NANYANG TECHNOLOGICAL UNIVERSITY

A SERVICE ORIENTED HLA RTI ON THE GRID

PAN KE

SCHOOL OF COMPUTER ENGINEERING

2010

A SERVICE ORIENTED HLA RTI ON THE GRID

PAN KE

School of Computer Engineering

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

2010

Acknowledgements

Firstly, I would like to express my heartfelt thanks to my supervisor, Prof. Stephen John Turner, and co-supervisor Prof. Cai Wentong for their continuous remarkable and patient guidance throughout my four years' study. Their expertise and invaluable advice has left a profound impact on my work, especially the way I think of and solve new problems.

I would like to thank my project partner Mr. Li Zengxiang for the precious discussions we have. I am greatly indebted to Mrs. Ng-Goh Siew Lai, Irene and Ms. Tay Siew Eng, the technicians of Parallel and Distributed Computing Center (PDCC) for their patient help along the way.

Special thanks should be given to Dr. Chen Dan, Dr. Georgios K. Theodoropoulos from School of Computer Science in University of Birmingham for my WAN experimental configurations between Singapore and UK. I really appreciate their help as most of the WAN data I have in this thesis are between Singapore and UK.

Special thanks should also be given to Prof. Richard M. Fujimoto, Dr. Gu Yan and Chad Huneycutt from College of Computing in Georgia Institute of Technology, USA, for my WAN experimental configurations between Singapore and USA.

I would like to thank all of my family: my grandma, my parents and my sister for their strong support. Last, but not least, I thank my girl friend Guo Xi, who lights my life and always drives me forward.

Table of Contents

Summary

Simulation is a low cost and safe alternative to solve complex problems in many areas of business, science and engineering. To promote the interoperability and reusability of simulation applications and link geographically dispersed simulation components, distributed simulation was introduced. While the High Level Architecture (HLA) is the IEEE standard for distributed simulation, a Run Time Infrastructure (RTI) provides the actual implementation of the HLA specification. With increased size and complexity of simulation applications, large amounts of distributed computational and data resources are required. The Grid provides a flexible, secure and coordinated resource sharing environment which can facilitate distributed simulation execution.

This thesis describes an open-source Service Oriented HLA RTI (SOHR) framework. SOHR provides the functionalities of an RTI as Grid services and enables distributed simulations to be conducted in a heterogeneous Grid Environment. The various services in SOHR can be dynamically deployed, discovered and undeployed, leading to a scalable distributed simulation environment. While the communications between simulation components are through Grid service invocations, the standard HLA interface is provided as a library to increase simulator reusability and interoperability. A subset of the HLA specification is implemented in a SOHR prototype based on Globus Toolkit GT4 and the experimental results show the feasibility of SOHR.

Since SOHR creates a plug-and-play paradigm for an RTI implementation, this project has developed new algorithms for the HLA TM (Time Management) and DDM (Data Distribution Management) service groups.

The HLA TM service group ensures a Time-Stamp-Ordered (TSO) message delivery sequence and correct time advancement of each simulation component in an HLA-based distributed simulation application. A traditional TM algorithm can be either synchronous

or asynchronous. However, both synchronous and asynchronous algorithms have their own drawbacks. To resolve the drawbacks of each algorithm, this thesis describes a hybrid TM algorithm that combines synchronous and asynchronous algorithms. The three TM algorithms have been implemented into SOHR and experimental results show that the hybrid algorithm effectively combines the advantages of both synchronous and asynchronous algorithms.

The HLA DDM relies on the computation of overlap between update and subscription regions, which is called matching. This has led to the development of an efficient sort-based DDM matching algorithm in this project. Later, this algorithm was extended to one which incrementally matches for dynamic region modifications. Theoretical analysis concludes that, in a large spatial environment, the efficient sort-based algorithm should have good storage and computational scalability; the extended efficient sort-based algorithm generally has more efficient dynamic matching computational performance, though it requires more storage space and has less efficient static matching computational performance. The experimental results have verified the theoretical conclusions by comparing them with the region-based and Raczy's sort-based matching algorithms.

The extended efficient sort-based DDM approach and the grid-based DDM approach have then been implemented into SOHR. Experiments have been carried out to compare the performance of the two implementations in different scenarios.

Finally, as a case study, this thesis describes SOHR's support for multi-user gaming on the Grid. A maze game is used to illustrate how SOHR enables users to join a game conveniently across administrative domains. Experiments have been carried out to show how DDM can improve the communication efficiency and how the plug-and-play paradigm of SOHR enables us to choose the best DDM approach for a specific scenario.

List of Figures

List of Tables

Index of Abbreviations

AABB	Axis-Aligned Bounding Box
ALSP	Aggregate Level Simulation Protocol
AMG	Architecture Management Group
API	Application Programming Interface
ATIM	Average Number of Time Information Messages
ATPI	Average Time Per Iteration
AUM	Attribute Update Message
AUMVP	Attribute Update Message Valid Percentage
BS	Broker Service
C	Constrained
C4I	Command, Control, Communication, Computers and Intelligence
CI	Conditional Information Report
DARPA	Defense Advanced Research Projects Agency
DDM	Data Distribution Management
DDMFS	Data Distribution Management Factory Service
DDMIS	Data Distribution Management Instance Service
DDMRI	Data Distribution Management Resource Instance
DDMS	Data Distribution Management Service
DIS	Distributed Interactive Simulation
DM	Declaration Management

DMFS	Declaration Management Factory Service
DMIS	Declaration Management Instance Service
DMRI	Declaration Management Resource Instance
DMS	Declaration Management Service
DMSO	Defense Modeling and Simulation Office
DoD	Department of Defense
DSRT	Distributed Simulation and Real-Time Applications
EESB	Extended Efficient Sort-Based
EPR	End Point Reference
FDD	FOM Document Data
FDK	Federated Simulations Development Kit
FED	Federation Execution Data
FEDEP	Federation Development and Execution Process
FIFO	First-In First-Out
FM	Federation Management
FMFS	Federation Management Factory Service
FMIS	Federation Management Instance Service
FMRI	Federation Management Resource Instance
FMS	Federation Management Service
FOM	Federation Object Model
FQR	Flush Queue Request
GALT	Greatest Available Logical Time
GSI	Grid Security Infrastructure

GT	Globus Toolkit
G-HLAM	Grid HLA Management System
HLA	High Level Architecture
IDSIM	Interoperable Distributed Simulation
IFSpec	Interface Specification
JVM	Java Virtual Machine
LAN	Local Area Network
LB	Lower Bound
LBTS	Lower Bound Time Stamp
LFS	Local Factory Service
LH	LookaHead
LIS	Local Instance Service
LP	Logical Process
LRC	Local RTI Component
LRI	Local Resource Instance
LS	Local Service
LT	Logical Time
LU	List Update
maxBS	Maximum Bound Shift
maxRS	maximum Range Size
maxSRRS	max Subscription Region Range Size
maxURRS	max Update Region Range Size
MAS	Multi Agent Systems

NG	Next Generation
NMR	Next Message Request
NMRA	Next Message Request Available
NTU	Nanyang Technological University
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
OMT	Object Model Template
OOM	Object and Ownership Management
OOMFS	Object and Ownership Management Factory Service
OOMIS	Object and Ownership Management Instance Service
OOMRI	Object and Ownership Management Resource Instance
OOMS	Object and Ownership Management Service
PDA	Personal Digital Assistant
R	Regulating
RO	Receive Order
RTI	Run Time Infrastructure
RUM	Region Update Message
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOAr-DSGrid	Service-Oriented Architecture for Distributed Simulation on the Grid
SOHR	Service Oriented HLA RTI
SOM	Simulation Object Model

SR	Subscription Region
STOW	Synthetic Theater Of War
TAG	Time Advance Grant
TAR	Time Advance Request
TARA	Time Advance Request Available
TIM	Time Information Message
TM	Time Management
TMFS	Time Management Factory Service
TMIS	Time Management Instance Service
TMRI	Time Management Resource Instance
TMS	Time Management Service
TS	Time Stamp
TSO	Time Stamp Order
UB	Upper Bound
UC	UnConstrained
UI	Unconditional Information Report
UR	UnRegulating
UR	Update Region
WAN	Wide Area Network
WS	Web Services
WSDL	Web Services Description Language
WSPRC	Web Service Provider RTI Component
WSRF	WS Resource Framework

XML	Extensible Markup Language
XMSF	Extensible Modeling and Simulation Framework

Chapter 1

Introduction

1.1 Background

1.1.1 Simulation and Parallel / Distributed Simulation

Modeling has been used for many years by scientists and engineers as a means to represent or abstract salient characteristics of complex systems or concepts. Simulation is the process to observe the behavior of a system by operating on its model. Computer simulation refers to a type of simulation, in which a computer program is created to represent and emulate the behavior of a physical system over time. It is a low-cost and safe alternative to solve complex problems in various areas such as production, business, education, science and engineering.

With increased problem size, parallel simulation was introduced to increase simulation speed. In a parallel simulation, the whole simulation is divided into some number of logical processes, each of which is run on a node inside a parallel computing infrastructure such as a shared memory multi-processor platform or a cluster with a high-speed internal connection. To promote the interoperability and reusability of simulation applications and link geographically dispersed simulation components, distributed simulation was introduced. It is similar to parallel simulation in that it consists of some number of logical processes, but the difference is that its logical processes are run on machines that may be

geographically distributed across a building, university campus, or even the world [FUJ00]. While the main concern of parallel simulation is execution speedup, distributed simulation is mainly concerned with the reuse and interoperability of different simulation components. The communications between simulation components of a distributed simulation application are normally through a communication middleware. To achieve the aim of reuse and interoperability, some standard needs to be defined for interfacing simulation components to this middleware. The High Level Architecture (HLA) has been standardized as IEEE 1516 for distributed simulation in September 2000 [IEEE1516]. While the HLA defines the rules [IEEE1516], interface specification [IEEE1516.1] and Object Model Template (OMT) [IEEE1516.2], an RTI (Run Time Infrastructure), such as the Defense Modeling and Simulation Office (DMSO) RTI [DMSO], provides the actual implementation of the HLA standard. An introduction to the HLA is given next.

1.1.2 Introduction to the High Level Architecture

1.1.2.1 History

The roots for the High Level Architecture (HLA) stem from the previous Distributed Interactive Simulation (DIS) standard [DIS95] and Aggregate Level Simulation Protocol (ALSP) [ALSP94]. Its development began in October 1993 with three industrial contracts awarded by the Defense Advanced Research Projects Agency (DARPA) to develop a common architecture that could encompass the Department of Defense (DoD) modeling and simulation community. An initial design was received in January 1995 and a group called the Architecture Management Group (AMG) was formed by the DMSO to oversee the development of the HLA. In March 1995, an initial architecture proposal was presented to the AMG when the HLA began to take form. The AMG developed the HLA based on cooperative efforts and applied it in a series of prototype federations to ensure that it can address a broad set of application requirements. In August 1996, the baseline HLA definition was completed and it continued evolution based on later experience. In February 1998, the HLA came to its Version 1.3 which formed the basis for a draft IEEE

standard for distributed simulation. Finally, the HLA was standardized as IEEE 1516 in September 2000 [IEEE1516]. It is compulsory that every IEEE standard needs to be reviewed at least once every five years to reflect the present state of the art. Nowadays, the specification of a new standard called HLA Evolved is on its way.

1.1.2.2 Overview

As discussed previously, a distributed simulation application consists of several simulation components which are also called logical processes. In HLA terminology, a distributed simulation application is called a federation and a simulation component is called a federate. A federate can be any type of computer simulator including a human-in-the-loop simulator which reacts to input from a person in real-time or a hardware-in-the-loop simulator which incorporates an embedded physical device. A Run Time Infrastructure (RTI) is the communication middleware which provides the actual implementation of the HLA standard. Federates in the same federation communicate with each other by invoking respective services provided by the underlying RTI.

The HLA standard comprises four major elements:

1. The HLA framework and rules [IEEE1516] defines the HLA, its components, five federation rules and five federate rules which outline the responsibilities of federations and federates respectively.

2. The HLA Interface Specification (IFSpec) [IEEE1516.1] defines the standard services of and interfaces to the HLA RTI. The Interface Specification includes six HLA service groups:

- Federation Management – manages the creation, dynamic control, modification, and deletion of a federation execution;
- Declaration Management – controls the exchange of instance attribute updates and interactions between federates with a class-based publication and subscription mechanism;

- Object Management – deals with the registration, updates to, and deletion of object instances and the sending and receiving of interactions;
- Ownership Management – manages the transfer of attribute ownership of object instances between federates;
- Time Management – controls the advancement of simulation time and ensures the correct delivery order of messages throughout the federation execution;
- Data Distribution Management – refines the data requirements at the instance attribute or interaction level to further reduce the irrelevant traffic between federates.

3. The HLA Object Model Template (OMT) [IEEE1516.2] defines the format and syntax of HLA object models. Each federate of a federation should have a Simulation Object Model (SOM) and the whole federation should have a common Federation Object Model (FOM). The SOM and FOM are used to evaluate the reusability and interoperability of simulation applications.

4. The Federation Development and Execution Process (FEDEP) [IEEE1516.3] defines the processes and procedures that should be followed to develop and execute HLA federations. It is intended to be used as a high-level framework, into which low-level management and system engineering practices native to HLA user organizations can be integrated. Its top-level view comprises the following seven basic steps, each of which can be implemented depending on a specific federation scenario.

- Define federation objectives
- Perform conceptual analysis
- Design federation
- Develop federation
- Plan, integrate, and test federation
- Execute federation and prepare outputs
- Analyze data and evaluate results

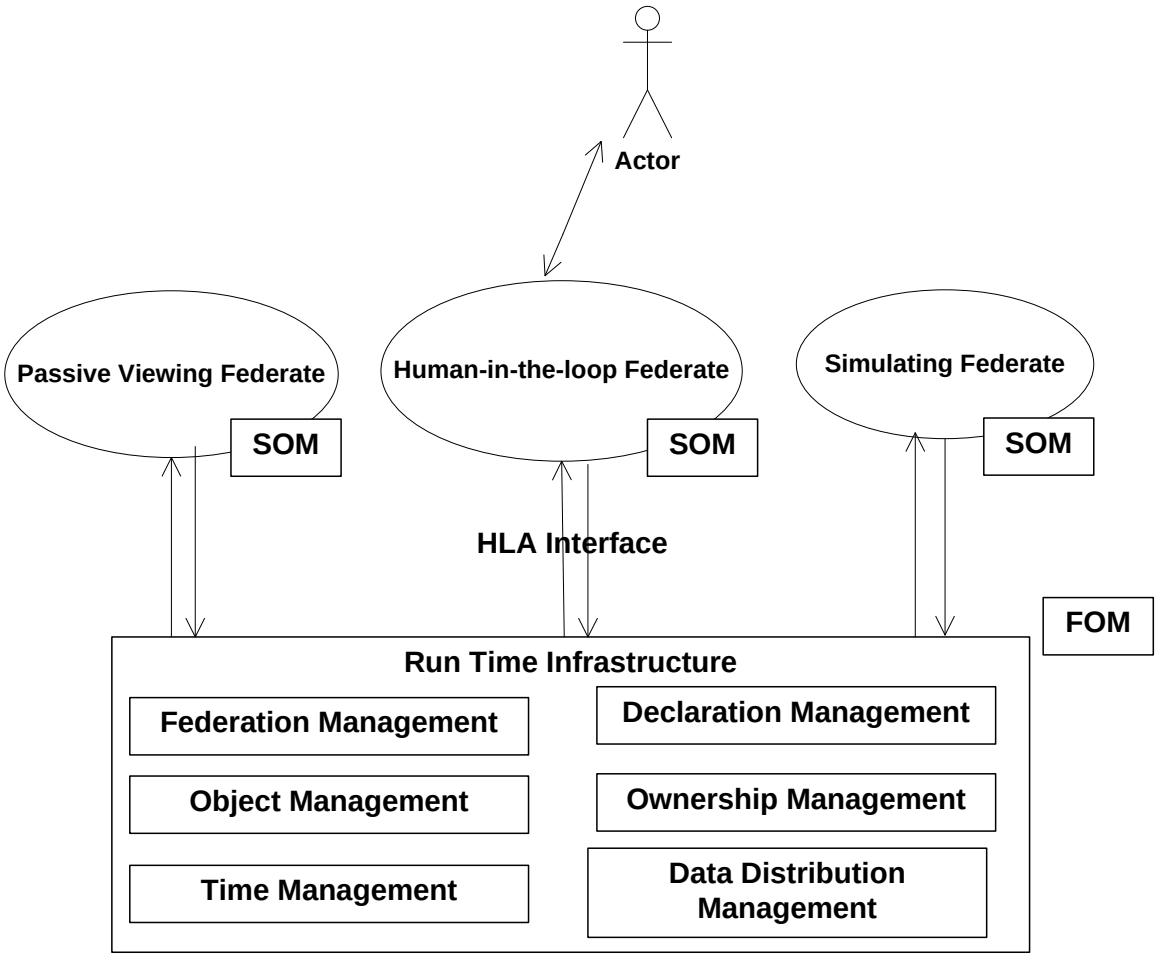


Figure 1.1.1: An overview of an HLA federation (derived from [DMSO])

An overview of an HLA federation is shown in Figure 1.1.1. It consists of:

- some federates each of which documents its supported types of objects, attributes and interactions in a SOM using the OMT;
- a FOM which describes the set of shared objects, attributes and interactions across a federation using the OMT;
- the RTI software which implements the interface specification of the HLA and provides the six groups of HLA services to the federates.

The HLA views a federation as a group of distributed objects which are managed by federates. The characteristics of an object are represented by its attributes. A federate can manage multiple objects and an object can be managed by multiple federates with one federate owning some attributes of the object. A federate periodically updates the instance

attributes which it owns by requesting the corresponding RTI service and the RTI is responsible for sending these messages to the expected receivers. Besides instance attribute updates, the HLA supports another message exchange mechanism called interaction, which is characterized by its parameters and used to exchange non-persistent events between federates. The supported types of objects, attributes, interactions and parameters in a federation are a subset of the FOM and are defined in the Federation Execution Data (FED) file in HLA 1.3 or FOM Document Data (FDD) file in HLA 1516. The HLA interface between federates and the RTI is bi-directional based on an ambassador paradigm as shown in Figure 1.1.2. A federate requests RTI services by invoking the corresponding functions of its RTIambassador object instance. When the federate joins a federation, it provides the RTI with a FederateAmbassador object instance so that the RTI can deliver callbacks to it by invoking the corresponding functions of the provided FederateAmbassador.

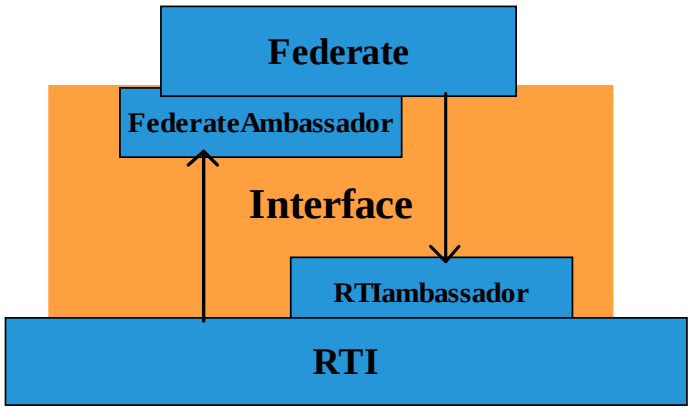


Figure 1.1.2: The HLA interface between federates and the RTI (derived from [DMSO])

1.1.2.3 Time Management (TM) in the HLA

The TM service group ensures a Time-Stamp-Ordered (TSO) message delivery sequence and correct time advancement of each federate in a federation [FUJ98]. A federate can be set to be regulating/unregulating (R/UR) and constrained/unconstrained (C/UC). A regulating federate regulates time advancement of all constrained federates. The time

advancement of a constrained federate is regulated by all regulating federates. Normally, a TM algorithm can be implemented in either a centralized or distributed manner. As centralized algorithms can cause system bottlenecks, this project only focuses on the performance of distributed TM algorithms.

