be interfered and the time taken to finish the game will be increased accordingly. Therefore, the time taken to finish the game is taken as the metric for evaluating the proposed algorithms in scenario B.

In the experiments, three servers were running on three different computers in PDCC, Nanyang Technological University. The player used another computer in PDCC to participate in the game. The network latency and the variance of latency between any two computers were controlled by the network simulator Shunra Cloud [1]. The network latencies from the emulated players to servers were modeled by the shifted exponential distribution with probability density function $f(x) =$

$\lambda e^{-\lambda(x-\tau)}(x \geq \tau)$ as introduced in Section 4.3.1. The parameters to control the latency and the variance of network latency were varied widely to test the impact on the performance of the proposed algorithms. The experiment setup is shown in Figure 7.2.

Figure 7.2: Experimental Setup

We examined the impact of the upload bandwidth, network latency on the performance of the proposed updating and load balancing algorithms. For each evaluation, both scenarios A and B are examined. For scenario A, the total number of replicas without QoS and the total number of prediction errors are collected (always shown in subfigure (a) and (b) in the experimental results). For scenario B, the time taken to destroy all the enemy tanks are collected (always shown in subfigure (c) in the experimental results). In order to make the evaluation more robust, we try to run the same experiment many times (usually more than 20 times) to make sure the results set is large enough. Then, we drop the "abnormal" results that have a big difference (more than 15%) from the average value. At last, we take the average value of the remaining results as the final result. In order to reduce the impact of the increasing playability of the player (that enhances its skills game after game), we let the player

spend some time getting familiar with the game before doing our experiments. The default values of other parameters used in the experiments are shown in Table 7.1.

Table 7.1: The default values of some experimental parameters

Parameters	Default Values
Speed of Enemy Tanks	[1,5] Distance Units/Frame
Average Network Latency (d)	0.1s
Variance of Network Latency (ν)	0.95
Frame Length	0.02s
Network Bandwidth for Each Server	10 Bandwidth Units/Frame
User Input Commands (u_i)	1 Bandwidth Unit/Frame

7.3 Experimental Results and Discussions

7.3.1 Evaluation of the LPUS and RPUS

In this subsection, the performance of the LPUS and RPUS with the game has been examined. The LPUS and the RPUS are proposed in Chapter 4 to reduce the number of replicas without QoS.

7.3.1.1 Performance with respect to Upload Bandwidth

Figure 7.3: Performance with respect to upload bandwidth with imbalanced workload distribution

Firstly, the impact of upload bandwidth was examined. The upload bandwidth was varied from 5 to 25 position updates per frame per server and other parameters

Figure 7.4: Performance with respect to upload bandwidth with uniform workload distribution

are set according to Table 7.1. The results with imbalanced workload distribution are shown in Figure 7.3.

As can be seen from Figure 7.3(a), the number of remaining replicas without QoS is decreased as the upload bandwidth capacity increases for both LPUS and RPUS. This is because more replicas without QoS can be updated with larger upload bandwidth capacity at each update frame. Similar to the simulation results in Chapter 4, the RPUS can update more replicas without QoS than the LPUS for the same upload bandwidth capacity.

As shown in Figure 7.3(b), the number of prediction errors is also decreased as the upload bandwidth capacity increases. This is because with more upload bandwidth, the replicas can be updated more frequently. It means that the prediction model is more accurate since the prediction model is calculated based on last two received position updates. Therefore, the probability of prediction error happening is lowered and the number of prediction errors is decreased. From the results, the RPUS achieves fewer prediction errors compared to the LPUS for the same upload bandwidth capacity. It implies that the RPUS allocates the upload bandwidth more efficiently than the LPUS.

The time taken to destroy all the enemy tanks also decreases as the upload bandwidth capacity increases, which can be seen in Figure 7.3(c). For the same player, the time taken is highly dependent on the playability of the game. If the upload

bandwidth is limited, the replicas cannot be updated frequently and more prediction errors will occur at client side. In this case, the player often sees wrong positions of enemy tanks. Therefore, the accuracy of the shot will be affected and the time to destroy all the tanks will be increased accordingly. From the results, we can see that the RPUS achieves shorter time to finish the game than the LPUS for the same upload bandwidth capacity. It implies that efficient update schedules such as the RPUS can really improve the playability of the game.

Figure 7.4 shows the performance of the proposed algorithms with uniform workload distribution. As can be seen, the number of remaining replicas without QoS, the prediction errors and the time taken to destroy all the enemy tanks are all greatly decreased compared to the results with imbalanced workload distribution (Figure 7.3). This is because with uniform workload distribution, the workload in terms of upload bandwidth is uniformly distributed among servers. All the replicas can be updated frequently and the overall inconsistency is greatly decreased. The playability of the game is improved compared to the case with imbalanced workload distribution. Therefore, the number of replicas without QoS, the prediction errors and the time taken to finish the task are all decreased.

From the results we can also see that the performance of different updating algorithms is quite close in uniform workload distribution. This implies that the effect of the updating algorithms for the uniform workload distribution is insignificant.