To control time advancement of a federation, a distributed TM algorithm requires each regulating federate to periodically propagate its local time information, which specifies a lower bound on the time stamps (TS) of messages it may send in the future, to all constrained federates. Based on lower bounds from all regulating federates, each constrained federate calculates a lower bound on the time stamps of messages it may receive in the future, which is called Greatest Available Logical Time (GALT) in HLA 1516 (or Lower Bound Time Stamp (LBTS) in HLA 1.3). Depending on whether a specified lower bound by a regulating federate can be guaranteed to be true conditionally or unconditionally, the time information propagated is called conditional information or unconditional information [FUJ00]. Traditionally, there are asynchronous and synchronous distributed TM algorithms [FUJ00]. An asynchronous TM algorithm generally uses unconditional information for the GALT calculation. Since unconditional information is always true, each constrained federate is able to calculate its GALT asynchronously based on unconditional information received from all regulating federates. A synchronous algorithm generally uses conditional information for the GALT calculation. As conditional information is not definitely true, a constrained federate needs to wait to receive conditional information from each regulating federate, normally in a barrier synchronization, before a new GALT can be calculated.

1.1.2.4 Data Distribution Management (DDM) in the HLA

DDM provides a set of services to reduce message traffic over the network and irrelevant processing by receiving federates [BOU03, MOR97]. It allows data producers (sending federates) to specify their update regions in a universal multi-dimensional routing space [IEEE1516, MOR01]. It also allows data consumers (receiving federates) to specify their

subscription regions in the same universal multi-dimensional routing space. Data are directed from sending federates to receiving federates if and only if there is an overlap between the update and subscription regions. A region is specified as a range on each dimension of a subset of all dimensions in the universal multi-dimensional routing space. An update region and a subscription region overlap if and only if they have common dimensions and their ranges overlap on each common dimension. An example is shown in Figure 1.1.3. There is one update region U1 and two subscription regions S1 and S2. All the three regions are specified using dimensions X and Y in the universal routing space. U1 overlaps with S1 because their ranges on both dimensions X and Y overlap with each other. U1 does not overlap with S2 because there is no overlap between their ranges on dimension X. This means S1 will receive data from U1 but S2 will not.

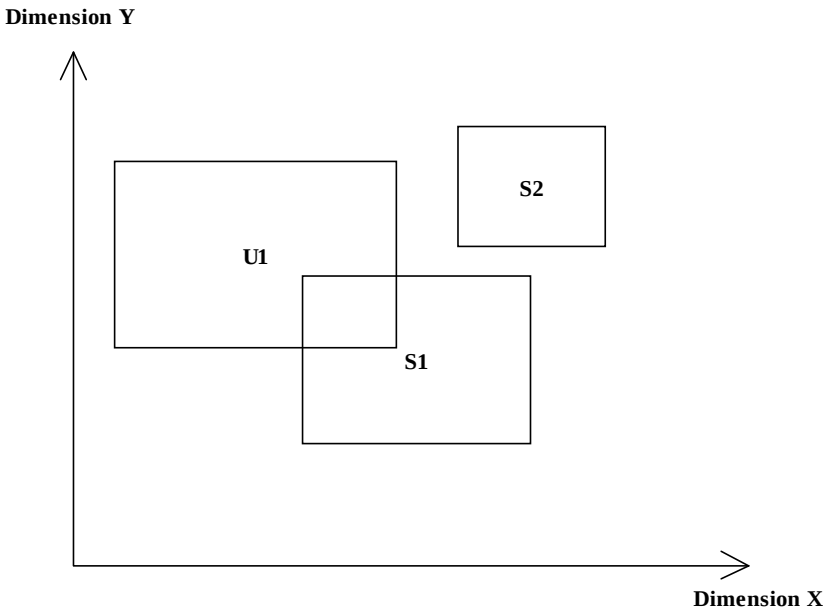


Figure 1.1.3: An example of overlapping regions

The DDM computation can be decomposed into the following four distinct processes [PET2004].

- Declaring: The federates create, modify or delete their respective update and subscription regions throughout a federation execution as their data requirements change. A change originates from a federate and other parts of the federation need to

be informed of it for the next process, matching.

- Matching: The RTI derives the overlaps between update and subscription regions with a matching algorithm.
- Connecting: Based on the result of the matching process, the RTI establishes network data flow connectivity depending on the network infrastructure.
- Routing: The RTI transports data from the sending federates to the appropriate receiving federates using the connectivity established during the connecting process. The efficiency of the routing process depends on the exactness of the matching process. An inexact matching algorithm results in many irrelevant data communications in the routing process.

1.1.3 Grid Computing

Grid computing was proposed by Ian Foster as flexible, secure and coordinated resource sharing among dynamic collections of individuals, institutions, and virtual organizations [FOS01b]. It aims to disintegrate the numerous barriers between different computing systems within and across organizations so that ubiquitous resources can be shared across administrative domains while retaining the local autonomy of each domain.

To realize the Grid vision, standardization is needed to enable discovery, access, allocation, and monitoring of heterogeneous distributed resources and manage them as a single virtual system. The Open Grid Services Architecture (OGSA) addresses this need for standardization by defining a set of core capabilities and behaviors that address key concerns in Grid Systems [FOS01a, FOS05]. These capabilities are realized by the definition of a collection of key services, the interfaces these services expose and the interaction between them within a Service-Oriented Architecture (SOA). These services include Execution Management Services, Data Services, Resource Management Services, Security Services, Self-Management Services and Information Services in the specification of OGSA Version 1.0 [FOS05]. They are based on the virtualization of diverse distributed resources including CPU cycles, memory, network capacity, software

and any other shared artifacts, entities or knowledge. To be part of an OGSA Grid, all services must implement a core set of interfaces and standards which are collectively referred to as the infrastructure services or the Grid fabric. Since a SOA is assumed by OGSA, the Grid fabric includes core Web Services (WS) standards. The Web Services Description Language (WSDL) [WSDL] is used to define service interfaces. The Extensible Markup Language (XML) [XML] is used for representations and the Simple Object Access Protocol (SOAP) [SOAP] is used as the primary message exchange format. Since the WS standards cannot meet all Grid requirements, extensions to existing specifications such as the WS-Security or new specifications are needed. A key Grid requirement which has motivated new specifications is the ability to represent and manipulate state. Due to this requirement, the WS Resource Framework (WSRF) [WSRF] has been proposed for modeling, accessing and managing state, grouping services and expressing faults. The WSRF is part of the Grid fabric and it is a successor of the Open Grid Services Infrastructure (OGSI) [TUE03].

An attractive point of the WSRF is that information (or state) can be organized into resource instances based on its WS-Resource factory design pattern [SOT] as shown in Figure 1.1.4. A resource home keeps track of the multiple resource instances each of which is identified by a unique ID. A factory service is defined to create new resource instances based on the client's requests. An instance service uses the resource home to find the resource instance specified by the client and operates on it.

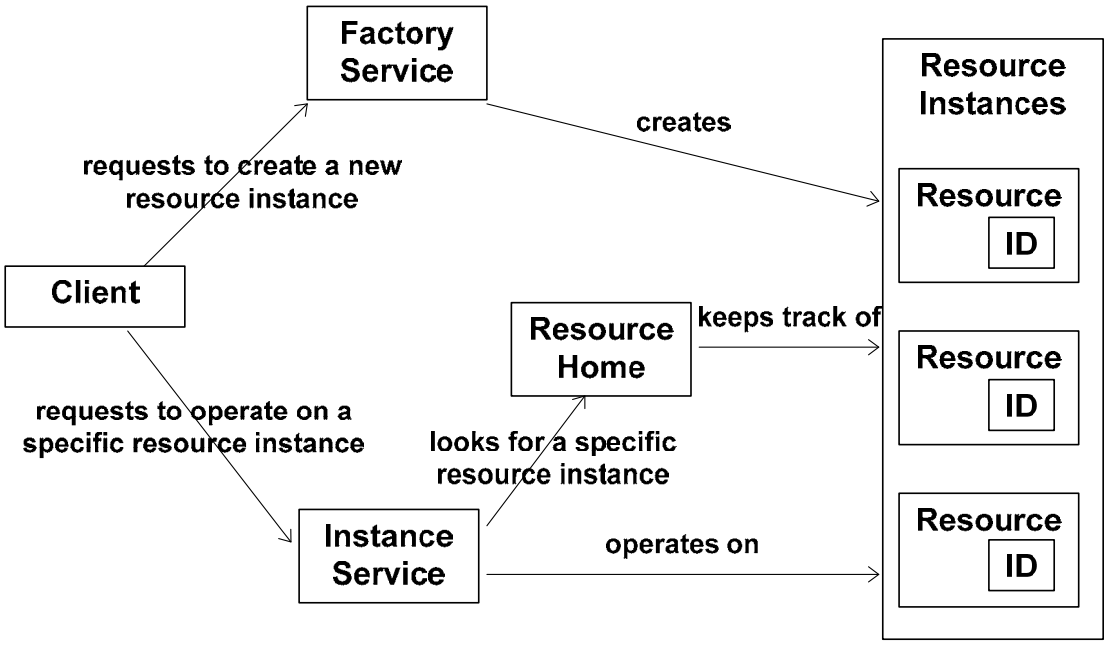


Figure 1.1.4: The WS-Resource factory design pattern

Among Grid middlewares, Globus Toolkit (GT) [GT] is the de facto standard middleware for Grid computing. Its previous version GT3 provided a partial implementation of the OGSA based on the OGSI. Its newest version GT4 provides an OGSA implementation based on the WSRF and contains five major components, namely Common Runtime Components, Security, Data Management, Information Services and Execution Management.

1.1.4 Distributed Simulation on the Grid

Traditionally, HLA-based distributed simulations are conducted using a vendor-specific RTI software and federates with different RTI versions cannot cooperate with each other. To run a distributed simulation, the required software and hardware resource arrangements and security settings must be made before the actual simulation execution. Because of this inflexibility, it is not easy to run HLA-based distributed simulations across administrative domains and most of the distributed simulation applications are conducted inside a LAN (Local Area Network). However, there is large demand for conducting HLA-based distributed simulation across a WAN (Wide Area Network). For example, to run an

application where a specific federate can only be executed at a fixed site due to a specific resource requirement or security issues, the only solution is to enable HLA-based distributed simulation across WANs. The ability to conduct distributed simulation on a WAN also intrinsically enables large scale applications utilizing the numerous WAN resources.

We define three approaches for HLA-based distributed simulation on the Grid, namely a Grid-facilitated approach, a Grid-enabled approach and a Grid-oriented approach. In the Grid-facilitated approach, Grid services are defined to facilitate the execution of HLA-based distributed simulations while the actual simulation communications are through a vendor-specific HLA RTI. For HLA-based distributed simulations to be conducted across administrative domains, this approach requires cross-domain trust and particular prior security setup depending on the RTI version, which is very cumbersome. In the Grid-enabled approach, Grid (or web) service interfaces are provided to enable HLA-based distributed simulations to be conducted in a Grid (or web) environment. A client federate on the Grid (or web) communicates with a federate server using Grid (or web) service communications and the federate server representing the client federate joins an HLA-based distributed simulation using a vendor-specific HLA RTI. The major drawback of this approach is that a vendor-specific RTI execution environment and federate servers must be set up beforehand, which makes this approach not so flexible. In the Grid-oriented approach, the RTI itself is implemented using Grid services according to the HLA specification. All communications are through Grid service invocations. This approach was raised in Fox's keynote at Distributed Simulation and Real-Time Applications (DS-RT) 2005 [FOX05].

1.2 Motivation of the Project

In traditional HLA-based distributed simulation, each federate relies on a local RTI component to do RTI-specific processing such as time synchronization and DDM matching. The local RTI component is a library that is linked with the federate and

executes on the same processor. However, RTI-specific processing itself can involve a heavy computational load and the installation of a local RTI component on each federate host is not always feasible especially on resource limited platforms. To save the host's processing power for the federate's own simulation computation and to provide an environment for pervasive distributed simulation, RTI-specific tasks have to be offloaded from federates. The question is to where. Santoro and Fujimoto have proposed offloading RTI-specific tasks to GPUs [SAN08]. In this project, we explore how RTI-specific tasks may be distributed over the various Grid resources.

The main aim of this project is therefore to research on how to build an HLA-based distributed simulation framework on the Grid following the Grid-oriented approach discussed in Subsection 1.1.4. With this, RTI-specific functionalities are offloaded to the available Grid resources so that a federate can spend most of its host's computation power on its own simulation activities. With Grid communications executed on top of the HTTP protocol, which can traverse firewalls, it is more convenient to run distributed simulations across administrative domains on a WAN (Wide Area Network). This increases the ease of cooperation between simulation models from different organizations. Moreover, the various Grid service components can be dynamically deployed, discovered and undeployed on demand. All these features enable scalable execution of HLA-based distributed simulations on a WAN.

There are six HLA service groups among which the TM and DDM service groups have drawn much research attention because their efficiency significantly affects the performance of a distributed simulation application. This is particularly true in a Grid environment as Grid resources are normally shared among different organizations and Grid communications normally involve long distances and have relatively large overhead. For either the TM service group or the DDM service group, there are many existing algorithms and, as we will discuss in detail in Chapters 4, 5 and 6, there is no single algorithm that is suitable for all federation scenarios. Our vision is to build a Service

Oriented HLA RTI (SOHR), in which different HLA algorithms can coexist by creating different Grid services for them. A user is able to choose the most suitable algorithms to be used for a specific federation scenario so as to optimize performance.

1.3 Objectives of the Project

With the motivation discussed in Subsection 1.2, the following objectives are defined for this project.

1.3.1 A Service Oriented HLA RTI (SOHR) Framework

A Service Oriented Architecture describes a distributed computing system which comprises a set of loosely coupled services to provide specific functionalities to users. Each of the services can be accessed without its implementation details and a typical discovery service acts as the coordinator of the system for dynamic discovery of various services and queries about their interfaces. New applications can be created by combining existing services. With a list of available services, a user is able to make a choice among them, which is flexible. Due to all these features, SOA is suitable for building complex network-based enterprise systems.

With a Service Oriented HLA RTI (SOHR), we can create a separate Grid service for a different algorithm of a particular service group such as the DDM service group [VAN98, RAK96, TAN00, RAC05]. A user is able to choose the most suitable Grid service based on a specific federation scenario to optimize its performance. New algorithms can easily be included by deploying their corresponding Grid services. This creates a plug-and-play paradigm for an RTI implementation and makes our SOHR efficient and extensible.

In order to accomplish this objective, tasks include the definition of a set of key Grid services and their interactions between each other based on the existing WS-* specifications (e.g., WSRF). When doing this, some properties are required: (1) the

standard HLA interfaces need to be provided to federates to enable legacy HLA-based federates to work in SOHR without any modification; (2) the federate-side library should be kept light-weight in order to enable federates to be run on heterogeneous platforms such as PDAs and cellphones; (3) a strict SOA approach should be followed to separate the six HLA service groups into different Grid services so that different algorithms for a particular HLA service group can be implemented and deployed in SOHR creating a plug-and-play paradigm of an RTI implementation; (4) other Grid services should be easily integrated into SOHR for more functionalities, such as checking-pointing services for fault tolerance purposes, so as to make SOHR an extensible framework. A prototype of SOHR is implemented based on GT4 [GT] and its performance is compared with commercial RTIs such as the DMSO RTI [DMSO].

1.3.2 A New Algorithm for the HLA TM Service Group

Since SOHR enables new algorithms for a service group to be easily included in our distributed simulation environment, we also aim to create more efficient HLA algorithms particularly for a Grid environment.

A traditional TM algorithm can be either synchronous or asynchronous. In general, an asynchronous algorithm utilizes unconditional information and a synchronous algorithm utilizes conditional information. This has already been introduced in Subsection 1.1.2.3.

It is well-known that an asynchronous algorithm has a “time creep” problem when the lookahead is small. This will be discussed in detail in Subsection 2.2.1.

On the other hand, a synchronous algorithm has its own drawback that its time advancement may be blocked by a regulating federate which sends conditional information with a low frequency. The low frequency situation may occur when the execution of an application is not balanced, as in a simulation application executed on the Grid where computing nodes have different temporary work loads. It also occurs when the

application itself is not balanced in that some components require more computation than others. For example, in a military training application, a trainee federate may need to display visual effects when hit by a bomb while a computer generated soldier federate may not. In this case, the trainee federate has more computational work than the soldier federate when processing the bomb event. The trainee federate may become a low frequency federate when high-quality visual effects are displayed. In addition to poor load balancing, the low frequency situation may also be caused by communication latency between federates. For example, in a distributed simulation scenario where federates at a local site receive data from and control a sensor at a remote site, the federates at the local site are close to each other and thus have less communication delay between each other, while the sensor federate sends its conditional information with a large delay and thus becomes a low frequency federate.

To address the drawbacks of the traditional synchronous and asynchronous TM algorithms, this project aims at the development of a hybrid algorithm based on both conditional and unconditional information by combing traditional synchronous and asynchronous algorithms together. Experiments are carried out to compare the performance of the three TM algorithms in SOHR.

1.3.3 New Approaches for the HLA DDM Service Group

As the communications in SOHR are through Grid service invocations which are costly, there is an important requirement to improve the communication efficiency in SOHR. As introduced in Subsection 1.1.2.4, the HLA DDM service group aims at optimizing communication efficiency between federates. There are various DDM approaches which will be discussed in Subsection 2.3. In order to improve the data communication efficiency in SOHR, this project also aims to develop new DDM approaches which are more efficient than the existing ones. Firstly, more efficient DDM matching algorithms are developed. In order to avoid irrelevant data communication in the DDM routing process, the new matching algorithms should be exact. Next, with the plug-and-play

feature of SOHR, the new DDM matching algorithms need to be implemented in SOHR and compared with other approaches.

1.3.4 Case Study

To illustrate how an existing distributed simulation can be adapted to use SOHR as its underlying communication infrastructure, a case study of multi-user gaming is carried out. The case study also demonstrates the major advantages of SOHR. These include offloading RTI-specific tasks to Grid resources, the convenience of running an application across administrative domains on a WAN and the plug-and-play paradigm that enables users to choose the most suitable HLA algorithm for a specific federation scenario.

1.4 Structure of the Thesis

The thesis is organized into 8 chapters.

Chapter 2 first provides an overview of related work in distributed simulation on the Grid. Three approaches can be defined for HLA-based distributed simulation on the Grid and related projects are introduced for each of the approaches. Then a literature review is given on traditional TM algorithms. After that, major existing DDM approaches are described with their respective advantages and disadvantages. Since the case study is concerned with SOHR's support for multi-user gaming, related work on HLA-based multi-user gaming is then reviewed.

Chapter 3 discusses a Service Oriented HLA RTI (SOHR) framework which implements an HLA RTI using Grid services. The design of SOHR is described in detail. A prototype has been implemented with a subset of the HLA specification. The messaging performance of the SOHR prototype and the DMSO RTI is compared and analyzed based on the results of ping-pong experiments on both a Local Area Network (LAN) and a Wide Area Network (WAN).

Chapter 4 discusses a hybrid TM algorithm that combines traditional synchronous and asynchronous algorithms. The algorithm is described in detail and its performance is compared with traditional TM algorithms on both a LAN and a WAN.

Chapter 5 firstly describes an efficient sort-based DDM matching algorithm for HLA applications with a large spatial environment, and then extends it to an algorithm which incrementally matches for dynamic region modifications. Theoretical analysis is given for both algorithms and their matching performance is compared with existing matching algorithms.

Chapter 6 discusses the implementation of two DDM approaches, the grid-based approach and the extended efficient sort-based approach, in SOHR. Experiments have been carried out to compare their performance in different scenarios.

To show the advantages of the SOHR framework, Chapter 7 describes a case study of SOHR to support multi-user gaming on the Grid.