7.3.1.2 Performance with respect to Network Latency

The impact of network latency was examined. The network latency between two nodes was varied from 0.1s to 0.5s and other parameters are set according to Table 7.1. Firstly, the variance of network latency (ν) was set at 0.95 to examine the situation with a small variance. The results with imbalanced workload distribution are shown in Figure 7.5.

As can be seen in Figure 7.5(a), the number of remaining replicas without QoS increases as the network latency increases. This is because larger network delay results in larger time-space inconsistency according to Equation 3.6. Since the threshold to

Figure 7.5: Performance with respect to network latency with imbalanced workload distribution ($\nu=0.95$)

Figure 7.6: Performance with respect to network latency with uniform workload distribution ($\nu=0.95$)

define the replica without QoS is not changed, the total number of replicas without QoS will be increased. The RPUS always achieves better performance than the LPUS for the same network latency.

The number of prediction errors also increases as the network latency increases. This is because with larger network delay, the transmission time of the position update from target server to the replica get larger. The prediction model at client side cannot be updated until the position update is received. Without timely updating, the accuracy of the prediction model will be decreased and more prediction errors will occur. From the results in Figure 7.5(b), the RPUS achieves fewer prediction error than the LPUS for the same network latency.

As the prediction errors increase, the playability of the game become worse. Therefore, the time taken to destroy the enemy tanks should be increased, which can be seen in Figure 7.5(c). The time taken when RPUS is used is shorter than that the LPUS is used for the same network latency. It implies that the RPUS gains more benefits than the LPUS for improving the playability of the game in this experiment.

Figure 7.6 shows the results with uniform workload distribution. As can be seen in Figure 7.6(a), Figure 7.6(b) and Figure 7.6(c), the total number of replicas without QoS, the prediction errors and the time taken to finish the game are all greatly decreased compared to the case with imbalanced workload. As has been explained, this is because the replicas can be updated frequently and the overall inconsistency is small with uniform workload distribution. Similar to the case with imbalanced workload distribution, the number of replicas without QoS and the prediction errors also increase with the network latency increases. Moreover, as the increase of the network latency, the player feels larger lag in the gameplay. Therefore, the playability is affected and the time taken to destroy the enemy tanks is increased. However, the performance of the RPUS and the LPUS is very close in uniform workload distribution, which implies that the benefit of the updating algorithms is insignificant.

Figure 7.7: Performance with respect to network latency with imbalanced workload distribution ($\nu=0.5$)

Then, we set $\nu = 0.5$ to simulate a large variance of network latency. The results with imbalanced workload distribution are shown in Figure 7.7. Compared to the

case with small variance of network latency (i.e., the results shown in Figure 7.5), the replicas without QoS, the prediction errors and the time taken to finish the game are all increased for the same network latency. This is because larger variance of network latency may incur some very large network delays. This results in large time-space inconsistency. As has been discussed, all the metrics will be increased as the increase of the time-space inconsistency.

7.3.1.3 Performance with respect to Threshold Value

Figure 7.8: Performance with respect to threshold value with imbalanced workload distribution

In this experiment, the performance of the LPUS and RPUS with respect to the time-space inconsistency threshold value which is used to define the replicas without QoS was examined. For the imbalanced workload distribution, the threshold value was varied from 10 to 50 and other parameters are set according to Table 7.1. The results are shown in Figure 7.8. Intuitively, the number of replicas without QoS is determined by the threshold value to define the replica without QoS. When the other experimental parameters keep unchanged, the total number of replicas without QoS should be decreased as the increase of the threshold value, which can be seen from the results in Figure 7.8(a).

We also notice that the total number of prediction errors and the time taken to finish the game both achieve the minimum value when the threshold value is set at 30. The threshold to define replica without QoS represents the time-space inconsistency

bound that the player can endure in the gameplay. If the threshold value is not set appropriately, how the player reacts to the time-space inconsistency cannot be precisely reflected. When the threshold is too big, there is less number of replicas without QoS. It results in less frequent updates and more prediction errors occur. So, the playability will be affected. From the results in Figure 7.8(b) and Figure 7.8(c), the most appropriate value of the threshold for our experiment is 30. Therefore, the threshold value is set at 30 in the rest of the experiments. Moreover, the RPUS always performs better than the LPUS for the same experimental settings. It implies that the RPUS is more efficient for allocating limited resources for improving the playability of the game with imbalanced workload distribution among servers.

In summary, from the experimental results, the updating algorithms for reducing replicas without QoS can really improve the playability of the game with upload bandwidth limitations. The RPUS has been shown to perform better than the LPUS for the same experimental settings.

7.3.2 Evaluation of the LMH

In this subsection, the performance of the LMH has been examined. The LMH is proposed in Chapter 5 for reducing the total time-space inconsistency. The other scheduling algorithms i.e., the TIME, SPACE and TotalTS were also compared in the experiment. The frequency of regulating optimal μ was set at once per 100 frames. This value is obtained by the similar experiments in Section 5.2.3.1. For each experiment, both scenario A and scenario B were examined with variant parameter settings. The total time-space inconsistency, the number of prediction errors and the time taken to destroy the enemies were collected as the metrics.

7.3.2.1 Performance with respect to Upload Bandwidth

Firstly, the impact of upload bandwidth was examined. For each experiment, the upload bandwidth of each server was set in a range from 5 to 25 position updates per-frame and other parameters are set as shown in Table 7.1.

Figure 7.9: Performance with respect to upload bandwidth with imbalanced workload distribution

Figure 7.10: Performance with respect to upload bandwidth with uniform workload distribution

Figure 7.9 shows the results with imbalanced workload distribution. Similar to the simulation results in Chapter 5, lower upload bandwidth capacity makes larger time-space inconsistency as can be seen in Figure 7.9(a). The LMH can greatly reduce time-space inconsistency compared to other update algorithms for the same upload bandwidth capacity.

The number of prediction errors also decreases as the upload bandwidth increases, which can be seen in Figure 7.9(b). This is because with large upload bandwidth capacity, the prediction model can be updated more frequently thus the probability of prediction error happening is decreased. With less prediction errors, the playability of the game is improved. Therefore, as can be seen in Figure 7.9(c), the time taken to destroy the enemy tanks is decreased as the increase of the upload capacity. The

LMH always achieves less prediction errors and less time taken than other updating algorithms. It implies that the update schedules for reducing time-space inconsistency can really improve the playability of the game. Moreover, the LMH is more efficient in reducing the inconsistency and improving the playability than other algorithms for the same upload bandwidth capacity.

The performance of all the algorithms with uniform workload distribution was also examined. As can be seen in Figure 7.10, compared to the situation of imbalanced workload distribution, the total time-space inconsistency, the number of prediction errors and the time taken to finish the game are all greatly decreased. As has been explained, this is because the replicas are updated frequently with more upload bandwidth capacity. The time-space inconsistency of replicas is decreased and the playability of the game is improved. We can also see that the performance of all the proposed algorithms is close to each other. It implies that the benefit of the update schedules for reducing time-space inconsistency and improving playability for uniform workload distribution is insignificant.

7.3.2.2 Performance with respect to Network Latency

Figure 7.11: Performance with respect to network latency with imbalanced workload distribution ($\nu=0.95$)

Next, the impact of network latency on the performance of the proposed algorithms was examined. For each experiment, the average network latency between two nodes was ranged from 0.1s to 0.5s. We firstly set $\nu = 0.95$ to simulate a small variance of

Figure 7.12: Performance with respect to network latency with uniform workload distribution ($\nu=0.95$)

network latency. The performance results for the imbalanced workload distribution are shown in Figure 7.11. The total time-space inconsistency, the prediction errors and the time taken to finish the game all increase as the network latency increases. The reason why the prediction errors and the time taken are increased as the increase of the network latency has been explained in subsection 7.3.1. As the increase of the network latency, the time-space inconsistency of replicas will be increased. Therefore, the total time-space inconsistency is increased accordingly. From the results we can see that the LMH always achieves a better performance than other updating algorithms for the same network latency.

The results with uniform workload distribution are shown in Figure 7.12. Similar to the results in subsection 7.3.1, the total time-space inconsistency, the prediction errors and the time taken are all greatly increased compared to the case with imbalanced workload distribution. All the evaluation metrics are increasing with the increase of network latency. The performance of the proposed algorithms are close which implies that the benefit of the update schedules is insignificant for uniform workload distribution.

Then, we set $\nu = 0.5$ to generate a large variance of the network latency. The results with imbalanced workload distribution are shown in Figure 7.13. As has been discussed in subsection 7.3.1, the time-space inconsistency of replicas increases with large network latency variance. As a result, the prediction errors and the time taken

Figure 7.13: Performance with respect to network latency with imbalanced workload distribution ($\nu=0.5$)

are both increased as the increase of the network latency variance. With large variance of network latency, the LMH can still outperform other updating algorithms for the experimental settings. It implies that the LMH is still efficient with variant network conditions.

In summary, the results imply that the update schedules for reducing time-space inconsistency can really improve the playability of the game. Moreover, the LMH has been shown high efficiency in reducing time-space inconsistency and improving the playability in the game compared to other updating algorithms.

7.3.3 Evaluation of the HAA-AT

In this subsection, the performance of the HAA-AT in the game was examined. The HAA-AT is proposed in Chapter 6, which is used to determine the zone mapping and client assignment for minimizing time-space inconsistency. The RANDOM algorithm which is proposed in Chapter 6 was also implemented in the game for comparison. For each experiment, both scenario A and scenario B were executed with variant parameter settings. The total time-space inconsistency, the prediction errors and the time taken to destroy all the enemy tanks were collected as the metrics.