Finally, Chapter 8 concludes the whole thesis.

Chapter 2

Related Work

This chapter discusses work related to this project, categorized according to the objectives of the project, defined in Subsection 1.3. There are four categories, namely Distributed Simulation the Grid, TM algorithms, DDM approaches and HLA-Based Multi-User Gaming. Each category is discussed in detail as follows.

2.1 Distributed Simulation on the Grid

2.1.1 HLA-Based Distributed Simulation on the Grid

We define three approaches for HLA-based distributed simulation on the Grid, namely a Grid-facilitated approach, a Grid-enabled approach and a Grid-oriented approach. Each of these approaches is discussed in separate subsections as follows.

2.1.1.1 Grid-facilitated Approach

In the Grid-facilitated approach, Grid services are defined to facilitate the execution of HLA-based distributed simulations while the actual simulation communications are through a vendor-specific RTI. Several projects using this approach are introduced in this subsection.

The Grid HLA Management System (G-HLAM) was proposed by Rycerz for efficient

execution of HLA-based distributed simulations on the Grid [RYC06]. HLA Interfacing Services are defined for interfacing and managing federates or RTIExec processes on each site, such as saving and restoring a federate for migration. Broker Support Services are defined and they reside on each federate site providing necessary information about the local federate such as the current host load and application behavior. The Migration Service is defined to coordinate federate migration between source and destination sites by communicating with their Interfacing Services. To decide when and where federate migration should be performed, collective services are defined including the Application Monitoring Main Service Manager, the Performance Decision Service and the Broker Service. They communicate with the Broker Support Services of each federate for global performance monitoring, decision making and migration triggering. Though the various Grid services cooperate for an optimized configuration of federation execution, the actual simulation communications are through a vendor-specific RTI.

Wu et al. proposed a data Grid called Aegis for large scale distributed simulation applications. It offers data resource management services and computing services for HLA-based simulation applications while the simulation execution still uses a vendor-specific RTI [WU04].

Choi et al. proposed an RTI execution environment called RTI-G based on the OGSA [CHO05]. In order to achieve high performance, RTI-G utilizes various GT3 services, such as MDS, GRAM and GridFTP, to enable dynamic resource allocation and automatic execution of HLA-based distributed simulations.

Simulation Grid is an infrastructure, which integrates Grid technology and the HLA to realize dynamic and secure resource sharing, optimized resource utilization, collaborative activities and fault tolerance for distributed simulation applications [CHA06, XIA06]. In Simulation Grid, HLA-based distributed simulations are Grid-facilitated in various ways. For example, registration services are defined for dynamic model resource discovery and

scheduling services are defined to generate proper simulation deployment schemes according to monitored computing resource status.

Our group proposed a load management system for HLA-based distributed simulations in [CAI02a]. The load management system and its federate migration mechanism are based on Grid services while communication between federates are through a vendor-specific RTI. It was later extended with an efficient migration protocol in [CAI05]. We also proposed a Grid service based framework for flexible execution of large scale HLA-based distributed simulations [ZON04]. RTI executive processes and federate models are encapsulated in Grid services for dynamic federation creation and management. An index service is provided for registration and dynamic discovery of the various Grid services. Simulation communications are also through a vendor-specific RTI.

As the Grid-facilitated approach still relies on a vendor-specific RTI for simulation communication, it requires cross-domain trust and particular prior security setup for HLA-based distributed simulations to be conducted across administrative domains, which is very cumbersome.

2.1.1.2 Grid-enabled Approach

In the Grid-enabled approach, Grid (or web) service interfaces are provided to enable HLA-based distributed simulations to be conducted in a Grid (or web) environment. There are various forms of this approach. One form is that a client federate communicates with a federate server using Grid (or web) service communications and the federate server representing the client federate joins an HLA-based distributed simulation using a vendor-specific RTI. Another form is that different federations are executed using a vendor-specific RTI at their own local sites and Grid service interfaces are defined to enable communications between the federations so that a larger federation community can be formed over the Grid.

An example of the first form of this approach is the work done by the XMSF group [XMSF] to integrate simulations with other applications using web services [PUL05]. Web service interfaces are provided for a particular simulation application so that other applications on the Web can interact with the simulation application. For example, a remote passive visualizer can periodically probe the simulation progress and display it accordingly; when the progress is not satisfactory, an army C4I (Command, Control, Communications, Computers and Intelligence) system can send a command to the simulation application for adjustment. The communications between the simulation application and the other applications are through web service invocations while the communications inside the simulation application are through a vendor-specific RTI. As another example of the first form of this approach, new web service APIs have recently been proposed for the HLA Evolved standard [MOL06]. With the web service APIs, a client federate on the web communicates with a Web Service Provider RTI Component (WSPRC), through which the federate joins a federation.

The second form of this approach exists in several existing works. For example, Zhu et al. proposed a Grid-based Distributed Simulation Architecture (GDSA) [ZHU06], in which Grid service interfaces are defined for a federation so that multiple federations can be integrated through the defined interfaces; Xu and Peng developed a Grid-based Simulation Service Bus (SSB) [XU06], through which a remote federate on the Grid can join a local federation in a LAN and multiple federations are able to exchange data over the Grid. Simulation Grid [CHA06] is a combination of the Grid-facilitated and Grid-enabled approaches. In addition to its Grid-facilitated features discussed in Subsection 2.1.1.1, it also exhibits the second form of the Grid-enabled approach in that RTI Grid service components are deployed to provide non-real-time communication services between different federations.

To run HLA-based distributed simulations on the Grid, our group proposed a Federate-Proxy-RTI based framework [XIE05]. Proxies are provided so that a remote

federate on the Grid is able to join a local federation through Grid service communications with a proxy. All proxies and local federates reside in a LAN and communicate with each other through a vendor-specific RTI. This effort also follows the Grid-enabled approach. Recently, Chen et al. integrated the Federate-Proxy-RTI based framework with the HLA RePast middleware system [MIN04] and thus developed an HLA Grid RePast platform for executing large scale agent-based distributed simulation on the Grid [CHE08].

The major drawback of the Grid-enabled approach is that vendor-specific RTI execution environments and communication servers have to be set up beforehand, which makes this approach not so flexible.

2.1.1.3 Grid-oriented Approach

In the Grid-oriented approach, the RTI itself is implemented using Grid services according to the HLA specification. The six HLA service groups should be mapped to different Grid services in order to create a service oriented architecture. This approach was raised in Fox's keynote at DSRT 2005 [FOX05].

The SOHR framework described in this thesis follows the Grid-oriented approach. Its design is discussed in detail in Section 3.

2.1.2 Non-HLA-Based Distributed Simulation on the Grid

Besides HLA-based work, there have also been some efforts to utilize the Grid in non-HLA-based distributed simulations. For example, Fitzgibbons et al. proposed an Interoperable Distributed Simulation (IDSim) framework [FIT04] based on the Open Grid Services Infrastructure (OGSI) [TUE03]. It provides distributed simulation services to federated simulators and has a high degree of extensibility due to the utilization of the OGSI and web technology standards. Our group proposed a Service-Oriented Architecture

for Distributed Simulation on the Grid (SOAr-DSGrid) [CHE06]. It is a component-based distributed simulation framework in which each simulation component is implemented as a GT4 service conforming to a component interface schema. A new simulation application can be developed by composition of existing simulation components, which enables collaborative development and lowers development cost.

2.2 Traditional TM Algorithms

As introduced in Subsection 1.1.2.3, a traditional distributed TM algorithm can be either asynchronous or synchronous. In general, asynchronous algorithms utilize unconditional information for the GALT calculation while synchronous algorithms utilize conditional information.

2.2.1 Asynchronous TM Algorithms

The well-known Chandy/Misra/Bryant null message algorithm [BRY77, CHA79] is a typical asynchronous TM algorithm. It is based on assumptions that a reliable FIFO link exists between each pair of neighboring Logical Processes (in HLA terminology, an LP is a federate) for sending messages in one direction and the TS sequence of messages sent on a link is nondecreasing. An LP waits until each incoming FIFO link contains a message and then continues to process the smallest time stamped message. The TS of the processed message can be seen as the LP's GALT, so it sets its Logical Time (LT) to that TS. After that, a null message can be sent to each neighboring LP with a TS equal to (LT+lookahead) guaranteeing that it will not send any messages with a TS smaller than (LT+lookahead) in the future. The null messages may then asynchronously trigger the GALT calculation of other LPs, which causes new null messages to be sent out. As the guarantee in a null message is true unconditionally, it is called an Unconditional Information report (UI). The HLA TM service group does not restrict a federate to send TSO messages in a nondecreasing TS order and received TSO messages therefore cannot be used to calculate the GALT as in the Chandy/Misra/Bryant null message algorithm.

Instead, each regulating federate sends its UIs periodically and each constrained federate calculates its GALT based on the last UIs received from all regulating federates.

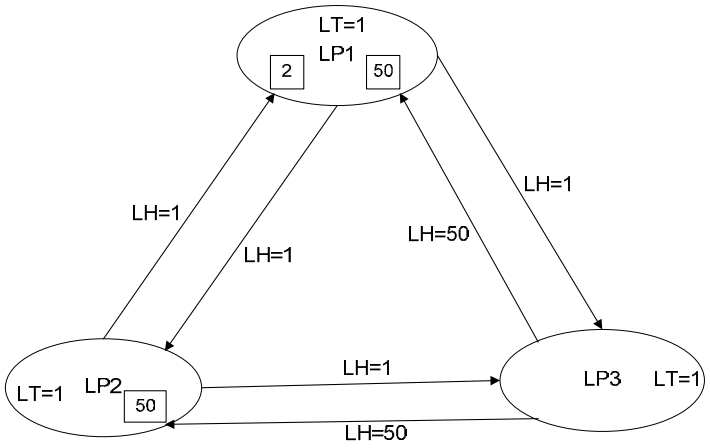


Figure 2.2.1: Time creep situation

The performance of a TM algorithm depends on the lookahead (LH) of the simulation application [FUJ88]. With a small lookahead, a prominent problem of asynchronous TM algorithms occurs, which is known as “time creep” where LPs can only advance their LTs by steps of a lookahead increment before the next non-null message in the simulation application can be processed. An example is given for the Chandy/Misra/Bryant null message algorithm with the scenario shown in Figure 2.2.1. A message is denoted as a rectangle with a time stamp. First of all, LP1 processes its message with TS=2, advances its LT to 2 and sends a null message with TS=(2+1) to both LP2 and LP3. Upon receiving the null message, LP2 advances its LT to 3 and sends a null message with TS=(3+1) to both LP1 and LP3. Upon receiving the null message sent by LP2, LP1 advances its LT to 4 and again sends a null message with TS=(4+1) to both LP2 and LP3. In this way, the next non-null message, which has TS=50, cannot be processed until LP1 and LP2 have advanced their LTs to 50 with increment steps of 1 based on null messages exchanged back and forth between them.

2.2.2 Synchronous TM Algorithms

In a simple synchronous TM algorithm [FUJ00], each LP first reports its LH plus the TS of its next unprocessed message to all other LPs. The GALT of an LP can be calculated by waiting to receive a report from each other LP and then taking the minimum of all these reports including the self-sent one. With the new GALT, an LP proceeds to process all messages with a TS smaller than or equal to GALT before sending another report out. A report is actually a guarantee that the sending LP will not send a message with a TS smaller than the reported value if it does not later receive a message with a smaller TS than the TS of its next unprocessed message. As the guarantee is true conditionally, such a report is called a Conditional Information report (CI). The synchronous algorithm calculates GALT based on the TS of the next unprocessed message in a simulation application, so it does not have the "time creep" problem as in an asynchronous algorithm. To illustrate this using the scenario in Figure 2.2.1, after LP1 has processed the message with TS=2, it sends its CI with TS=(50+1); LP2 sends its CI with TS=(50+1); as LP3 does not have a buffered unprocessed message, it sends its CI with TS equal to infinity. Then the GALT of each federate can be calculated as the minimum of these three CIs, which equals to 51 enabling the two messages with TS=50 to be processed by LP1 and LP2 respectively.

The earliest effort of utilizing conditional information can be traced back to the conditional event algorithm by Chandy and Sherman in 1989 [CHA89]. Other early work includes Lubachevskey's bounded lag algorithm [LUB89], which defines a future simulation time window to reduce synchronization overhead, and Ayani's distance between objects algorithm [AYA89], which exploits the topology information of logical processes for GALT calculation. The YAWNS algorithm in [NIC93] is based on the simple synchronous algorithm described above. Synchronous TM algorithms need to deal with transient messages in a proper way to ensure that they are considered in the GALT calculation. Traditionally, this is solved using message counters [FUJ00].

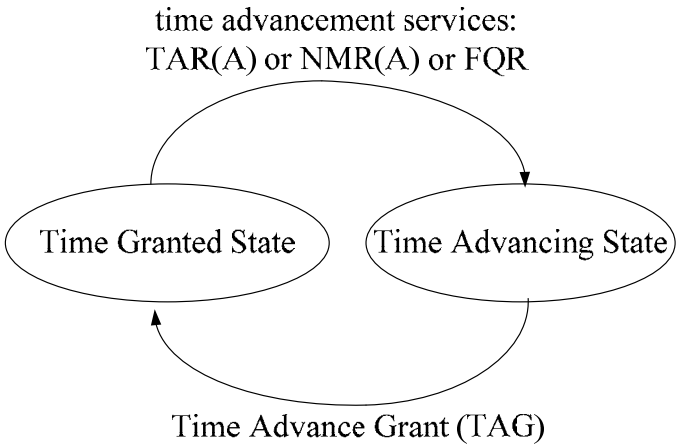
A synchronous TM algorithm requires each LP to send a CI before calculating a new GALT, so its time advancement may be blocked by an LP which sends CIs with a low frequency. This low frequency situation occurs in many real world scenarios as described in Subsection 1.3.2. This is the major drawback of synchronous TM algorithms.

2.2.3 TM in HLA

The Time Management (TM) service group provides services for a federate to coordinate its Logical Time (LT) advancement with other federates in the same federation [FUJ98]. The HLA supports alternative TM approaches, including the conservative approach and the optimistic approach. The conservative approach is a pessimistic way of delivering messages, in which any violation of out-of-order message delivery is completely prevented. This is realized using a synchronous or asynchronous TM algorithm to ensure the correct causally ordered message delivery as discussed in Subsections 2.2.1 and 2.2.2. In contrast, the optimistic approach allows violations to occur. Whenever a violation is detected, it rolls back the state of the relevant federates to previous states. Users can choose the particular approach to be used by invoking the corresponding time advancement service provided by the RTI. Three sets of time advancement services are provided, namely Time Advance Request (TAR) and Time Advance Request Available (TARA), Next Message Request (NMR) and Next Message Request Available (NMRA), and Flush Queue Request. The first two sets, i.e., TAR(A) and NMR(A), are used for the conservative approach while FQR is used for the optimistic approach. For the conservative approach, TAR(A) is generally used for a time-stepped simulation application while NMR(A) is generally used for an event-driven simulation application. For the details of these time advancement services, please refer to the HLA interface specification [IEEE1516.1].

A constrained federate can be in either of two TM states, namely the time advancing state and the time granted state as shown in Figure 2.2.2. When a federate is in the time granted

state and wants to advance its LT, it explicitly makes a request to the underlying RTI and transfers to the time advancing state. Only when the RTI issues a grant callback, the LT of the requesting federate can be advanced to the granted time and the federate returns to the time granted state.



TAR(A): Time Advance Request (Available)
NMR(A); Next Message Request (Avaialble)
FQR: Flush Queue Request

Figure 2.2.2: TM state transition in HLA

2.3 Major DDM Approaches

As discussed in Subsection 1.1.2.4, the DDM computation can be decomposed into four distinct processes, namely declaring, matching, connecting and routing [PET04]. Various DDM approaches exist and their differences can be clarified with this decomposition. The following discussion assumes filtering takes place at the sender side.

2.3.1 Region-Based Approach

The region-based approach is also known as the brute force approach. Each subscription region modification results in sending a region update message in its declaration process. The matching process checks each update region with each subscription region to derive the overlapping information and therefore has $O(N^2)$ computational complexity, which is not scalable with respect to the total number of regions. Since its matching derives exact

overlapping information, its routing process does not involve irrelevant data and therefore is efficient. The region-based approach is used in an early version of DMSO RTI 1.3 [VAN98], and the MAK high performance RTI [WOO02].

2.3.2 Grid-Based Approach

The grid-based approach divides a routing space into a grid of cells. Each region is then mapped to the grid cells. An update region and a subscription region are assumed to overlap with each other if and only if they share at least one common grid cell. DMSO RTI 1.3 NG provides an implementation of this approach in its StaticGridPartitioned strategy [HYE02].

Upon a subscription region modification, the declaration process checks whether the set of cells with which the region overlaps changes or not. Only when there is a change, communication cost is incurred for sending region updates. The matching process simply manages all grid cells with their respective overlapping subscription region list. However, its matching is not exact so that its routing process involves irrelevant data exchange. This incurs not only communication cost but also receiver-side filtering cost. A larger grid cell size causes more irrelevant data to be exchanged and therefore more routing cost [TAN00, RAK96, VAN94]. On the other hand, a smaller grid cell size results in more grid cells, which increases both the declaration cost and the matching cost [TAN00]. Ayani et al. have carried out a study on optimization of the cell size for the grid-based approach [AYA00].

2.3.3 Hybrid Approach

Tan proposed a hybrid approach which utilizes ideas of both the region-based and grid-based approaches [TAN00]. Each subscription region modification results in sending a region update message in its declaration process. The matching process first uses the grid-based approach to map all regions to the grid cells. Then the region-based approach

is used to do exact matching between update regions and subscription regions which overlap the same grid cell. In this way, the matching cost is reduced compared with the pure region-based approach and its routing cost is reduced compared with the pure grid-based approach. However, its matching cost depends on the chosen size of grid cells. The grid-filtered region-based approach proposed by Boukerche et al. [BOU05] is a variation of the hybrid approach.

2.3.4 Sort-Based Approach

The sort-based approach aims at improving the matching performance by sorting the bounds of all regions before checking their overlapping status. Similar to the region-based approach, each subscription region modification results in sending a region update message in the declaration process. The matching is exact so the routing process is efficient.

Raczy proposed a sort-based approach in [RAC05]. The overlapping information is maintained in an N-by-N bit-matrix assuming there are N update regions and N subscription regions. The matrix row denotes the update region and column denotes the subscription region. Each matrix element is initialized to be 1 with the assumption that each update region initially overlaps with each subscription region. Two subscription region sets (SubscriptionSetBefore and SubscriptionSetAfter) are maintained as N-bit vectors. The following procedure of the algorithm is based on one dimension and it is repeated for each dimension. It first inserts bounds of all regions into a list L and sorts L. Next it initializes SubscriptionSetBefore to empty and inserts all subscription regions into SubscriptionSetAfter. Then it scans L from low to high and operates according to the region type (update or subscription region) and bound type (lower or upper bound) of a point.

- If the point is a lower bound of a subscription region R, R is removed from SubscriptionSetAfter.
- If the point is an upper bound of a subscription region R, R is inserted into

SubscriptionSetBefore.

- If the point is a lower bound of an update region R, a conclusion can be made that all regions in SubscriptionSetBefore do not overlap with R, which causes a whole row of the N-by-N bit-matrix of overlapping information to be updated.
- If the point is an upper bound of an update region R, a conclusion can be made that all regions in SubscriptionSetAfter do not overlap with R, which also causes a whole row of the bit-matrix to be updated.

The computational complexity of Raczy's sort-based approach is still $O(N^2)$, but the performance is good due to the utilization of bit operations. It has been shown that Raczy's sort-based algorithm is the best choice in many situations compared with other existing matching algorithms [RAC05].

Though Raczy's sort-based approach is very promising, it still has some drawbacks. The main drawback is that, the algorithm statically processes all the regions in one round and cannot dynamically deal with a selective region modification without processing all the regions once again. Secondly, when a bound of an update region is reached, a conclusion can be made based on one of the two subscription region sets causing a whole row of the N-by-N bit-matrix to be updated, which is a costly operation. Thirdly, the N-by-N bit-matrix of overlapping information requires $N^2/8$ bytes storage, which is not scalable with respect to N.

The sweep and prune algorithm is a well-known graphics algorithm for detecting Axis-Aligned Bounding Box (AABB) intersections [COH95]. It constructs a list for each dimension. The list maintains the coordinate values of all AABBs on the corresponding dimension in an increasing order. When used as a DDM algorithm, for a new simulation time step or frame, as some AABBs may have moved, each list is insertion-sorted. When the insertion sort performs a swap between a lower bound and an upper bound of two different AABBs, the overlapping status of the two AABBs is rechecked. In this way, the AABB intersections for the new time step are detected incrementally based on the

intersections detected for the last time step and its computational complexity is O(n+s), where n is the total number of AABBs and s is the total number of swaps made on any dimension. Based on the assumption of temporal coherence that AABBs do not move large distances between neighboring time steps, s should be small and thus the sweep and prune algorithm performs efficiently. The Lucid multiplayer online game platform provides an HLA DDM implementation based on the sweep and prune algorithm [LIU06]. Liu and Theodoropoulos then extended this DDM implementation to parallel computing architectures for further speed-up [LIU09]. However, the sweep and prune algorithm has the following limitations for its use in DDM matching.

- It scans and re-sorts all lists in each time step or frame. In the case when only a small number of regions are modified in a frame, this is definitely not efficient. A dynamic mechanism for a selective AABB modification is needed.
- DDM matching should only be between update regions and subscription regions. However, the sweep and prune algorithm maintains all types of bounds on a dimension in a single sorted list, so half of the swaps during its insertion sort are wasteful. In Subsection 5.3, we will show how this wasteful processing is prevented in our extended efficient sort-based matching algorithm by using multiple sorted lists per dimension.

2.4 HLA-Based Multi-User Gaming

Nowdays, computer games are a highly profitable business with revenues almost twice that of movies [YVG]. Especially, interactive multi-user Internet games, such as Counter Strike [CS] and World of Warcraft [WOW], have gained the most significant popularity. They are coordinated through either client-server or distributed serverless architectures [BAU01]. The traditional client-server architecture [BUTFLY] provides centralized game coordination at the server site and thus offers more opportunity for ensuring game security such as prevention of players' cheating. However, it creates a processing and communication bottleneck at the server site and thus is not scalable with respect to the number of players. The message relaying at the server site causes a large end-to-end

latency between players. Recently, the distributed serverless game architecture [DIO99, BET01, KNU04, BHA08] has become popular as it is able to solve the above drawbacks of the traditional client-server architecture and increases the scalability and performance. However, it is more complicated to ensure state consistency and game security in the distributed architecture. As the case study described in Chapter 7 is more concerned with optimizing communication performance of games especially in terms of their network capacity requirement and end-to-end latency rather than aspects of game security, the distributed game architecture is chosen for discussions in this thesis.

To accommodate the great demand for reality and interactivity, it is required that status updates are exchanged between players frequently. This requirement is however not achievable because of the large network latency and limited network capacity on the Internet [KNU04, SIN99, BHA08]. Without well-provisioned dedicated game servers, modern fast-paced action games, such as Quake III [QUAKE], limit the number of interacting players to 16-32 due to the high network capacity requirement for exchanging state updates [BHA08].

Due to the reuse and interoperability supported by the HLA and communication optimization supported by the HLA DDM service group, the HLA is promising at supporting multi-user gaming on the Internet.

Faucher has integrated HLA with the Spearhead game, a tank game developed by MAK technologies, for better interoperability between different components of the game architecture [FAU98]. Multi Agent Systems (MAS) have been in existence for many years and its major target applications include computer games and training virtual environments. To promote the reuse and interoperability of different MAS architectures and environments, Kumar et al. proposed a reusable architecture and visualization test bed for distributed agent simulation based on HLA [KUM05]. Lees et al. have developed the HLA_AGENT toolkit by integrating the HLA with the SIM_AGENT toolkit, which

enables the development of HLA-compliant game agents [LEE06]. The Lucid platform is a multi-user online game middleware developed by the Multimedia Innovation Centre of the Hong Kong Polytechnic University [LUCID]. It employs the HLA DDM as its communication filtering mechanism [LIU05, LIU06].

Our group has previously developed a scalable architecture for supporting interactive games on the internet [CAI02b]. Using a maze game as an example, it divides the whole maze into four partitions and assigns a server to each partition. Based on the current position, a player connects to a specific server with a TCP/UDP connection for communication with other players. The communication between servers are through an RTI and DDM is employed to limit the amount of communication between servers. It retains the security advantages of the simple centralized client-server architecture in that cheating among players can be prevented and a consistent state of scoring and accounting can be maintained. In addition, the load of computation and communication are shared amongst multiple servers and thus the architecture is more scalable with respect to the number of players.

All the existing HLA-based gaming architectures use a vendor-specific RTI and thus require prior security setup (port opening) across administrative domains according to the specific RTI used if executed on the Internet. For this reason, users may not be able to join the games as their administrators may not allow such security setup to be performed. As Grid service invocations, which are executed on top of the http protocol and thus can traverse firewalls, are used for communications in SOHR, this means SOHR is very suitable for multi-user gaming as users are able to join a game more conveniently on the Internet. This will be discussed in detail in Chapter 7.

2.5 Summary

This chapter first reviews research in the area of distributed simulation on the Grid. Three approaches are defined for HLA-based distributed simulation on the Grid, namely the

Grid-facilitated approach, Grid-enabled approach and Grid-oriented approach. Related work is reviewed for each of the three approaches. As this project aims at developing a new TM algorithm which overcomes the drawbacks of traditional TM algorithms, a literature review is given on traditional TM algorithms, including synchronous and asynchronous algorithms, with their respective drawbacks. As this project also aims to develop more efficient DDM approaches to improve the communication efficiency in SOHR, major DDM approaches are then reviewed including the Region-Based approach, Grid-Based approach, Hybrid approach and Sort-Based approach. For the case study of SOHR to support multi-user gaming on the Grid, related work on HLA-based multi-user gaming is reviewed.

Chapter 3

SOHR Framework

To achieve the first objective of this project as described in Subsection 1.3.1, a Service Oriented HLA RTI (SOHR) is developed. It is an open source middleware [SOHR] that enables flexible execution of HLA-based distributed simulations on the Grid. This chapter elaborates on the design of SOHR and evaluates its performance in both a LAN and a WAN environment.

3.1 Overview of SOHR Framework

The architecture of our SOHR framework is illustrated in Figure 3.1.1. It contains seven key Grid services, namely the RTI Index Service, the Local Service (LS), the Federation Management Service (FMS), the Declaration Management Service (DMS), the Object and Ownership Management Service (OOMS), the Time Management Service (TMS) and the Data Distribution Management Service (DDMS), all of which are implemented based on GT4. All services except the RTI Index Service follow the WS-Resource factory design pattern as introduced in Subsection 1.1.3.

The RTI Index Service provides a system-level registry so that all other services are able to register their End Point References (EPRs) here and dynamically discover each other. It also provides services to create and destroy federations in the system.

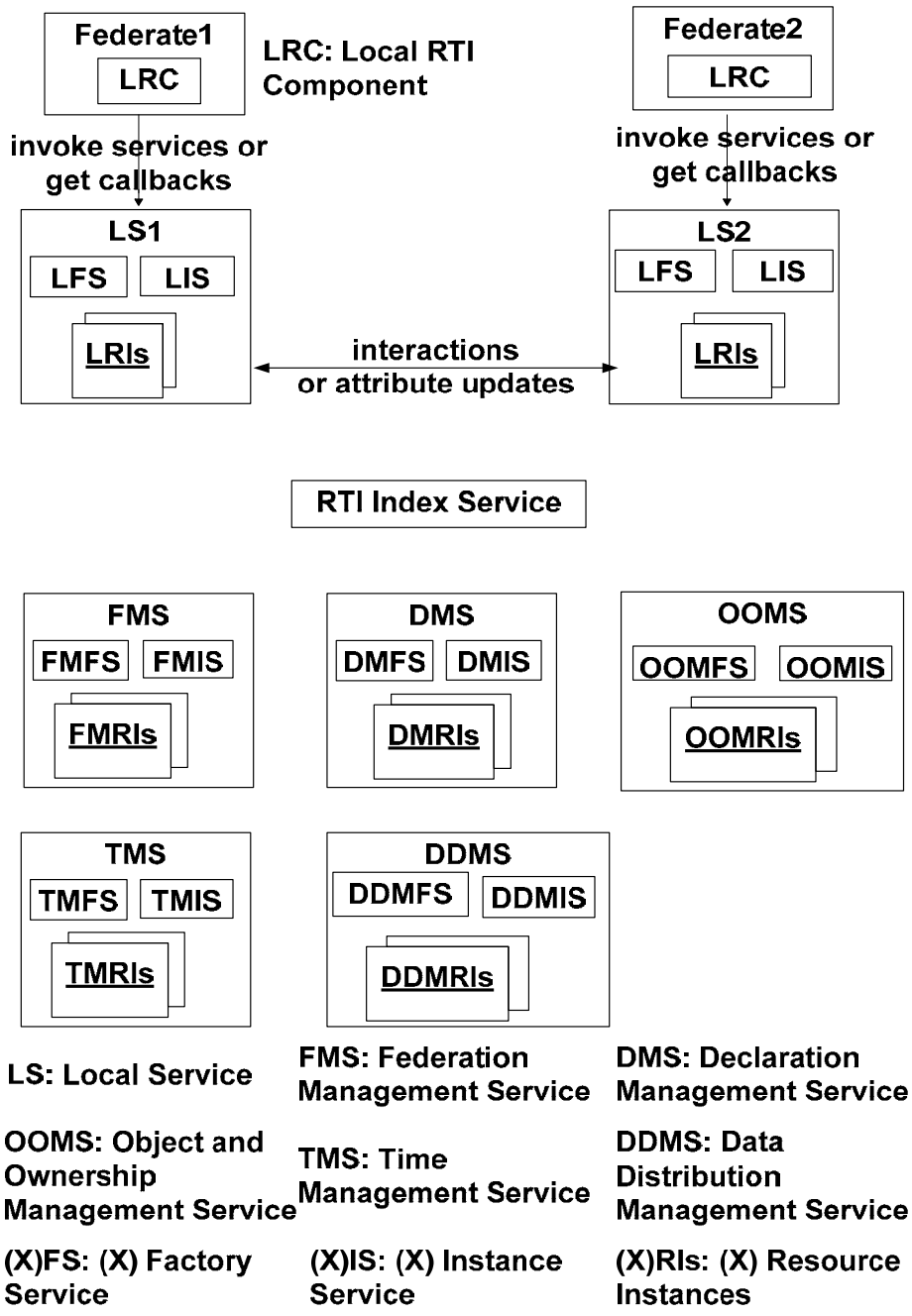


Figure 3.1.1: SOHR architecture overview

The five management services correspond to the six HLA service groups. Each of them consists of a factory service, an instance service and multiple resource instances, with one resource instance for each federation. For example, the FMS provides functionalities of the HLA Federation Management service group as a Grid service and consists of the factory service FMFS, the instance service FMIS and multiple resource instances FMRIs, with one FMRI for each federation. The OOMS corresponds to two service groups, the Object

Management service group and Ownership Management service group. Because attribute ownership information is specified per object instance, it is convenient to keep object instances and their attribute ownership information in a single OOMS instead of separating them into two Grid services.

To offload major RTI-specific workload from federates, a decoupled federate architecture is adopted. The Local Service (LS) is responsible for major RTI-specific processing and is also used as a messaging broker of federates. The LS consists of the factory service LFS, the instance service LIS and multiple resource instances LRIs, with one LRI for each federate. Multiple federates may share the same LS, but each has its own LRI. A federate communicates with the outside world through its LRI by invoking services and getting callbacks. As shown in Figure 3.1.2, the LRI is structured into six modules, namely the Callback Module, FM Module, DM Module, OOM Module, TM Module and DDM Module. Each of the modules except the Callback Module corresponds to one of the management services. The Callback Module is used to buffer callbacks for a federate.

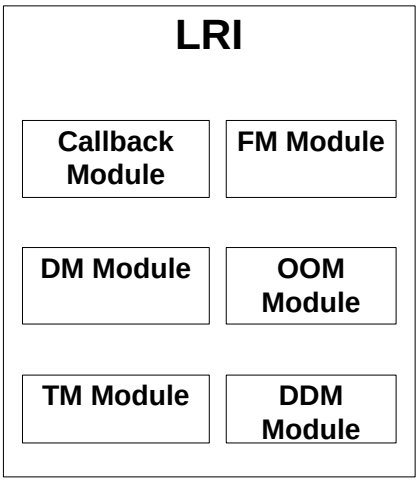


Figure 3.1.2: LRI structure

The objective of separating the HLA service groups into different Grid services and using a modular structure for the LRI is to create a plug-and-play paradigm for an HLA RTI implementation so as to build an extensible SOHR framework. There may be multiple

algorithms for the implementation of an HLA service group, and these can generally be classified as centralized algorithms and distributed algorithms. In a centralized algorithm, the major processing is done by the corresponding management service while the corresponding module in the LRI simply keeps some necessary information related to its federate. In a distributed algorithm, the major processing is done by the corresponding module in the LRI while the corresponding management service simply keeps relevant centralized information of the federation. A particular module in the LRI and its corresponding management service cooperate to provide the services of the HLA service group. Based on a specific algorithm for the HLA service group, a corresponding management service and a particular module in the LRI can be developed. It may be thought that the separation of the HLA services groups into the various Grid services may impact the overall performance of SOHR. However, this is not the case as most of the communications, which are generally attribute updates, interactions, TM synchronization messages and DDM region update messages, are directly conducted between LRIs in a peer-to-peer manner.

The Local RTI Component (LRC) is a federate's local library that implements the HLA service interfaces [IEEE1516.1] and does the translation between HLA service interfaces and the corresponding LIS Grid service invocations. It allows legacy HLA-based federates to work in SOHR. A decoupled design is used between a federate and its LRI so that the LRC passes all requests by the federate to the underlying LRI through the interfaces of the LIS and the LRI buffers callbacks for the federate in its Callback Module. Both HLA 1.3 and HLA 1516 service interfaces are supported in two different LRC libraries. The LRC is light-weight, so a federate can be run on resource-limited platforms such as Personal Digital Assistants (PDAs) and cellphones. Most of the local RTI processing, such as that related to DDM and TM, is offloaded to the LRI in a similar way to offloading it to the network processors in [SAN08], so the federate has more computation time for its real simulation work. Since the callbacks are buffered by the LRI, federate migration is simplified [LI07]. The migration protocol proposed in [CAI05] requires the source

federate to flush all messages pending delivery and insert them in a queue. Then the queue is encoded and transferred to the migration destination where it gets decoded. Additionally, a complex algorithm is used in the protocol to prevent event loss. SOHR does not have these problems when migrating federates. A source federate just transfers the federate code and its status. The migrated federate can then connect to the same LRI to fetch the buffered callbacks.

3.2 Detailed Design of SOHR

In this subsection, the design details of each system component shown in Figure 3.1.1 will be discussed.

3.2.1 Local RTI Component (LRC)

The Local RTI Component (LRC) is a federate side library which implements the HLA service interfaces and makes legacy HLA-based federates work in SOHR. It simply does the translation between the HLA standard interfaces and LIS Grid service invocations, so it is quite light-weight. As mentioned, both HLA 1.3 And HLA 1516 service interfaces are supported in two different LRC libraries. When referring to a specific HLA service interface, its HLA 1.3 version is used for description in this chapter.

The major component in LRC is the RTIambassador class which implements various HLA service interfaces such as createFederationExecution and tick. The other components are supporting classes and exception classes such as the SuppliedParameters class used in the sendInteraction service and the InteractionParameterNotKnown class which defines a type of exception thrown when an interaction parameter is not valid. The RTIambassador class translates HLA service calls to corresponding Grid service invocations of the LIS. For example, an RTIambassador.createFederationExecution request by the local federate will be translated to a createFederationExecutionOut Grid service invocation of the LIS which will further contact the RTI Index Service to create a new federation. A pulling mechanism

is used for callbacks to the local federate. When the local federate requests for callbacks by calling the RTIambassador.tick (or Evoke Multiple Callbacks in HLA 1516) function, the LRC pulls callbacks from the LRI by invoking the getCallbacks service of the LIS. If any callback is received, the LRC delivers it to the local federate by calling the corresponding callback function provided by the local federate.

3.2.2 Local Service (LS)

The Local Service (LS) operates as a messaging broker for federates. On initialization, the RTIambassador instance in each federate calls LFS's createResource function to create a new LRI for communication purposes. A federate's LRC communicates with the outside world through its corresponding LRI and the LIS defines miscellaneous Grid service interfaces for accessing the LRI, such as an interaction-sending operation invoked by other federates' LRIs to exchange an interaction and an unsubscription-notification operation invoked by the DMS to notify an unsubscription made by another federate.

As introduced in Subsection 3.1, the LRI has six modules and each of them except the Callback Module corresponds to one of the management services. The FM Module communicates with the FMS to provide HLA Federation Management services to the federate. The DM Module communicates with the DMS to provide HLA Declaration Management services to the federate. It keeps subscription and publication information of the LRI's corresponding federate as well as relevant subscription and publication information of other federates. This subscription and publication information ensures correct message exchanges between federates' LRIs. The OOM Module communicates with the OOMS to provide HLA Object and Ownership Management services to the federate. It is responsible for directly exchanging messages with other federates' LRIs with the help of the DM and DDM Module. The TM Module communicates with the TMS to provide HLA Time Management services to the federate. The DDM Module communicates with the DDMS to provide HLA Data Distribution Management services to the federate. It keeps the relevant overlapping information of regions to ensure correct

message exchanges between federates' LRIs.

The Callback Module maintains two callback queues, namely the TSO Queue and non-TSO Queue. The TSO Queue keeps time-stamp-order data messages while the non-TSO queue keeps receive-order data messages as well as system callbacks such as timeRegulationEnabled and timeAdvanceGrant. Other modules insert callbacks in the corresponding queues of the Callback Module. The Callback Module works closely with the TM Module to ensure the correct delivery sequence of TSO data messages to the federate.

Based on different algorithms for the HLA service group, multiple corresponding management services with their respective particular modules can be implemented and deployed in SOHR. Through a federate-side configuration file, the user of SOHR is able to choose the most suitable algorithm to optimize the performance based on a specific federation scenario. An LRI with an appropriate version of the module will then be generated. This plug-and-play paradigm will be demonstrated by the implementation of multiple TM algorithms discussed in Chapter 4 and the implementation of multiple DDM algorithms discussed in Chapter 6.

Originally, we were considering whether each of the modules could be implemented as a separate Grid service. Realizing that it will make the framework performance too low because of the close communications between modules, we decided to bundle all modules in the LS and implement each module as a separate Java class. An LIS Grid service invocation is translated to a method invocation of the corresponding module class which will take over the processing work. For example, a getCallbacks Grid service invocation of the LIS is translated to a CallbackModule.getCallbacks Java method invocation which will scan the callback queues and return the available callbacks as a list. To create a fully service-oriented architecture in the future, each of the modules can be implemented as a separate Grid service. However, the performance will suffer because of the close

communication between modules. The details will be discussed in Chapter 8 when discussing the future work of this project.

3.2.3 Federation Management Service (FMS)

The FMS is a Grid service related to the HLA Federation Management service group. The FMFS defines a createResource operation invoked by the RTI Index Service to create an FMRI for a new federation. The FMIS defines some operations corresponding to the HLA Federation Management service group such as joinFederationExecution and resignFederationExecution. It also defines operations for fetching some information kept in FMRIs. When a federation is created, a new FMRI will be created and it is the FMRI's responsibility to create a new DMRI, OOMRI, TMRI and DDMRI for the new federation. The FMRI operates as a federation-wide registry. It keeps the End Point References (EPRs) of the DMRI, OOMRI, TMRI and DDMRI created for the federation. It also keeps the EPRs of federates' LRIs for the federation. It generates federate handles for new joining federates. When a federate resigns from the federation, the FMS is responsible to notify the whole federation about this resign for deletion of any information related to the federate.