7.3.3.1 Performance with respect to upload bandwidth

Firstly, the performance of the HAA-AT with respect to upload bandwidth capacity was examined. The upload bandwidth capacity was ranged from 5 to 25 updates

Figure 7.14: Performance with respect to upload bandwidth with imbalanced workload distribution

Figure 7.15: Performance with respect to upload bandwidth with uniform workload distribution

per frame per server and other parameters were set according to Table 7.1. Figure 7.14 shows the results with imbalanced workload distribution. Similar to the previous results, the time-space inconsistency, the prediction errors and the time taken all decrease as the increase of the upload bandwidth capacity. The HAA-AT algorithm outperforms the RANDOM on all metrics for the same upload bandwidth capacity. This implies that the load balancing mechanisms (zone mapping and client assignment) can really reduce the overall inconsistency and improve the playability of the game.

The results with uniform workload distribution are shown in Figure 7.15. Since the workload is uniformly distributed among servers, the saturations of upload bandwidth seldom happens. So, the time-space inconsistency, the prediction errors and the

time taken are all decreased greatly compared to the case with imbalanced workload distribution. Since the workload is already uniformly distributed, the benefit of the load balancing algorithms is insignificant. So, the performance of the HAA-AT and the RANDOM are very close for the same upload bandwidth capacity, which can be seen in Figure 7.15.

7.3.3.2 Performance with respect to network latency

Figure 7.16: Performance with respect to network latency with imbalanced workload distribution ($\nu=0.95$)

Figure 7.17: Performance with respect to network latency with uniform workload distribution ($\nu=0.95$)

Next, the impact of network latency on the performance of the proposed algorithms was examined. For each experiment, the average network latency between two nodes was ranged from 0.1s to 0.5s and other parameters were set according to Table 7.1. We firstly set $\nu = 0.95$ to simulate a small variance of network latency. The results with

imbalanced workload distribution are shown in Figure 7.16. As has been explained, the total time-space inconsistency, the prediction errors and the time taken to destroy all enemy tanks all increase with the network latency increases. For the same network latency setting, the HAA always achieves better performance than the RANDOM. It implies that even with large network latency, the HAA is still efficient in reducing time-space inconsistency and improving the playability of the game.

The results with uniform workload distribution are shown in Figure 7.17. Similarly, all the evaluation metrics all increase as the increase of the network latency. However, the performance of the HAA and the RANDOM is similar which implies that the load balancing algorithms have little effect on reducing the inconsistency and improving the playability of the game for uniform workload distribution.

Figure 7.18: Performance with respect to network latency with imbalanced workload distribution ($\nu=0.5$)

Then, we set $\nu = 0.5$ to simulate a large variance of network latency. The results with imbalanced workload distribution are shown in Figure 7.18. Compared to the results with small variance of network latency (Figure 7.16), larger variance results in larger time-space inconsistency, more prediction errors and longer time taken for the same network latency setting. The HAA can still outperform the RANDOM for large variance of network latency.

In summary, the results in this experiment show the efficiency of the HAA in reducing the time-space inconsistency and improving the playability of the game.

7.4 Summary

In this Chapter, we implemented a multi-server Tank Combat Game to evaluate that how the proposed updating and load balancing algorithms perform in a real MSDVE system. The results show that updating and load balancing algorithms can really reduce the inconsistency of the game and improve the playability of the game when the upload bandwidth is limited at servers. Our proposed algorithms have been shown high efficiency in reducing inconsistency and improving the playability of the game compared to other algorithms.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

At present, a large scale MSDVE like MMOGs can include millions of concurrent users spread across the world, which demands a huge amount of computing and network bandwidth resources on various servers. Facing huge resource demand, servers could get saturated with resources. Moreover, a DVE system is changing over time which results in dynamic workload requirement at servers. If the workload is not properly distributed, the resource saturations at servers will be worsened. In the presence of resource limitations, some state updates may not be disseminated timely and time-space inconsistency is likely to be increased.

In this thesis, we study the update scheduling and load balancing issues for improving consistency in MSDVEs with a set of servers constrained by upload bandwidth. The update scheduling aims to schedule state updates according to their potential impacts on consistency. The existing work on investigating update schedules in DVEs all concentrates on the DVEs with single server. In MSDVEs, the problem gets much more complicated due to the inter-server communications.

The load balancing in MSDVEs generally refers to zone mapping and client assignment issues. The existing work on load balancing in MSDVEs mainly aims to balance workload among servers, reduce inter-server communication and improve the interactivity of the DVE. In this thesis, we consider the load balancing issues in MSDVEs from a new perspective with the purpose of reducing inconsistency.

A large scale MSDVE system often requires a considerable upload bandwidth which would take a large part of budget. Therefore, this thesis is primarily concerned with the upload bandwidth resource. However, the proposed techniques can be applied to manage other type of resources as well.

For the update scheduling issue, we mainly focus on scheduling position update of avatars for minimizing the overall inconsistency. The overall inconsistency of a DVE is defined in two different ways. Two problems are formulated respectively. In the first problem, the overall inconsistency is measured by the total number of replicas without QoS. The updating problem of minimizing the total number of replicas without QoS was formulated as an integer programming problem. The solution of the first problem was studied in Chapter 4 and two distributed algorithms (i.e., LPUS and RPUS) were proposed and evaluated. The simulation results show that the RPUS generally achieves better performance than the LPUS for the same experimental settings.