3.2.4 Declaration Management Service (DMS)

The DMS is a Grid service related to the HLA Declaration Management service group. The DMFS defines a createResource operation invoked by the FMRI to create a DMRI for a new federation. The DMIS defines some operations corresponding to the HLA Declaration Management service group such as publishInteractionClass and subscribeInteractionClass. It also defines operations for fetching some information kept in DMRIs. The DMRI keeps the subscription and publication information of all federates in the federation. When a DMRI is created and initialized, it reads the FED (Federation Execution Data in HLA 1.3) or FDD (FOM Document Data in HLA 1516) file and generates handles for the interaction classes, interaction class parameters, object classes

and object class attributes defined in the federation. All these handles are kept in the DMRI and the DMIS defines operations for fetching the handle information. Since these FOM (Federation Object Model) [IEEE1516] data are centralized in the DMRI, the DMIS can define some operations to enable dynamic change of the FOM which is not supported in most of the traditional RTIs. For example, the DMIS can define an addObjectClass operation for adding a new object class in the federation. Similar to the HLA Evolved modular FOMs approach [MOL07], the DMIS can define some operations to enable a FOM module to be dynamically added to or deleted from the federation FOM.

3.2.5 Object and Ownership Management Service (OOMS)

The OOMS is a combined Grid service related to the HLA Object Management and Ownership Management service groups. The OOMFS defines a createResource operation invoked by the FMRI to create an OOMRI for a new federation. The OOMIS defines some operations corresponding to the HLA Object Management and Ownership Management service groups such as registerObjectInstance and attributeOwnershipAcquisition. It also defines operations for fetching some information kept in OOMRIs. The OOMRI keeps all object instances in the federation and their handles. It also keeps the attribute ownership information for each object instance. Since the attribute ownership information is specified per object instance, it is convenient to include both the HLA Object Management and Ownership Management service groups in the OOMS.

3.2.6 Time Management Service (TMS)

The TMS is a Grid service related to the HLA Time Management service group. The TMFS defines a createResource operation invoked by the FMRI to create a TMRI for a new federation. The TMIS defines some operations corresponding to the HLA Time Management service group. The TMRI keeps relevant information for Time Management in the federation. The operations defined by the TMIS and data kept in the TMRI depend on the TM algorithm used in the SOHR framework.

If a centralized algorithm is used, the TMS does the major processing of Time Management, such as the recalculation of LBTS in HLA 1.3 or GALT in HLA 1516 for each constrained federate based on time information of regulating federates. Since time information is frequently exchanged, the TMS may be easily overloaded with increasing federation size. So a distributed TM algorithm is more attractive for SOHR.

A distributed algorithm, such as the one in the RTI Version F.0 [CAR97], can be easily adapted to our SOHR framework. The TMIS defines operations for changing of a federate's regulating or constrained status such as enableTimeRegulating and disableTimeConstrained. The TMRI keeps the regulating and constrained status of all federates in the federation. When a federate enables/disables its constrained status, the TMRI notifies all regulating federates' LRIs to update their TM Modules. In this way, a list of constrained federates in the TM Module of each regulating federate's LRI is always kept updated. With the always updated list of constrained federates, a regulating federate’s LRI directly sends its time information to each of the constrained federates’ LRIs. The TM Module of a constrained federate’s LRI may recalculate the constrained federate’s LBTS (or GALT in HLA 1516) when time information of a regulating federate is received. In this way, the major processing of Time Management is done by the TM Module of each federate's LRI, which makes SOHR more scalable.

A new TM algorithm can be incorporated in SOHR by adding a TM Module of the LRI and a TMS based on the new algorithm, which makes SOHR an extensible framework. This will be demonstrated by the implementation of multiple distributed TM algorithms in SOHR in Chapter 4.

3.2.7 Data Distribution Management Service (DDMS)

The DDMS is a Grid service related to the HLA Data Distribution Management service group. The DDMFS defines a createResource operation invoked by the FMRI to create a

DDMRI for a new federation. The DDMIS defines some operations corresponding to the HLA Data Distribution Management service group. The DDMRI keeps relevant information for Data Distribution Management in the federation. Similar to the case of TMS, the operations defined by the DDMIS and data kept in the DDMRI depend on the DDM algorithm used in the SOHR framework. Basically a DDM algorithm can be classified as a centralized algorithm or a distributed algorithm. In a centralized DDM algorithm, a DDM coordinator calculates the overlaps between regions (matching). In a distributed DDM algorithm, the matching is done at each federate's local site.

If a centralized DDM algorithm is utilized in SOHR, the DDMRI acts as the DDM coordinator for its federation. It keeps information of all regions and calculates their overlaps. It sends the updated overlapping information to relevant federates' LRIs to update their DDM Modules. The DDMIS defines relevant operations for updating region information kept in the DDMRI, such as a createRegion operation for creating a new region and an associateRegionForUpdates operation for associating attributes of an object instance with a region.

If a distributed DDM algorithm is utilized in SOHR, the matching is done by the DDM Module of each LRI while the DDMRI may simply keep some centralized information for Data Distribution Management, such as all regions and their respective creators.

There are many DDM matching algorithms as introduced in Subsection 2.3. Each algorithm has different performance, but there is no algorithm that fits all federation scenarios [RAC02]. Multiple combinations of a DDM Module of the LRI and a DDMS can be incorporated in SOHR based on different algorithms so that the combination with the best performance can be used for a specific federation scenario. This will be demonstrated by the distributed implementations of multiple DDM approaches in SOHR in Chapter 6.

3.2.8 RTI Index Service

The RTI Index Service provides a system-level registry so that all other services are able to register their End Point References (EPRs) here and dynamically discover each other. To use SOHR, a federate only needs to know the EPR of the RTI Index Service. With this knowledge, a federate is able to request the RTI Index Service to return the EPR of any registered service. Since there may be multiple instances of the same service type registered (e.g., multiple DMS), the RTI Index Service provides a load balancing mechanism to improve the overall system efficiency. It keeps the load information of each registered service in terms of the number of resource instances created, and returns the least loaded service instance of a particular type when requested by a federate. For example, it keeps the load of each DMS in terms of the number of DMRIs created, and returns the least loaded DMS when a federate asks for a DMS. If a federate already knows the EPR of a deployed service (e.g., LS), it can specify the EPR of the service in a local configuration file and directly access the service without querying the RTI Index Service.

The RTI Index Service also defines operations for creation and destruction of federations and keeps the mapping between each federation and its FMRI EPR.

3.3 Experiments and Results

We have implemented a SOHR prototype [SOHR]. It implements most HLA service groups including FM, DM, Object Management, TM and DDM. The Ownership Management service group is not yet implemented.

3.3.1 Ping-Pong Experiment Design

To compare the performance of SOHR and the performance of the DMSO RTI 1.3NG [DMSO], a ping-pong experiment was designed. A federate sends an interaction to another federate and then waits to receive an interaction from that federate before

repeating the same process. The average round trip time is measured and the one way latency is calculated as half of the round trip time.

3.3.2 Ping-Pong Experiment Trace on SOHR

The trace of the initialization stage of the ping-pong experiment on SOHR is shown in Figure 3.3.1. The communications between a federate and its LRC are through HLA service calls and callbacks, while all the other communications are through Grid service invocations. Before the actual simulation execution, the RTI Index Service, FMSs, DMSs, OOMSs, TMSs, DDMSs and LSs have to be deployed and started on respective hosts. The phases of the initialization trace are as follows:

1. Each FMFS, DMFS, OOMFS, TMFS, DDMFS or LFS registers its EPR with the RTI Index Service (Step 0). When LRC1 (Federate1's RTIambassador instance) is created, it asks the RTI Index Service to return the LFS with the lowest load (LFS1), creates and initializes a new LRI (LRI1) using the returned LFS (Step 1, 2, 3). Since a new LRI is created with LFS1, LRC1 updates the load information of LFS1 in the RTI Index Service (Step 4).

2. The creation of a new federation starts with Federate1's createFederationExecution request (Step 5). LRC1 translates this request to a createFederationExecutionOut service invocation of LIS1 (Step 6) which next forwards the invocation to the RTI Index Service (Step 7). After the RTI Index Service receives the request, it chooses the FMFS with the lowest load, creates and initializes an FMRI for the new federation (Step 8, 9). The initialization of the FMRI creates and initializes a DMRI (Step 10, 11, 12, 13), an OOMRI, a TMRI and a DDMRI for the new federation.

3. After the federation is created, Federate1 calls a joinFederationExecution request (Step 14) which is translated to a joinFederationExecutionOut service invocation of LIS1 by LRC1 (Step 15). The LIS1 asks the RTI Index Service for the FMRI EPR of the federation (Step 16) and invokes a joinFederationExecution service invocation of the corresponding FMIS (Step 17). The DMRI EPR of the federation is fetched and stored in LRI1's DM Module (Step 18). Similarly, the OOMRI EPR, TMRI EPR and DDMRI EPR of the federation are fetched and respectively stored in LRI1's OOM Module, TM Module and DDM Module.

4. Federate1 requests to subscribe to interactions of Class2 (Step 19) and publish interactions of Class1 (Step 22). Each of the two requests is forwarded to LIS1 by LRC1 (Step 20, 23) and further forwarded to DMIS (Step 21, 24). The getSuperClassesOfInteractionClass service invocation of DMIS (Step 25) is to deal with an interaction class hierarchy defined in the FOM. Since no interaction class hierarchy exists in this ping-pong experiment, it simply returns an empty list.

5. After Federate1 finishes its initialization, Federate2 joins the created federation, subscribes to interactions of Class1 and publishes interactions of Class2 in a similar way.

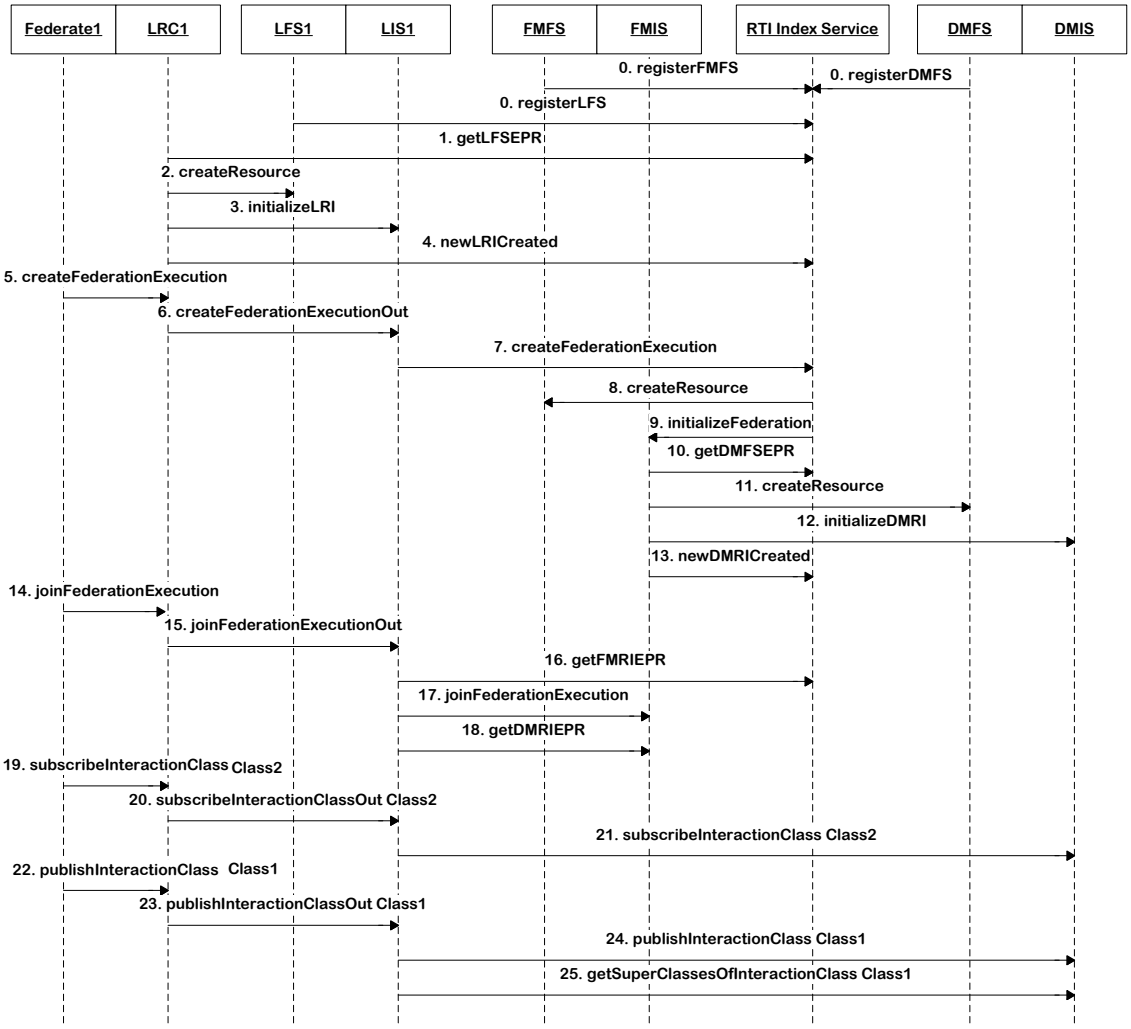


Figure 3.3.1: Initialization trace of the ping-pong experiment on SOHR

After the initialization stage, interactions are exchanged between Federate1 and Federate2 as shown in Figure 3.3.2.

1. Federate1 keeps calling RTIambassador.tick until an interaction is received from Federate2 (Step 29). Federate2 requests to send an interaction of Class2 (Step 26) and LRC2 translates this request to a sendInteractionOut service invocation of LIS2 (Step 27). The LRI2's DM Module keeps the list of subscribers and their LRI EPRs. Because Federate1 is subscribing to interactions of Class2 (Step 19), it should be included in the list. This causes a sendInteraction service invocation of LIS1 by LIS2 (Step 28). Then LRC1 receives this interaction after invoking

getCallbacks (Step 30) when Federate1 calls an RTIambassador.tick (Step 29), and delivers this interaction to Federate1 (Step 31).

2. After Federate1 receives the interaction, it requests to send an interaction of Class1 and this interaction is received by Federate2 in the reverse way (Step 32, 33, 34, 35, 36, 37).
3. The above two phases form one round of the ping-pong style communication. After multiple iterations, the two federates resign from the federation and the federation is destroyed by Federate2. After the whole simulation execution, the FMRI, DMRI, OOMRI, TMRI, DDMRI and LRIs created for the federation have all been destroyed.

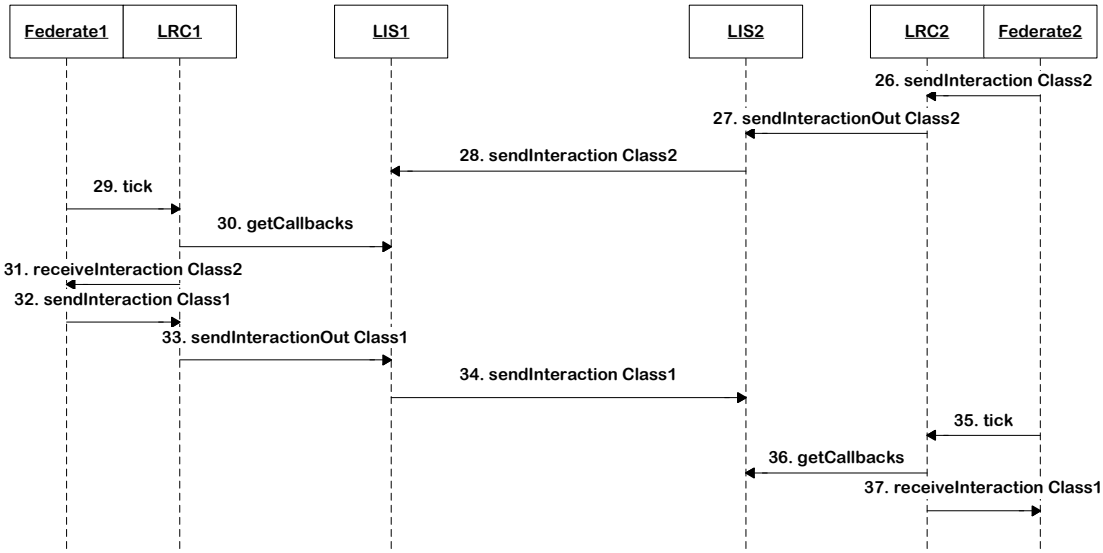


Figure 3.3.2: Interaction trace of the ping-pong experiment on SOHR

3.3.3 Experimental Configurations and Results

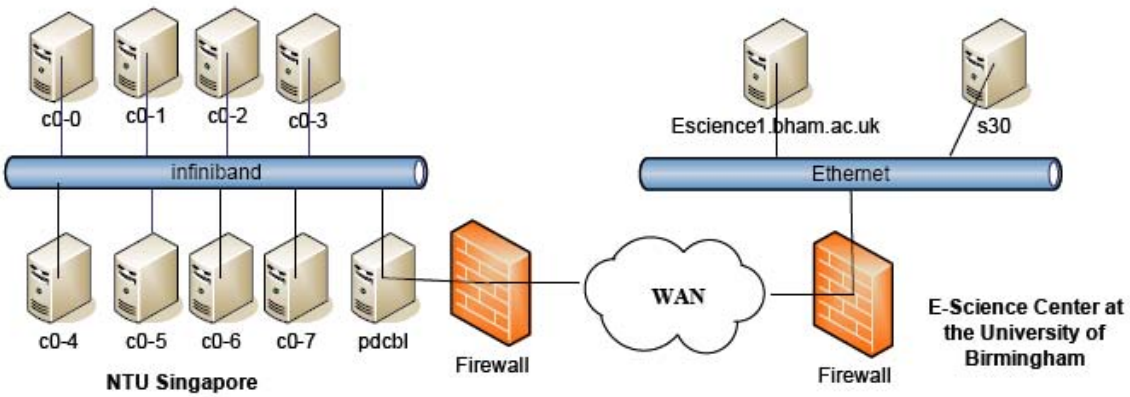


Figure 3.3.3: Experimental testbed

Test	Federate1	Federate2	rtiexec + fedex	Result
LAN	c0-6	c0-7	c0-0	5.3
WAN	pdcbl	Escience1	pdcbl	116.5

DMSO RTI

****

Test	Federate1	LS1	Federate2	LS2	GT4.0.2	GT4.1.1
LAN	c0-6	c0-4	c0-7	c0-5	85.6	19.6
WAN A	Escience1	pdcbl	c0-6	c0-4	1026.8	241.6
WAN B	c0-6	pdcbl	s30	Escience1	916.7	154.7

SOHR

Figure 3.3.4: Experimental configurations and latency results in milliseconds

Our experimental testbed consists of a cluster in Nanyang Technological University (NTU), Singapore, and two hosts in the E-Science Center at the University of Birmingham, UK, as shown in Figure 3.3.3. The cluster has a 10Gb/s infiniBand connection. Each node of the cluster is installed with two dual core Xeon 3.0GHz CPUs, 4GB RAM and Redhat Enterprise Linux 4 OS. The two hosts in the E-Science Center are connected by an Ethernet and each of them is installed with two Xeon 3GHz CPUs, 2GB RAM and Redhat Enterprise Linux AS 3 OS. The cluster has only one externally accessible node which is its main node (pdcbl) and our experiments were configured based on this restriction. The performance of the DMSO RTI and our SOHR framework

was measured both inside the cluster (LAN) and across continents (WAN). The RTI Index Service and all management services were deployed in the cluster for all experimental configurations of SOHR. The experimental configurations and latency results are shown in Figure 3.3.4. For SOHR, the columns GT4.0.2 and GT4.1.1 respectively show the SOHR performance on GT4.0.2 and GT4.1.1.

The experimental results show that SOHR's performance on GT4.1.1 both on the LAN and on the WAN is much better than its performance on GT4.0.2. This is due to the support of persistent TCP/IP connections by GT4.1.1. Without persistent TCP/IP connection support as in GT4.0.2, each service invocation creates a new TCP/IP connection for communication and this has a significant overhead of connection setup and tearing down per service invocation. For a detailed analysis of this overhead, please refer to [CHE08]. With persistent TCP/IP connection support as in GT4.1.1, multiple service invocations between the same client and server share the same TCP/IP connection for communication, so the overhead of the connection setup and tearing down is amortized among the multiple service invocations. Since SOHR's performance on GT4.1.1 is much better, the performance results on GT4.1.1 are used for comparison with the DMSO RTI in the following paragraphs.

On the LAN, SOHR's latency is 19.6 milliseconds which is 3.7 times that of the DMSO RTI. SOHR's latency is larger than that of the DMSO RTI due to the overheads of Grid service invocations. Basically Grid service invocations have two kinds of overheads, processing cost and communication cost. As discussed in [XIE05], the size of a SOAP message is around 1-2 kilobytes per Grid service request/response which is much larger than a DMSO RTI message. This increases both processing and communication costs in SOHR. To further analyze this, we carried out a test of a simple add(int a) service invocation on the LAN using GT4.1.1 and found that service invocations by clients executed on the same machine as the service and also on a different machine take almost the same time of 8 milliseconds inside the cluster. This means that the major overhead of

the LAN experiment is the processing cost while the communication cost is not high due to the fast intra-cluster network connection.

Our SOHR was tested on the WAN using two configurations, with two LSs deployed at the NTU side (WAN A) and one LS deployed at each side (WAN B) respectively. WAN B has better performance, because the interval between the time an interaction reaches an LRI (Step 28 in Figure 3.3.2) and the time the interaction is fetched by the receiving federate (Step 30 in Figure 3.3.2) depends on the service invocation sequence of the LIS. If the receiving federate tries to fetch the interaction (Step 30 in Figure 3.3.2) before the interaction reaches the LRI (Step 28 in Figure 3.3.2), this incurs extra time and the effect is more prominent if a remote LS is used as in WAN A. To check this hypothesis, sleeping time can be used by the receiving federate to delay the interaction fetching (Step 30 in Figure 3.3.2) until the interaction has reached the LRI (Step 28 in Figure 3.3.2). This generates an appropriate service invocation sequence of the LIS so that WAN A may have the same performance as WAN B. WAN B's latency is 154.7 milliseconds, which is around 1.3 times that of the DMSO RTI. The relative performance of SOHR to the DMSO RTI on the WAN is improved compared with the performance on the LAN. The communication cost becomes the dominant overhead on the WAN because of the long distance.

3.4 Major Benefits of SOHR

The major benefits of the SOHR framework are summarized as follows:

- The functionalities of an RTI are provided as Grid services so that distributed simulations can be conducted without any vendor-specific RTI software.
- Grid services, which can traverse firewalls, are used as the communication infrastructure so that distributed simulations can be conveniently conducted across administrative domains on a WAN.
- The six HLA service groups are mapped to different modules in the LRI structure and different management services so that multiple combinations of a particular

module of the LRI and its corresponding management service can be implemented based on different algorithms for an HLA service group and deployed in SOHR. This plug-and-play paradigm makes SOHR an extensible framework. A user is able to choose the most suitable algorithm based on a specific federation scenario to optimize the performance. New algorithms can be implemented and studied on SOHR, which makes SOHR a good experimental environment for distributed simulation research.

- The decoupled design between a federate and its LRI makes the LRC very light-weight so that a federate can be run on resource-limited platforms such as PDAs and cellphones. Both HLA 1.3 and HLA 1516 service interfaces are supported in two different LRC libraries. As the major local RTI processing is offloaded to the LRI, the federate has more computation time for its real simulation work. The LRI buffers callbacks for its federate, so federate migration is simplified.
- Handles of interaction classes, object classes, parameters and attributes are centralized in the DMRI and Grid service interfaces can be provided by the DMIS to enable dynamic configuration of the FOM which is not supported by most of the traditional RTIs.
- The various Grid services can be deployed and undeployed on demand. The RTI Index Service operates as a system-wide registry so that the other services can dynamically discover each other. The RTI Index Service also employs a load balancing mechanism to improve the overall system resource efficiency.
- Other Grid services can easily be integrated into SOHR for more functionalities such as check-pointing services for fault tolerance purposes and messaging services for optimization of cross-cluster communications.
- Based on the Grid, SOHR can utilize the underlying Grid infrastructure, such as the Grid Security Infrastructure (GSI), to evolve to a secure, scalable and coordinated large scale distributed simulation environment.

3.5 Summary

This chapter proposes a Service Oriented HLA RTI (SOHR) framework to support distributed simulations in a heterogeneous Grid environment. SOHR has many benefits as summarized in Section 3.4. A prototype of SOHR has been implemented with a subset of the HLA specification and the experiment results show that, SOHR's performance on GT4.1.1 is much better than its performance on GT4.0.2 due to the persistent TCP/IP connection support by GT4.1.1. The SOHR's performance on GT4.1.1 is comparable with the performance of the DMSO RTI especially on the WAN.

Chapter

4

A Hybrid Time Management Algorithm Based on Both Conditional and Unconditional Information

The HLA TM service group ensures a TSO message delivery sequence and correct time advancement of each federate in a federation. Traditional TM algorithms can be either synchronous or asynchronous. However, each traditional algorithm has its own drawbacks. To resolve their drawbacks, the second objective of this project is the development of a hybrid TM algorithm as introduced in Subsection 1.3.2. It is based on both conditional and unconditional information, combining traditional synchronous and asynchronous algorithms together. This chapter describes the asynchronous, the synchronous and the new hybrid algorithms in detail and compares their performance in both a LAN and a WAN environment. Then the hybrid algorithm is compared with the TM algorithm implemented in the Federate Simulations Development Kit [FDK].

4.1 A Classification of Regulating Federates

Regulating federates can be classified into two groups based on their respective regulating and constrained status, and types of time advancement services used as shown in Table 4.1.1. Federates from Group UIOnly only provide UIs for the time management while federates from Group UICI can provide both UIs and CIs. Depending on the algorithm, either UIs, CIs, or both UIs and CIs can be used for time advancement. The classification considers only conservative time advancement services, including Time Advance Request

(TAR), Time Advance Request Available (TARA), Next Message Request (NMR) and Next Message Request Available (NMRA). The optimistic time advancement service Flush Queue Request (FQR) can be added in a similar way to NMRA [LI08].

Group UIOnly includes regulating federates using TAR or TARA, and regulating and unconstrained (R&UC) federates using NMR or NMRA. According to the HLA interface specification [IEEE1516.1], a federate may not send any TSO messages with a TS smaller than t+LH after a TAR(t) or TARA(t) is invoked. This can be directly translated to a UI to be sent out to constrained federates as implemented in the RTI Version F.0 [CAR97], so regulating federates using TAR or TARA send UIs only. When an R&UC federate invokes an NMR(t) or NMRA(t), a Time Advance Grant (TAG) with parameter t can be immediately delivered back and the federate's LT is advanced to t. According to the HLA interface specification [IEEE1516.1], the federate may not send TSO messages with a TS smaller than t+LH afterwards. This can be directly translated to a UI to be sent out to constrained federates, so R&UC federates using NMR or NMRA also send UIs only.

Group UICI includes regulating and constrained (R&C) federates using NMR or NMRA. According to the TM algorithm used, such a federate may send UIs or CIs or both UIs and CIs. Subsection 4.2 discusses this in detail in the description of our three TM algorithms. For simplicity, we only consider federates from Group UICI when discussing our experiments in Subsection 4.3 and comparing the proposed hybrid TM algorithm with FDK's TM algorithm in Subsection 4.4.

Table 4.1.1: A classification of regulating federates

R and C Status	Time Advancement Service	Group
R&UC	TAR or TARA	UIOnly
R&C	TAR or TARA	UIOnly
R&UC	NMR or NMRA	UIOnly
R&C	NMR or NMRA	UICI

4.2 Algorithm Description

In the following description of our algorithms, only federates from Group UICI are considered. Several additional notations are used in addition to the ones which have already been introduced. “head” denotes the TS of the first message in the TSO queue. “EPSILON” denotes an infinitesimal positive value.

4.2.1 Asynchronous Algorithm

```
void NMR(t)
{
1.      if(inTimeAdvancingState==true)
2.              throw InTimeAdvancingStateException;
3.      requestedTime=t;
        // EPSILON is added when LH=0
4.      UI=min(requestedTime,head,GALT)+LH(+EPSILON);
5.      if (UI>lastUISent)
        {
6.              send UI to constrained federates by calling their
                receiveUI functions;
7.              lastUISent=UI;
        }
8.      if (GALT>min(requestedTime,head))
9.              deliver TAG(min(requestedTime,head));
10.     else
11.             inTimeAdvancingState=true;
}

void receiveUI(sendingFederateHandle, newUI)
{
12.     updateGALT by taking the minimum of the last UIs
        received from all other federates;
13.     if(inTimeAdvancingState==false)
14.             return;
        // EPSILON is added when NMR is used and LH=0
15.     UI=min(requestedTime,head,GALT)+LH(+EPSILON);
16.     if (UI>lastUISent)
        {
17.             send UI to constrained federates by calling their
                receiveUI functions;
18.             lastUISent=UI;
        }
        // > for NMR; >= for NMRA
19.     if (GALT >(=) min(requestedTime,head))
        {
20.             deliver TAG(min(requestedTime,head));
21.             inTimeAdvancingState=false;
        }
}
```

Figure 4.2.1: Asynchronous algorithm

Our asynchronous algorithm uses UIs only and its pseudo code is shown in Figure 4.2.1. As the code for NMRA is similar to NMR, it is not shown. When an NMR or NMRA is invoked, a UI is sent out if a larger value is available (Line 4-7). If NMR is used and the federate has a lookahead of 0, EPSILON is added in the UI to prevent a deadlock situation (Line 4). The receiveUI function is invoked when a federate receives a UI from another federate. The receiving federate may update its GALT with the new UI (Line 12). This may trigger a new UI to be sent out by the receiving federate if it is in time advancing state (Line 15-18). Also, a TAG may be delivered to the receiving federate (Line 19-21). Similar to traditional asynchronous TM algorithms, our asynchronous algorithm has the "time creep" problem when LH is small, in which situation the GALT value is increased slowly in steps of the LH value before the next message can be processed.

4.2.2 Synchronous Algorithm

Our synchronous algorithm uses CIs only as shown in Figure 4.2.2. One of its major differences with the asynchronous algorithm is that a sequence number is added to each CI and a new GALT can only be calculated by taking the minimum of all CIs with the same sequence number including the CI sent by the local federate. When an NMR or NMRA is invoked, GALT is checked to determine whether a TAG can be delivered or not (Line 4-6). If not, GALT has to be advanced before a TAG can be delivered. The local federate enters a new phase of GALT calculation by sending out a new CI with an increased sequence number (Line 10, 11). If the local federate has received CIs with the same sequence number from all other federates, its GALT can be updated by taking the minimum of all CIs including the self-sent one (Line 15). Since GALT has been updated, it is checked again whether a TAG can be delivered (Line 16-19). If a TAG cannot be delivered, the federate has to enter another phase of GALT calculation (Line 20-22). The receiveCI function is invoked when a federate receives a CI from another federate. If all CIs are ready, a new GALT can be calculated (Line 25). If the federate is in time advancing state, this may trigger a TAG to be delivered (Line 28-31). If a TAG cannot be delivered, the federate has to enter a new phase of GALT calculation (Line 32-36).

```
void NMR(t)
{
1.      if(inTimeAdvancingState==true)
2.              throw InTimeAdvancingStateException;
3.      requestedTime=t;
4.      if (GALT>min(requestedTime,head))
        {
5.              deliver TAG(min(requestedTime,head));
6.              return;
        }
7.      inTimeAdvancingState=true;
8.      CI=min(requestedTime,head)+LH(+EPSILON);
9.      CI=min(CI,minTSOfTransientMessages);
10.     CISeqNum++;
11.     send CI-CISeqNum to constrained federates by calling
        their receiveCI functions;
12.     reset minTSOfTransientMessages;
13.     if (CI-CISeqNum from a federate is not ready)
14.             return;
15.     update GALT by taking the minimum of all CI-CISeqNum
        including the self-sent one;
16.     if (GALT>min(requestedTime,head))
        {
17.             deliver TAG(min(requestedTime,head));
18.             inTimeAdvancingState=false;
19.             return;
        }
20.     CI=min(requestedTime,head)+LH(+EPSILON);
21.     CISeqNum++;
22.     send CI-CISeqNum to constrained federates by calling
        their receiveCI functions;
}

void receiveCI(sendingFederateHandle, seqNumOfNewCI, newCI)
{
23.     if (CI-seqNumOfNewCI from a federate is not ready)
24.             return;
25.     update GALT by taking the minimum of all
        CI-seqNumOfNewCI including the self-sent one;
26.     if(inTimeAdvancingState==false)
27.             return;
28.     if (GALT >(=) min(requestedTime,head))
        {
29.             deliver TAG(min(requestedTime,head));
30.             inTimeAdvancingState=false;
31.             return;
        }
32.     CI=min(requestedTime,head)+LH(+EPSILON);
33.     CI=min(CI,minTSOfTransientMessages);
34.     CISeqNum++;
35.     send CI-CISeqNum to constrained federates by calling
        their receiveCI functions;
36.     reset minTSOfTransientMessages;
}
```

Figure 4.2.2: Synchronous algorithm

To solve the transient message problem, we use an acknowledgement-based solution similar to Samadi's Global Virtual Time (GVT) algorithm [SAM85]. The variable minTSOfTransientMessages remembers the minimum time stamp of transient messages sent by the local federate, which is considered for the local CI calculation (Line 9, 33).

Similar to other synchronous algorithms, the major drawback of our synchronous algorithm is that its time advancement may be blocked by a federate which sends CIs with a low frequency.

4.2.3 Hybrid Algorithm

To benefit from the advantages of both the asynchronous and synchronous algorithms in order to resolve their drawbacks, a hybrid algorithm is developed by combining the asynchronous and synchronous algorithms together. The pseudo code is shown in Figure 4.2.3. In our hybrid algorithm, a federate sends both UIs and CIs and maintains UIGALT and CIGALT. UIGALT is calculated using UIs in the same way as in the asynchronous algorithm. CIGALT is calculated using CIs in the same way as in the synchronous algorithm. To utilize both UIs and CIs, GALT is calculated as max(CIGALT,UIGALT). When a federate is in time advancing state, the advancement of either its CIGALT or UIGALT may result in a TAG delivery. Suppose that, a federate F1 invokes NMR and sends its CI-n out (Line 14-18) entering time advancing state (Line 11), but this cannot trigger the CIGALT calculation because the CI-n from another federate is not ready yet (Line 19, 20). Then it may happen that the update of F1's UIGALT (Line 55, 56) results in a TAG delivery to F1 (Line 64) and makes F1 return to time granted state (Line 65). After that, F1 invokes NMR again and reenters time advancing state (Line 11). To prevent F1 from sending its CI-(n+1) out (Line 14-18) before its CIGALT calculation based on CI-n, a variable seqNumOfLastCIGALTCalculated is utilized to remember the sequence number of CIGALT calculation (Line 22, 38). Before F1 is able to send the new CI out (Line 14-18), the condition CISeqNum=seqNumOfLastCIGALTCalculated is checked (Line 12, 13).

```
void NMR(t)
{
1.      if(inTimeAdvancingState==true)
2.              throw InTimeAdvancingStateException;
3.      requestedTime=t;
4.      UI=min(requestedTime,head,GALT)+LH(+EPSILON);
5.      if(UI>lastUISent)
        {
6.              send UI to constrained federates by calling their
                receiveUI functions;
7.              lastUISent=UI;
        }
8.      if (GALT>min(requestedTime,head))
        {
9.              deliver TAG(min(requestedTime,head));
10.             return;
        }
11.     inTimeAdvancingState=true;
12.     if(CISeqNum>seqNumOfLastCIGALTCalculated)
13.             return;
14.     CI=min(requestedTime,head)+LH(+EPSILON);
15.     CI=min(CI,minTSOfTransientMessages);
16.     CISeqNum++;
17.     send CI-CISeqNum to constrained federates by calling
        their receiveCI functions;
18.     reset minTSOfTransientMessages;
19.     if(CI-CISeqNum from a federate is not ready)
20.             return;
21.     update CIGALT by taking the minimum of all
        CI-CISeqNum including the self-sent one;
22.     seqNumOfLastCIGALTCalculated=CISeqNum;
23.     GALT=max(CIGALT,UIGALT);
24.     UI=min(requestedTime,head,GALT)+LH(+EPSILON);
25.     if(UI>lastUISent)
        {
26.             send UI to constrained federates by calling their
                receiveUI functions;
27.             lastUISent=UI;
        }
28.     if (GALT>min(requestedTime,head))
        {
29.             deliver TAG(min(requestedTime,head));
30.             inTimeAdvancingState=false;
31.             return;
        }
32.     CI=min(requestedTime,head)+LH(+EPSILON);
33.     CISeqNum++;
34.     send CI-CISeqNum to constrained federates by calling
        their receiveCI functions;
}
```

```
void receiveCI(sendingFederateHandle, seqNumOfNewCI, newCI)
{
35.     if(CI-seqNumOfNewCI from a federate is not ready)
36.             return;
37.     update CIGALT by taking the minimum of all
        CI-seqNumOfNewCI including the self-sent one;
38.     seqNumOfLastCIGALTCalculated=seqNumOfNewCI;
39.     GALT=max(CIGALT,UIGALT);
40.     if(inTimeAdvancingState==false)
41.             return;
42.     UI=min(requestedTime,head,GALT)+LH(+EPSILON);
43.     if(UI>lastUISent)
        {
44.             send UI to constrained federates by calling their
                receiveUI functions;
45.             lastUISent=UI;
        }
46.     if (GALT >(=) min(requestedTime,head))
        {
47.             deliver TAG(min(requestedTime,head));
48.             inTimeAdvancingState=false;
49.             return;
        }
50.     CI=min(requestedTime,head)+LH(+EPSILON);
51.     CI=min(CI,minTSOfTransientMessages);
52.     CISeqNum++;
53.     send CI-CISeqNum to constrained federates by calling
        their receiveCI functions;
54.     reset minTSOfTransientMessages;
}
void receiveUI(sendingFederateHandle, newUI)
{
55.     update UIGALT by taking the minimum of the last UIs
        received from all other federates;
56.     GALT=max(CIGALT,UIGALT);
57.     if(inTimeAdvancingState==false)
58.             return;
59.     UI=min(requestedTime,head,GALT)+LH(+EPSILON);
60.     if (UI>lastUISent)
        {
61.             send UI to constrained federates by calling
                their receiveUI functions;
62.             lastUISent=UI;
        }
63.     if (GALT >(=) min(requestedTime,head))
        {
64.             deliver TAG(min(requestedTime,head));
65.             inTimeAdvancingState=false;
        }
}
```

Figure 4.2.3: Hybrid algorithm

With a small lookahead, the asynchronous algorithm has the “time creep” problem, while the hybrid algorithm may not if CIs are sent in time and can directly advance GALT to the next unprocessed message. In this case, the hybrid algorithm has similar performance to the synchronous algorithm and performs better than the asynchronous algorithm.

A low time-resolution federate can be defined as a federate which has a large lookahead and advances simulation time with large increment steps. With a low time-resolution federate which sends CIs with a low frequency due to various reasons such as complex

computation or long latency of CI transmitting, the time advancement of the synchronous algorithm is blocked. In contrast, the hybrid algorithm can use UIs to advance GALT. In this case, the hybrid algorithm has similar performance to the asynchronous algorithm and performs better than the synchronous algorithm.