In the second problem, the overall inconsistency is measured by the total time-space inconsistency of all replicas of avatars in a DVE over a given period. The problem of finding optimal update schedule to minimize the overall inconsistency was formulated as an inequality constrained problem for an ideal situation. Then, in Chapter 5, Lagrange Multipliers were used to derive a heuristic position update algorithm (i.e., LMH) for the practical systems. The simulation results show that the LMH always outperforms other updating algorithms for the same experimental settings.

For the load balancing issue, we consider the zone mapping and client assignment issues in MSDVEs from a different point of view of the existing work. The objective is to minimize the overall inconsistency and the problem of minimizing the overall inconsistency is formulated as a mix-integer programming problem. The solution of the problem was studied in Chapter 6. A centralized algorithm based on the Alternating Optimization (AO) method was proposed to obtain an approximate solution. Moreover, a distributed adaptive tuning algorithm was used to adapt dynamic changes in the practical systems. The simulation results show that the HAA-AT achieves better performance than the RANDOM for the same experimental settings.

In Chapter 7, a multi-server Battle City Game was designed for a case study. Various experiments have been designed to evaluate the performance of the proposed algorithms in real applications. The experimental results show that the proposed updating and load balancing algorithms can indeed reduce the overall inconsistency of the game. Moreover, the proposed updating and load balancing algorithms can also improve the playability of the game. The results show the importance and efficiency of the proposed algorithms, which confirms the motivation and the contributions of this thesis.

8.2 Future Work

In this section, we will discuss how the work reported in this thesis can be future extended and related research issues.

8.2.1 Fairness-aware Consistency Maintenance Issues

In this thesis, we only consider minimizing the inconsistency between the replicas maintained by clients and the primary copy of avatars maintained by servers. This kind of inconsistency reflects the real-time property of a DVE. However, due to variant network delays, the replicas of the same avatar maintained by different clients may receive the same state update at different times. Therefore, different clients may see different views of the virtual world. This kind of inconsistency between clients reflects the fairness among players in the DVE. The unfairness between clients also has great influence on the experiences. For example, the player who sees an event happening earlier will have more advantages over other players. Therefore, as a part of the future work, the fairness among clients can be considered in the design of update scheduling and load balancing algorithms in MSDVEs.

8.2.2 Consistency Maintenance in Other DVE Architectures

In this thesis, we mainly focus on the state updating and load balancing issues in zone-based MSDVEs. As has been introduced in the related work, except for the

zone-based architecture, the mirrored server architecture and the peer-to-peer architecture are also used in the practical systems. In such architectures, the system model and update schema are very different from zone-based architecture. Therefore, the consistency maintenance issues such as load balancing and state update scheduling in such architectures will be different. In the future work, we plan to extend the system model to such systems and study the consistency maintenance issues in such DVE architectures.

8.2.3 Exploring New DVE Infrastructures

As the fast development of the computer science technology, more and more new techniques can be used in the development of more scalable DVE infrastructures. Cloud computing is one of the promising techniques. By hosting the DVE in the Cloud, the burden of support and maintaining a huge number of servers and data centers will be lifted from the shoulder of DVE service providers. On-demand scalability and dynamic resource allocation can also be easily supported. Rather than keeping huge number of servers based on the peak demand, new server instance can be acquired from the Cloud on the actual demand. The above benefit of hosting DVEs in the Cloud will translate into substantial savings for the DVE service providers. In the future work, we plan to consider the deployment of the DVEs on the Cloud. The corresponding resource allocation and load balancing issues will be also investigated.

Author's Publications

Papers Published

- (i) Yusen Li, Wentong Cai, “Consistency-aware Partitioning Algorithm in Multi-server Distributed Virtual Environments”. *27th IEEE International Parallel and Distributed Processing Symposium*, 2012.
- (ii) Yusen Li, Wentong Cai, “Determining Optimal Update Period for Minimizing Inconsistency in Multi-server Distributed Virtual Environments”, *15th International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'11)*, 2011.
- (iii) Yusen Li, Wentong Cai, “Consistency Aware Dead Reckoning Threshold Tuning with Server Assistance in Client-Server-Based DVEs”. *10th IEEE International Conference on Computer and Information Technology*, 2010.
- (iv) Yusen Li, Wentong Cai, “Consistency Aware Update Schedule in Multi-Server Distributed Virtual Environments”. *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, 2010.