The hybrid algorithm is always able to use the better information (CIs or UIs) to advance time, so it is a good choice for all federation scenarios.

4.2.4 Extension to Group UIOnly

Our TM algorithms have only considered federates from Group UICI so far. For completeness, we have extended our algorithms to Group UIOnly. The major steps we have taken are as follows:

- GALT consists of two parts now, namely GALTOfUICI and GALTOfUIOnly. Group UICI federates determine GALTOfUICI using any of our three algorithms. GALTOfUIOnly is determined by taking the minimum of the last UIs sent by all Group UIOnly federates. Since the advancement of GALT should be constrained by all regulating federates from the two groups, it is calculated as min(GALTOfUICI,GALTOfUIOnly).
- When a federate from Group UIOnly requests time advancement with parameter t, a UI is immediately sent out with parameter t+LH(+EPSILON).
- When a federate from Group UICI sends a CI out in the synchronous or hybrid algorithm, it should consider possible messages which may be received from Group UIOnly federates in the future. So the value of the CI should be changed to min(requestedTime,head,GALTOfUIOnly)+LH (+EPSILON). This change should be made to Line 8, 20, 32 of Figure 4.2.2 and Line 14, 32, 50 of Figure 4.2.3.

4.3 Experiments and Results

The three TM algorithms are incorporated into the SOHR framework by implementing a different TM Module for each algorithm. The TMS is the same for each algorithm as they

are all implemented in a distributed manner. The plug-and-play paradigm of SOHR enables us to conveniently choose different TM algorithms to be used in different scenarios.

As Grid service invocations are used for communications and the return value of a Grid service invocation can be used as an acknowledgement, our acknowledgement-based solution to the transient message problem does not incur additional message exchange. To compare the performance of our three algorithms, three experiments were carried out both on a LAN and on a WAN with the configuration shown in Figure 4.3.1.

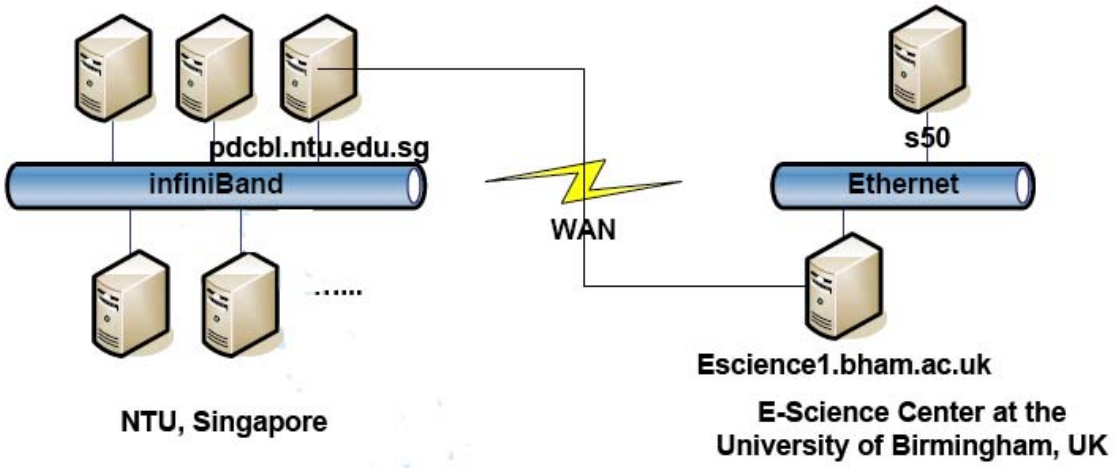


Figure 4.3.1: Experiment configuration

The experiments on the LAN were carried out in a cluster with a 10Gb/s infiniBand connection at Nanyang Technological University (NTU), Singapore. Each node of the cluster is installed with two dual core Xeon 3.0GHz CPUs, a 4GB RAM and a Redhat Enterprise Linux 4 OS. Each of the Grid services in SOHR and each federate process was run on a separate node in the cluster.

The experiments on the WAN were carried out between the cluster at NTU and the E-Science Center at the University of Birmingham, UK. Two machines with an Ethernet connection were used at the Birmingham site. One LS is deployed on the public node pdcbl.ntu.edu.sg at the NTU site and another LS is deployed on the public node

Escience1.bham.ac.uk at the Birmingham site. All other SOHR Grid services were deployed on the public node pdcbl.ntu.edu.sg at the NTU site. Federates were run on other nodes as federates do not need to have Grid services deployed. Federates at the NTU site used the LS deployed at the NTU site while federates at the Birmingham site used the LS deployed at the Birmingham site.

4.3.1 Simulation Execution Performance with respect to Lookahead

An experiment was designed to test the performance of our three TM algorithms with respect to lookahead. Two federates were used, each of which has the same lookahead and requests its time advancement using NMR with time increment of 10. After receiving a TAG, a federate invokes another NMR. The federation was executed for a large number of iterations without any data message communication between the two federates and the average execution time per iteration was measured for each federate. Since the two federates are symmetric, their performance is the same.

On the LAN, the two federates were run on two separate nodes in the cluster. Figure 4.3.2 shows the performance results. It can be seen that the performance of the asynchronous algorithm improves significantly with an increasing lookahead while the performance of both synchronous and hybrid algorithms does not change much. With a small lookahead, the asynchronous algorithm performs much worse than the synchronous and hybrid algorithms due to its “time creep” problem. With a large enough lookahead, the performance of the three algorithms is very close to one another. As the “time creep” problem is a well-known problem in an asynchronous TM algorithm, as discussed in Subsection 2.2.1, the above conclusion is also true for a federation with more than two federates.

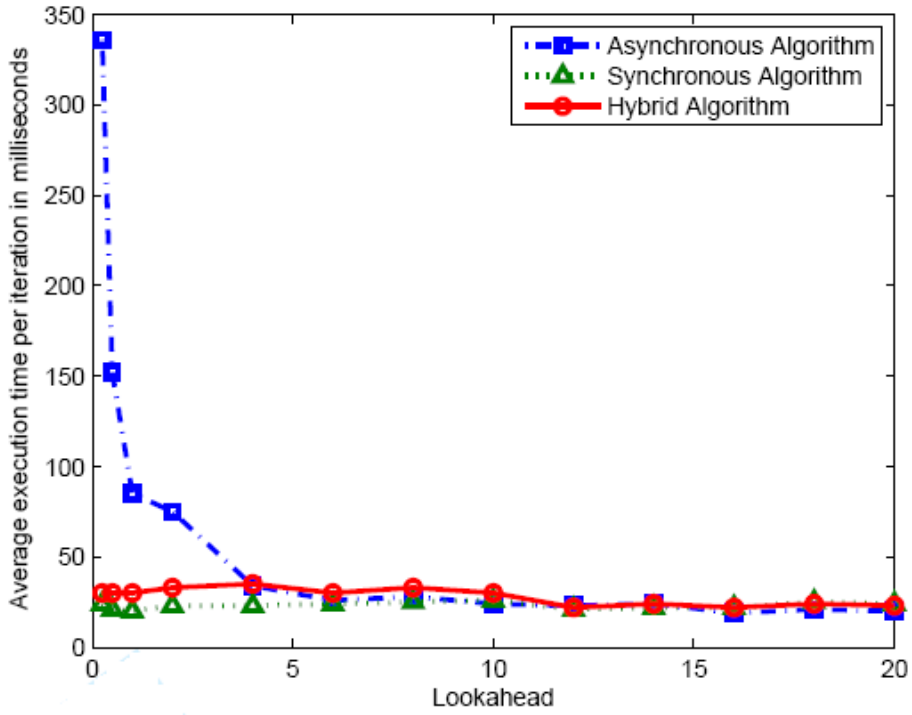


Figure 4.3.2: Simulation execution performance with respect to lookahead on the LAN

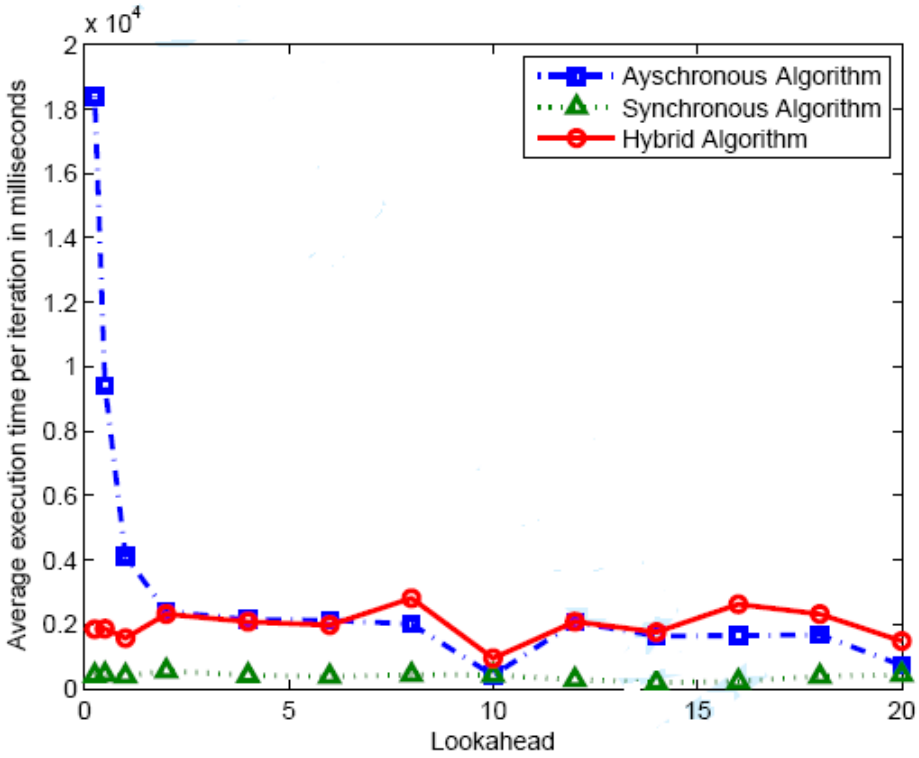


Figure 4.3.3: Simulation execution performance with respect to lookahead on the WAN

On the WAN, one federate and its LS were run at the NTU site and the other federate and its LS were run at the Birmingham site. Figure 4.3.3 shows the performance results. Similar to the results on the LAN, the “time creep” problem of the asynchronous algorithm is also observed on the WAN. One major difference compared to the LAN is that, even with a large enough lookahead, the asynchronous and hybrid algorithms may still perform worse than the synchronous algorithm. This is because, exchanges of UIs may trigger new UIs to be sent out by the receiving federates. In general, this means there are more UI exchanges in the asynchronous or hybrid algorithm than CI exchanges in the synchronous algorithm. On the LAN, the performance of the asynchronous and hybrid algorithms is not affected much by this due to the high speed LAN connection.

4.3.2 Simulation Execution Performance with respect to Processing Time of a Low Time-Resolution Federate

Another experiment was designed to test the performance of our algorithms with respect to the processing time of a low time-resolution federate. Two federates were used. F1 is a high time-resolution federate which has a lookahead of 14 (note: 14 is chosen because it is a value in the middle of the range for lookahead where the “time creep” problem does not occur in the asynchronous algorithm, as shown in Figures 4.3.2 and 4.3.3). It requests time advancement using NMR with time increment of 10 as in the previous experiment. In each iteration of time advancement, it sleeps for a fixed period of time to simulate a fixed processing time. F2 is a low time-resolution federate which has a lookahead of 140 and requests time advancement using NMR with time increment of 100. F2 also sleeps for a fixed period of time in each iteration. The sleeping period of F2 was varied for different runs, each of which was executed for a large number of iterations without any data message communication between the two federates. The average execution time per iteration of F1 and F2 with respect to the processing time of F2 is measured.

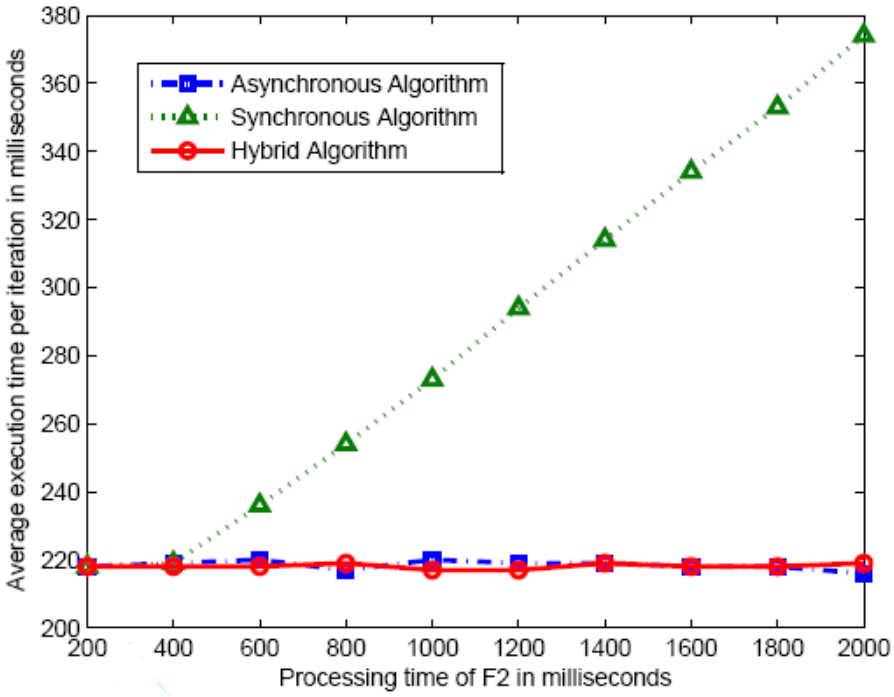


Figure 4.3.4: Simulation execution performance of F1 with respect to processing time of F2 on the LAN

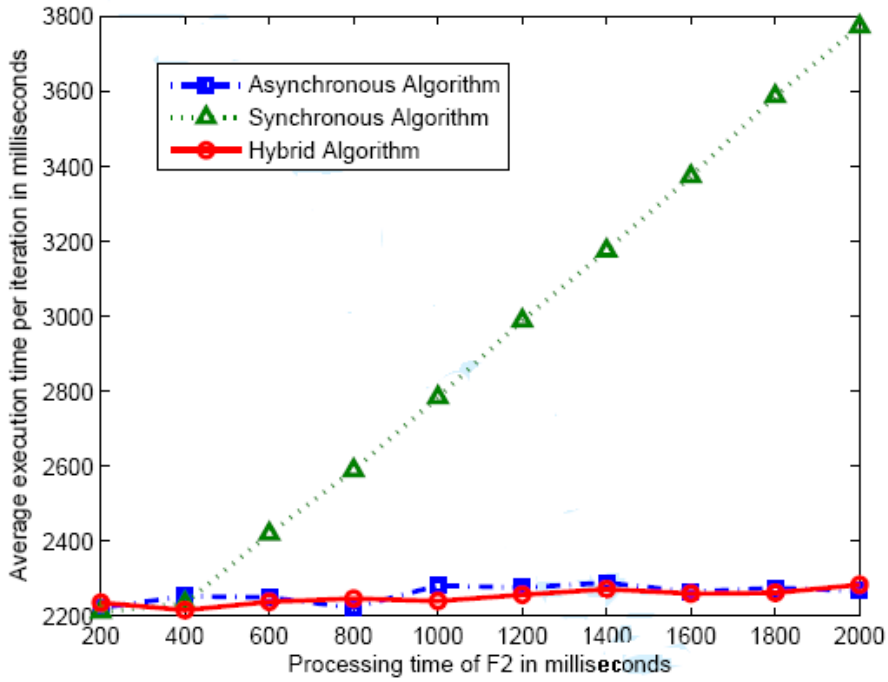


Figure 4.3.5: Simulation execution performance of F2 with respect to processing time of F2 on the LAN

On the LAN, the two federates were run on two separate nodes in the cluster. The sleeping time of F1 was chosen to be 200 milliseconds. The sleeping time of F2 was increased in milliseconds from 200 to 2000 in increment steps of 200 for different runs. The performance results of F1 and F2 are shown in Figure 4.3.4 and Figure 4.3.5 respectively. Figures 4.3.4 and 4.3.5 are similar except the scales of their vertical axes are different. This is due to the fact that each execution iteration of F1 corresponds to a simulation time increase of 10 while each execution iteration of F2 corresponds to a simulation time increase of 100. When the processing time of F2 is small, our three algorithms have similar performance. When the processing time of F2 becomes larger, the frequency of time advancement requests by F2 becomes lower. The frequency of CIs sent by F2 thus also becomes lower, so the performance of the synchronous algorithm becomes worse. Both the asynchronous and hybrid algorithms can use UIs to advance GALT, so their performance does not change much with respect to the processing time of F2.

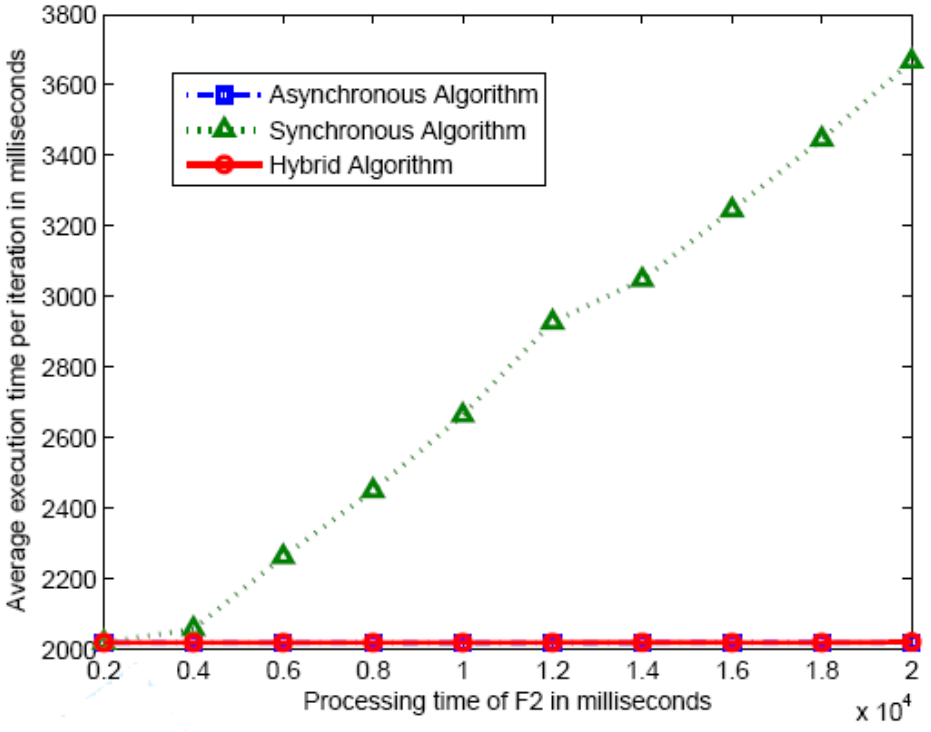


Figure 4.3.6: Smulation execution performance of F1 with respect to processing time of F2 on the WAN

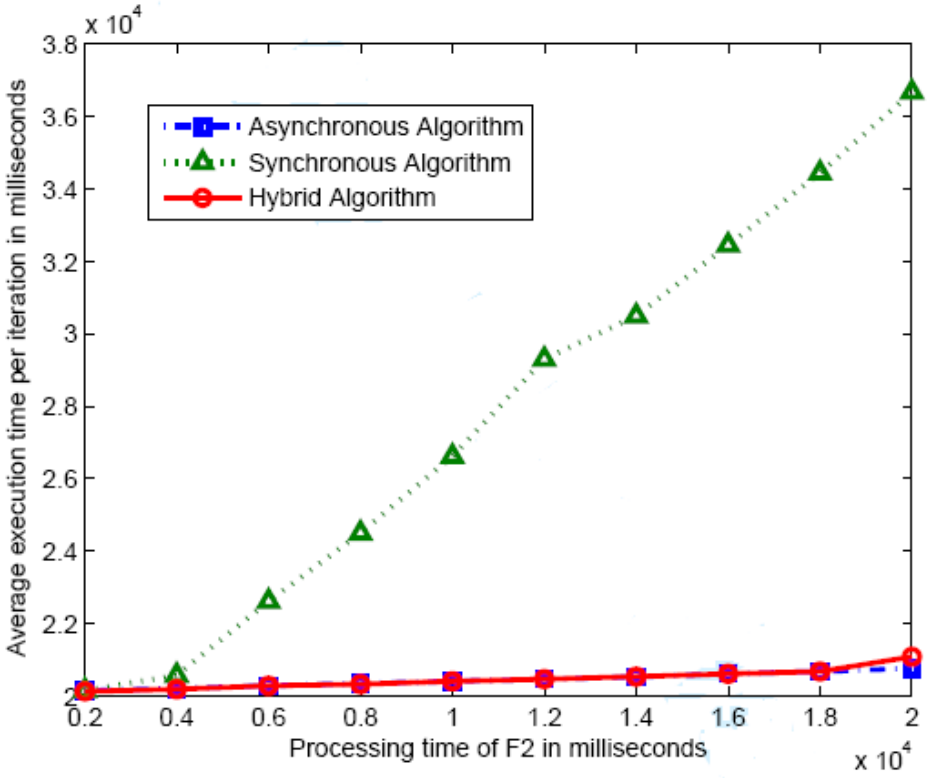


Figure 4.3.7: Simulation execution performance of F2 with respect to processing time of F2 on the WAN

On the WAN, Federate1 and its LS were run at the NTU site while Federate2 and its LS were run at the Birmingham site. To compensate for the large latency on the WAN, the sleeping period of each federate was increased correspondingly. The sleeping time of F1 was increased to 2000. The sleeping time of F2 was varied in milliseconds from 2000 to 20000 in increment steps of 2000 for different runs. The performance results of F1 and F2 are shown in Figure 4.3.6 and Figure 4.3.7 respectively. Figures 4.3.6 and 4.3.7 are similar except the scales of their vertical axes are different. The asynchronous and hybrid algorithms have similar performance and their performance curves closely overlap with each other. Similar to the performance results on the LAN, the drawback of the synchronous algorithm is also observed on the WAN.