Papers under Preparation

- (i) Yusen Li, Wentong Cai, “Update Schedules for Improving Consistency in Multi-server Distributed Virtual Environments”. *Journal of Network and Computer Applications*, (Under Review).
- (ii) Yusen Li, Wentong Cai, “Consistency-aware Zone Mapping and Client Assignment in Multi-server Distributed Virtual Environments”. *IEEE Transactions on Parallel and Distributed Systems*, (Under Review)

References

- [1] <http://www.shunra.com/>.
- [2] H. Abrams. Three-tiered interest management for large-scale virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 125–129, 1998.
- [3] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan. Accuracy in dead-reckoning based distributed multi-player games. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, pages 161–165. ACM, 2004.
- [4] D. Ahmed and S. Shirmohammadi. A microcell oriented load balancing model for collaborative virtual environments. In *IEEE Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems, 2008.*, pages 86–91. IEEE, 2008.
- [5] K. Aida and H. Casanova. Scheduling mixed-parallel applications with advance reservations. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC)*, pages 65–74, 2008.
- [6] J. Barrus, R. Waters, and D. Anderson. Locale and beacons: Efficient and precise support for large multi-user virtual environments. *MERL TR95-16a (August, 1996)*, 1996.
- [7] M. A. Bassiouni, M.-H. Chiu, M. Loper, M. Garnsey, and J. Williams. Performance and reliability analysis of relevance filtering for scalable distributed interactive simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 7(3):293–331, 1997.
- [8] D. Bertsekas. Constrained optimization and Lagrange multiplier methods. 1982.

- [9] J. Bezdek and R. Hathaway. Some notes on alternating optimization. *Advances in Soft Computing AFSS 2002*, pages 187–195, 2002.
- [10] C. Bezerra and C. Geyer. A load balancing scheme for massively multiplayer online games. *Multimedia Tools and Applications*, 45(1):263–289, 2009.
- [11] F. Bonomi and A. Kumar. Optimality of weighted least squares load balancing. In *Proceedings of the 27th IEEE Conference on Decision and Control*, pages 1480–1485. IEEE, 1988.
- [12] 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.
- [13] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge Univ Pr, 2004.
- [14] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on mobile computing and networking*, pages 85–97. ACM, 1998.
- [15] W. Cai, B. Lee, and L. Chen. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation*, pages 82–89. IEEE Computer Society, 1999.
- [16] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen. The SIMNET virtual world architecture. In *Virtual Reality Annual International Symposium*, pages 450–455, 1993.
- [17] C. Carlsson and O. Hagsand. Dive a multi-user virtual reality system. In *IEEE Virtual Reality Annual International Symposium*, pages 394–400. IEEE, 1993.
- [18] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, 1988.
- [19] J. Chen, B. Wu, M. Delap, B. Knutsson, H. Lu, and C. Amza. Locality aware dynamic load management for massively multiplayer games. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 289–300. ACM, 2005.

- [20] K.-T. Chen, P. Huang, and C.-L. Lei. How sensitive are online gamers to network quality? *Communications of the ACM*, 49(11):34–38, 2006.
- [21] K.-T. Chen, P. Huang, and C.-L. Lei. Effect of network quality on player departure behavior in online games. *IEEE Transactions on Parallel and Distributed Systems*, 20(5):593–606, 2009.
- [22] R. Chertov and S. Fahmy. Optimistic load balancing in a distributed virtual environment. In *Proceedings of the 2006 international workshop on network and operating systems support for digital audio and video*, page 13. ACM, 2006.
- [23] S. Cheshire. Latency and the quest for interactivity. In *White paper commissioned by Volpe Welty Asset Management*, 1996.
- [24] J. Chim, R. W. H. Lau, H. V. Leong, and A. Si. Cyberwalk: A web-based distributed virtual walkthrough environment. *IEEE Transactions on Multimedia (TMM)*, 5:503–515, 2003.
- [25] A. Corlett, D. Pullin, and S. Sargood. Statistics of one-way Internet packet delays. *53 rd IETF*.
- [26] A. Cox, E. Luiijf, R. Van Kampen, and R. Ripley. Time synchronization experiments. In *Proceedings of the 14th DIS workshop*, 1996.
- [27] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. *Information and Computation*, 118(1):158, 1995.
- [28] E. Cronin, B. Filstrup, A. Kurc, et al. A distributed multiplayer game server system. 2001.
- [29] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. *Multimedia Tools and Applications*, 23(1):7–30, 2004.
- [30] T. Das, G. Singh, A. Mitchell, P. Kumar, and K. McGee. Neteffect: a network architecture for large-scale multi-user virtual worlds. In *Proceedings of the ACM symposium on Virtual reality software and technology*. ACM, 1997.
- [31] T. De Senna Carneiro and J. Cotrim Arabe. Load balancing for distributed virtual reality systems. In *International Symposium on Computer Graphics, Image Processing, and Vision (SIBGRAPI), 1998.*, pages 158–165. IEEE, 1998.

- [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 *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–7. ACM, 2005.
- [33] E. Deelman and B. Szymanski. Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. In *20th Workshop on Parallel and Distributed Simulation*, pages 46–53. IEEE, 1998.
- [34] D. Delaney, T. Ward, and S. McLoone. On consistency and network latency in distributed interactive applications: A survey-part i. *Presence: Teleoperators & Virtual Environments*, 15(2):218–234, 2006.
- [35] D. Delaney, T. Ward, and S. McLoone. On consistency and network latency in distributed interactive applications: A survey-part ii. *Presence: Teleoperators & Virtual Environments*, 15(4):465–482, 2006.
- [36] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet. *IEEE Network*, 13(4):6–15, 1999.
- [37] T. Duncan and D. Gracanin. Pre-reckoning algorithm for distributed virtual environments. In *Proceedings of the 2003 Winter Simulation Conference, 2003.*, volume 2, 2003.
- [38] T. Duong and S. Zhou. A dynamic load sharing algorithm for massively multiplayer online games. In *The 11th IEEE International Conference on Networks (ICON), 2003.*, pages 131–136. IEEE, 2003.
- [39] Everquest. <http://everquest.station.sony.com>.
- [40] C. Faisstnauer, D. Schmalstieg, and W. Purgathofer. Priority scheduling for networked virtual environments. *IEEE Computer Graphics and Applications*, 20(6):66–75, 2002.
- [41] C. Floudas, A. Aggarwal, and A. Ciric. Global optimum search for nonconvex nlp and minlp problems. *Computers and chemical engineering*, 13(10):1117–1132, 1989.
- [42] D. Frey, J. Royan, R. Piegay, A.-M. Kermarrec, E. Anceaume, F. Le Fessant, et al. Solipsis: A decentralized architecture for virtual environments. In *1st International Workshop on Massively Multiuser Virtual Environments*, 2008.

- [43] R. Fujimoto. Parallel and distributed simulation. In *1999 Winter Simulation Conference Proceedings*. IEEE, 1999.
- [44] T. Funkhouser. Ring: a client-server system for multi-user virtual environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*. ACM, 1995.
- [45] L. Gautier, C. Diot, et al. Mimaze, a multiuser game on the internet. 1997.
- [46] L. Gautier, C. Diot, and J. Kurose. End-to-end transmission control mechanisms for multiparty interactive applications on the internet. In *IEEE International Conference on Computer Communications*. IEEE, 1999.
- [47] C. Greenhalgh, J. Purbrick, and D. Snowdon. Inside massive-3: flexible support for data consistency and world structuring. In *Proceedings of the third international conference on Collaborative virtual environments*, pages 119–127. ACM, 2000.
- [48] A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs. Scheduling jobs with varying parallelizability to reduce variance. In *Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 11–20, 2010.
- [49] 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.
- [50] S. Hu, J. Chen, and T. Chen. VON: a scalable peer-to-peer network for virtual environments. *IEEE Network*, 20(4):22–31, 2006.
- [51] S.-Y. Hu and G.-M. Liao. Scalable peer-to-peer networked virtual environment. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 129–133. ACM, 2004.
- [52] D. R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(3):404–425, 1985.
- [53] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.
- [54] S. Khuri and T. Chiu. Heuristic algorithms for the terminal assignment problem. In *Proceedings of the 1997 ACM symposium on Applied computing*, pages 247–251. ACM, 1997.

- [55] R. Lea, Y. Honda, K. Matsuda, and S. Matsuda. Community place: architecture and performance. In *Proceedings of the second symposium on Virtual reality modeling language*, pages 41–50. ACM, 1997.
- [56] H. Lee. Optimal static distribution of prioritized customers to heterogeneous parallel servers. *Computers & operations research*, 22(10):995–1003, 1995.
- [57] K. Lee and D. Lee. A scalable dynamic load distribution scheme for multi-server distributed virtual environment systems with highly-skewed user distribution. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 160–168. ACM, 2003.
- [58] F. Li, R. Lau, and F. Ng. Vsculpt: a distributed virtual sculpting environment for collaborative design. *IEEE Transactions on Multimedia (TMM)*, 5(4):570–580, 2003.
- [59] Y. Li and W. Cai. Consistency aware update schedule in multi-server distributed virtual environments. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, page 3. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.
- [60] F. Lu, S. Parkin, and G. Morgan. Load balancing for massively multiplayer online games. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 1. ACM, 2006.
- [61] J. Lui. Constructing communication subgraphs and deriving an optimal synchronization interval for distributed virtual environment systems. *IEEE Transactions on Knowledge and Data Engineering*, 13(5):778–792, 2001.
- [62] J. Lui and M. Chan. An efficient partitioning algorithm for distributed virtual environment systems. *IEEE Transactions on Parallel and Distributed Systems*, pages 193–211, 2002.
- [63] M. R. Macedonia, D. P. Brutzman, M. J. Zyda, D. R. Pratt, P. T. Barham, J. Falby, and J. Locke. Npsnet: a multi-player 3d virtual environment over the internet. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 93–ff. ACM, 1995.
- [64] D. L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991.

- [65] D. Min, D. Lee, B. Park, and E. Choi. A load balancing algorithm for a distributed multimedia game server architecture. In *IEEE International Conference on Multimedia Computing and Systems, 1999.*, page 882. Published by the IEEE Computer Society, 1999.
- [66] P. Morillo et al. Improving the performance of distributed virtual environment systems. *IEEE Transactions on Parallel and Distributed Systems*, pages 637–649, 2005.
- [67] P. Morillo, J. Orduna, and M. Duato. A fine-grain method for solving the partitioning problem in distributed virtual environment systems. In *Parallel and Distributed Computing and Systems*. ACTA Press.
- [68] P. Morillo, J. Orduña, M. Fernández, and J. Duato. On the characterization of distributed virtual environment systems. *Euro-Par 2003 Parallel Processing*, pages 1190–1198, 2003.
- [69] P. Morillo, S. Rueda, J. Orduna, and J. Duato. A latency-aware partitioning method for distributed virtual environment systems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, pages 1215–1226, 2007.
- [70] 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.
- [71] D. Nguyen Binh Ta, S. Zhou, W. Cai, X. Tang, and R. Ayani. Multi-objective zone mapping in large-scale distributed virtual environments. *Journal of Network and Computer Applications*, 34(2):551–561, 2011.
- [72] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proceedings of the Twenty-Sixth International Conference on Very Large Data Bases (VLDB)*, 2000.
- [73] X. Qin. Delayed consistency model for distributed interactive systems with real-time continuous media. *Journal of Software*, 13(6):1029–1039, 2002.
- [74] K. Ramamritham and J. Stankovic. Dynamic task scheduling in hard real-time distributed systems. *IEEE Software*, 1(3):65–75, 2006.
- [75] W. Romberg. Vereinfachte numerische integration. *Det Kongelige Norske Videnskabers Selskab Forhandlinger*, 28(7):30–36, 1955.

- [76] P. Rosedale and C. Ondrejka. Enabling player-created online worlds with grid computing and streaming. *Gamasutra Resource Guide*, 2003.
- [77] K. W. Ross and D. D. Yao. Optimal load balancing and scheduling in a distributed computer system. *Journal of the ACM (JACM)*, 38(3):676–689, 1991.
- [78] H. G. Rotithor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. In *IEE Proceedings Computers and Digital Techniques*, volume 141, pages 1–10. IET, 1994.
- [79] SecondLife. <http://secondlife.com>.
- [80] M. Sharaf, J. Beaver, A. Labrinidis, and P. Chrysanthis. Balancing energy efficiency and quality of aggregate data in sensor networks. *The VLDB journal*, 13(4):384–403, 2004.
- [81] P. M. Sharkey, M. D. Ryan, and D. J. Roberts. A local perception filter for distributed virtual environments. In *Virtual Reality Annual International Symposium*, pages 242–249, 1998.
- [82] A. Steed and R. Abou-Haidar. Partitioning crowded virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 7–14. ACM, 2003.
- [83] 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.
- [84] D. Ta et al. Greedy algorithms for client assignment in large-scale distributed virtual environments. *Simulation*, 84(10-11):521, 2008.
- [85] D. Ta and S. Zhou. Efficient client-to-server assignments for distributed virtual environments. In *20th International Parallel and Distributed Processing Symposium, 2006.*, pages 10–pp. IEEE, 2006.
- [86] X. Tang and W. Cai. Communication-efficient support for spatialfiltering of state updates in distributed virtual environments. In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pages 129–136, 2009.

- [87] 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.
- [88] M. Tsai. The study of dynamic management for massive-players virtual environment. 2005.
- [89] G. Wang, B. Zhang, and T. Ng. Towards network triangle inequality violation aware distributed systems. In *Proceedings of the 7th ACM SIGCOMM conference on Internet Measurement*, pages 175–188. ACM, 2007.
- [90] WoW. <http://www.worldofwarcraft.com/index.xml>.
- [91] WoWCensus. <http://www.warcraftrealms.com/censusplus.php>.
- [92] Y. Yu, Z. Li, L. Shi, Y. Chen, and H. Xu. Network-aware state update for large scale mobile games. In *Proceedings of 16th International Conference on Computer Communications and Networks (ICCCN)*, pages 563–568, 2007.
- [93] L. Zhang and X. Tang. Client assignment for improving interactivity in distributed interactive applications. In *2011 Proceedings IEEE INFOCOM*, pages 3227–3235.
- [94] L. Zhang and X. Tang. The client assignment problem for continuous distributed interactive applications. In *31st International Conference on Distributed Computing Systems (ICDCS), 2011*, pages 203–214.
- [95] 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*, page 7. ACM, 2006.
- [96] W. Zhao, K. Ramamritham, and J. Stankovic. Scheduling tasks with resource requirements in hard real-time systems. *IEEE Transactions on Software Engineering*, (5):564–577, 2006.
- [97] S. Zhou, W. Cai, B. Lee, and S. Turner. Consistency in distributed interactive applications. In *European Simulation Interoperability Workshop 2001*. IEEE Computer Society, 2001.

- [98] S. Zhou, W. Cai, B. Lee, and S. Turner. Time-space consistency in large-scale distributed virtual environments. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 14(1):31–47, 2004.
- [99] S. Zhou, S. J. Turner, W. Cai, H. Zhao, and X. Pang. A utility model for timely state update in distributed wargame simulations. In *Proceedings of the Eighteenth Workshop on Parallel and Distributed Simulation (PADS)*, PADS '04, 2004.