4.3.3 Simulation Execution Performance with respect to a Variable-processing-time Federate

The third experiment was designed with three federates. F1 and F2 are two high time-resolution federates both of which have a lookahead of 1 and request time advancement using NMR with time increment of 10. They sleep for a fixed period of time in each iteration. F3 is a low time-resolution federate which has a lookahead of 140 and requests time advancement using NMR with time increment of 100. Instead of sleeping for a fixed period of time in each iteration, it sleeps for a random period simulating variable processing time in different iterations. This is because processing time depends on the temporary machine load and thus is normally uncertain, especially in a Grid environment where computing resources are shared among dynamic collections of individuals and institutions [FOS01b]. The federation was executed for a large number of iterations without any data message communication between federates. The average execution time per iteration of each federate was measured for all the three algorithms.

On the LAN, all the three federates were run on separate nodes in the cluster. The sleeping time for both Federate1 and Federate2 was 200 milliseconds. The sleeping time in milliseconds for F3 was generated using a random number generator with a uniform distribution between 1000 and 2000 in each iteration and the same seed was used for all the three algorithms for a fair comparison. The federation was executed for a large number of iterations. The average execution time per iteration in milliseconds of each federate was measured for all the three algorithms and the results are shown in Table 4.3.1. It can be seen that the hybrid algorithm outperforms the other two algorithms in this experiment. When small sleeping time is generated for F3, the asynchronous algorithm has the “time creep” problem due to the small lookaheads of F1 and F2 while the synchronous and hybrid algorithms can advance time faster using CIs. When large sleeping time is generated for F3, the synchronous algorithm is not efficient as its time advancement is blocked by F3 not sending its CI promptly, while the asynchronous and hybrid algorithms

can advance time faster using UIs even though “time-creep” occurs. The hybrid algorithm is always able to choose the better information (CIs or UIs) to advance time under different circumstances, so it outperforms the other two algorithms.

Table 4.3.1: Simulation execution performance in milliseconds with a variable-processing-time federate on the LAN

TM Algorithm	F1	F2	F3
Asynchronous Algorithm	388	388	3965
Synchronous Algorithm	361	362	3668
Hybrid Algorithm	291	291	3004

On the WAN, Federate1 and Federate3 and their LS were run at the NTU site while Federate2 and its LS were run at the Birmingham site. Similar to the experiment in Subsection 4.3.2, the sleeping period of each federate was increased to compensate for the large latency on the WAN. The sleeping time for both Federate1 and Federate2 was increased to 2000 milliseconds. The sleeping time in milliseconds for F3 was generated using a random number generator with a uniform distribution between 10000 and 20000 in each iteration and the same seed was used for all the three algorithms for a fair comparison. The performance results are shown in Table 4.3.2. Due to the same reasons as in the LAN experiment, the hybrid algorithm outperforms the other two algorithms in the WAN experiment.

Table 4.3.2: Simulation execution performance in milliseconds with a variable-processing-time federate on the WAN

TM Algorithm	F1	F2	F3
Asynchronous Algorithm	3881	3881	39231
Synchronous Algorithm	3567	3564	35447
Hybrid Algorithm	3299	3299	33225

4.4 Comparison between the Hybrid TM Algorithm and FDK's TM Algorithm

So far, we have discussed the drawbacks of both synchronous and asynchronous algorithms and provided a solution in our hybrid algorithm that combines synchronous and asynchronous algorithms together. All the algorithms previously discussed are based on a “pushing” style of time information exchange in that a federate sends its time information when new information is available. To solve the drawbacks of traditional synchronous and asynchronous algorithms, another solution is to use a “pulling” style of time information exchange. To the best of our knowledge, the Federated Simulations Development Kit (FDK) is the only system which uses this alternative solution. So in this section, we perform an empirical comparison between our hybrid TM algorithm and FDK's TM algorithm.

FDK is a software system developed at Georgia Tech [FUJ00b, FDK]. It contains a major component, RTI-Kit, which is a collection of composable libraries for developing RTIs. Its “library-of-libraries” approach to RTI development enhances the modularity and maintainability of the RTI software. Among the libraries of the RTI-Kit, TM-Kit is the one that provides a distributed implementation of a TM algorithm, which we name as FDK's TM algorithm.

With FDK's TM algorithm, when a federate requests its time advancement, there are two cases. If the time advancement request can be granted immediately, a TAG is delivered to the federate. Otherwise, the federate enters time advancing state, and calls for a new round of GALT calculation by sending its time information out and notifying the other federates of the new GALT calculation. When another federate receives the notification, it sends its own time information out. If a federate is in time granted state, the time information sent is equal to LT+lookahead (UI); if a federate is in time advancing state, the time information sent is equal to min(requestedTime, head) + LH (CI). All federates get the same GALT by

taking the minimum of all the times. The transient message problem is solved using message counters [FUJ00b]. To optimize performance, simultaneous GALT calculations initiated by different federates are merged, and different communication patterns can be used for exchanging time information between federates, e.g. “all-to-all”, “star” and “butterfly”.

The FDK's TM algorithm is similar to the hybrid algorithm in that both UIs and CIs are utilized for calculating GALT. However, UIs and CIs are used separately for calculating UIGALT and CIGALT respectively in the hybrid algorithm, while both UIs and CIs contribute to the single GALT calculation in FDK's TM algorithm. Another major difference is that, the hybrid algorithm uses a “pushing” style of time information exchange while FDK's TM algorithm uses a “pulling” style. The “pulling” style of FDK's TM algorithm enables a federate in time advancing state to process its next safe message immediately after the GALT calculation has completed. (A safe message is one that can be processed immediately with a guarantee that no causality violation will occur.) It therefore does not have the “time creep” problem which may still occur in the hybrid algorithm when some CI is blocked and UIs advance GALT slowly due to small lookaheads of some federates. This will be illustrated in Subsection 4.4.1. On the other hand, the “pushing” style of the hybrid algorithm has its own advantage that it exchanges time information only when new information is available, so it is generally more efficient. With FDK's TM algorithm, even though a federate does not have its time status updated, it still sends its time information out upon receiving the notification of a new round of GALT calculation. This may cause many redundant GALT calculations in the federation and thus make FDK's TM algorithm less efficient, which will be illustrated in Subsection 4.4.2.

To illustrate the above differences between the hybrid algorithm and FDK's TM algorithm, we have implemented FDK's TM algorithm in SOHR in a similar way to implementing the other TM algorithms. The following two experiments were carried out both on a LAN and on a WAN with the same configuration as that shown in Figure 4.3.1.

4.4.1 Performance Comparison with respect to Lookahead

The “time creep” problem may still occur in the hybrid algorithm, as shown in Subsection 4.3.3. On the other hand, FDK's TM algorithm does not have the “time creep” problem. To illustrate this, the experiment in Subsection 4.3.3 was repeated for the hybrid algorithm and FDK's TM algorithm, with the lookahead of Federate1 and Federate2 increased from 0.25 to 10. The performance results on the LAN and on the WAN are respectively shown in Figure 4.4.1 and Figure 4.4.2. Both the performance results on the LAN and on the WAN show that the performance of the hybrid TM algorithm improves significantly with an increasing lookahead value. However, the performance of FDK's TM algorithm does not vary much as there is no data exchange between federates in this federation. When the lookahead of Federate1 and Federate2 is small, “time creep” occurs between Federate1 and Federate2 in the hybrid TM algorithm, so its performance is much worse than that of FDK's TM algorithm. When the lookahead is large, the performance of the two algorithms becomes close to each other.

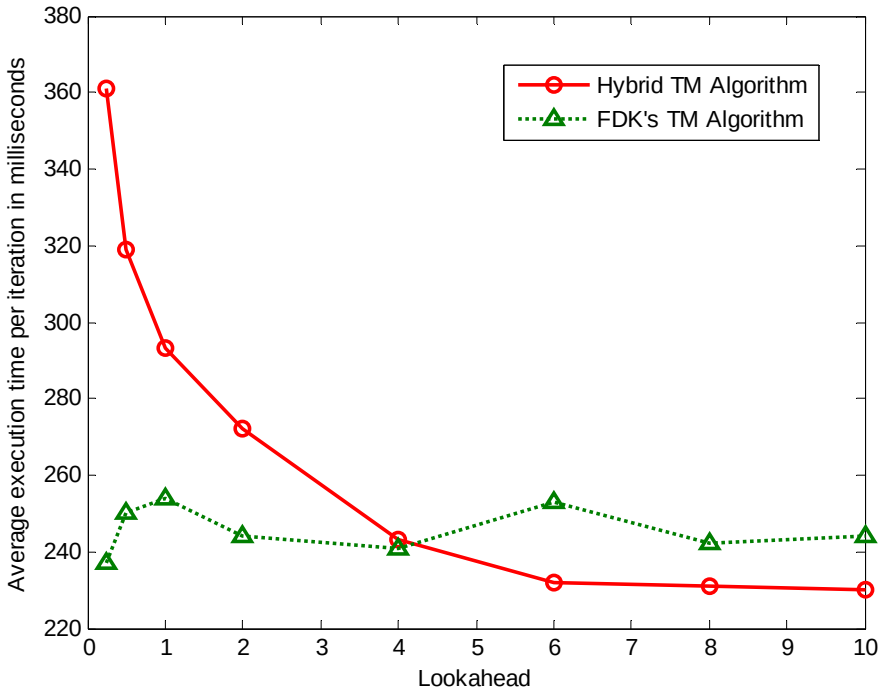


Figure 4.4.1: Simulation execution performance of F1 with respect to lookahead on the LAN

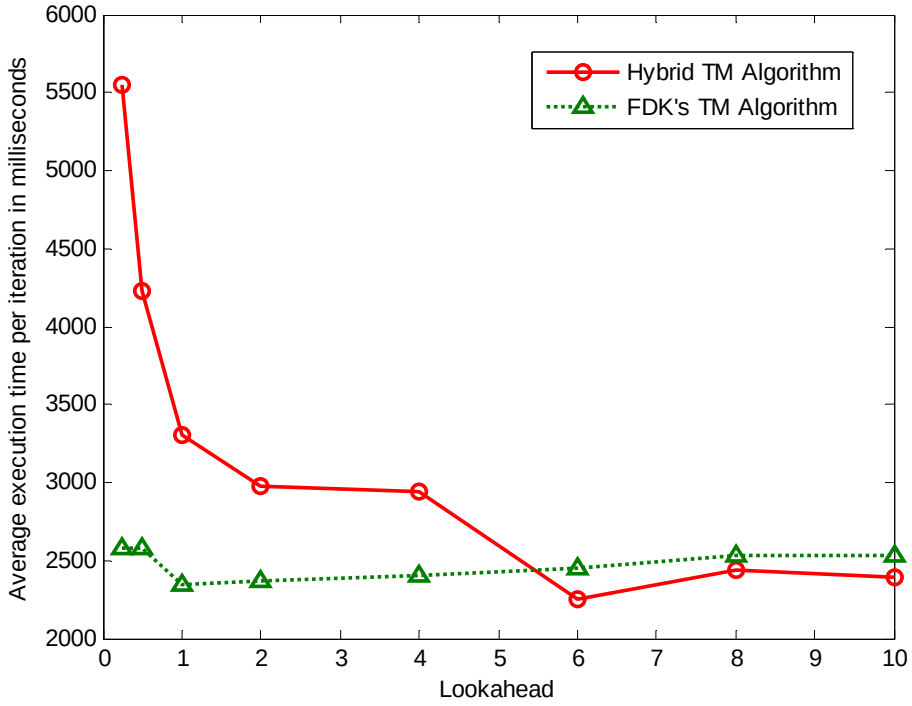


Figure 4.4.2: Simulation execution performance of F1 with respect to lookahead on the WAN

4.4.2 Scalability Comparison with respect to the Total Number of Federates

As discussed earlier, FDK's TM algorithm has the problem of redundant GALT calculations, which may make its performance worse than the hybrid TM algorithm when the number of federates in a federation becomes large. To illustrate this, an experiment was designed to compare the scalability of the two algorithms with respect to the total number of federates in a federation. All federates have the same lookahead value and request time advancement using NMR with time increment of 10. Among all federates, Federate1 is a slow federate which sleeps for 1000 milliseconds in each iteration, while all the other federates are fast federates which sleep for 200 milliseconds in each iteration. With FDK's TM algorithm, the time advancement requests made by the fast federates trigger redundant GALT calculations. With an increasing total number of federates, the number of redundant GALT calculations increases and this decreases the performance of FDK's TM algorithm.

The experiment was carried out both on the LAN and on the WAN, with the total number of federates increased from 2 to 7. It has been repeated with a small lookahead of 2, a medium lookahead of 6 and a large lookahead of 10.

On the LAN, all federates were run on separate nodes in the cluster. The performance results are shown in Figures 4.4.3, 4.4.4 and 4.4.5. Due to the redundant GALT calculations, the performance of FDK's TM algorithm decreases significantly with an increasing total number of federates. As there are no data message exchanges between federates in this benchmark test, the performance of FDK's TM algorithm is not affected much by the chosen lookahead values. When the lookahead is small, the hybrid algorithm has the "time creep" problem which has a similar overhead to the redundant GALT calculations in FDK's TM algorithm. As the lookahead is increased, its performance improves significantly and becomes much more scalable than FDK's TM algorithm with respect to the total number of federates.

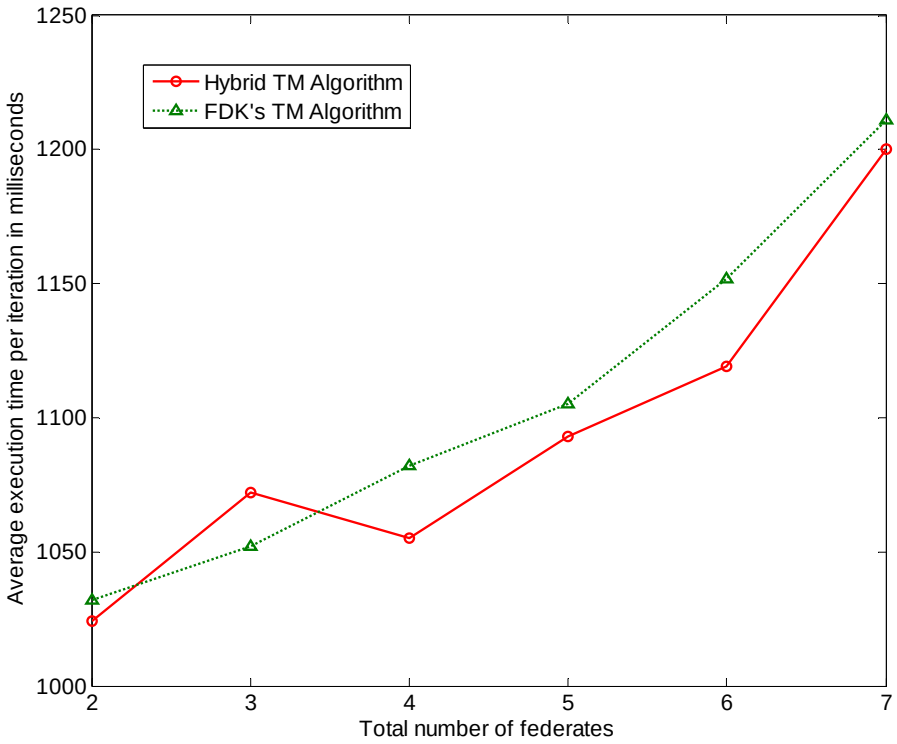


Figure 4.4.3: Simulation execution performance of F1 with respect to the toal number of federates in a federation on the LAN (LH=2)

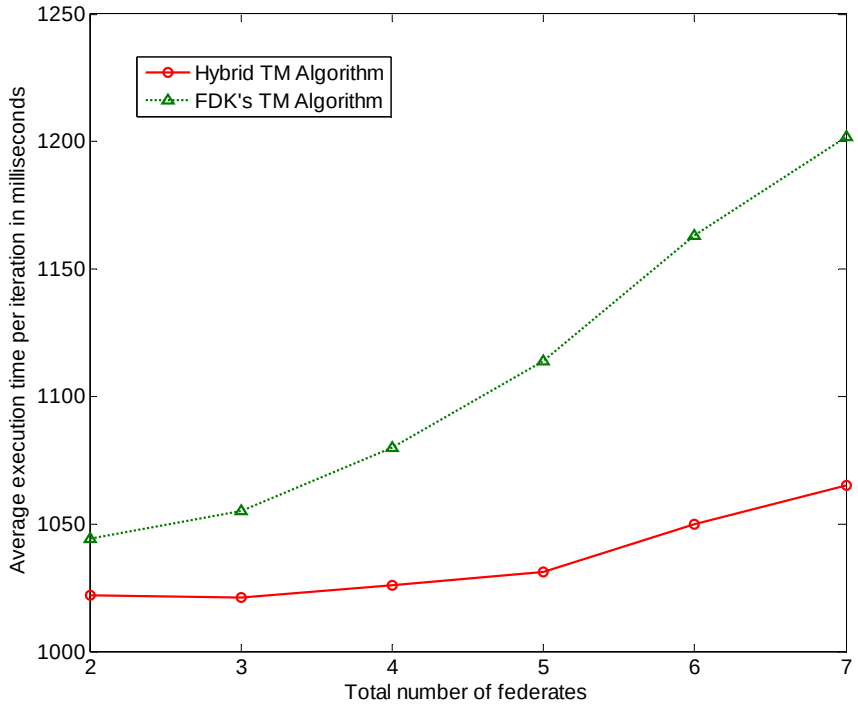


Figure 4.4.4: Simulation execution performance of F1 with respect to the total number of federates in a federation on the LAN (LH=6)

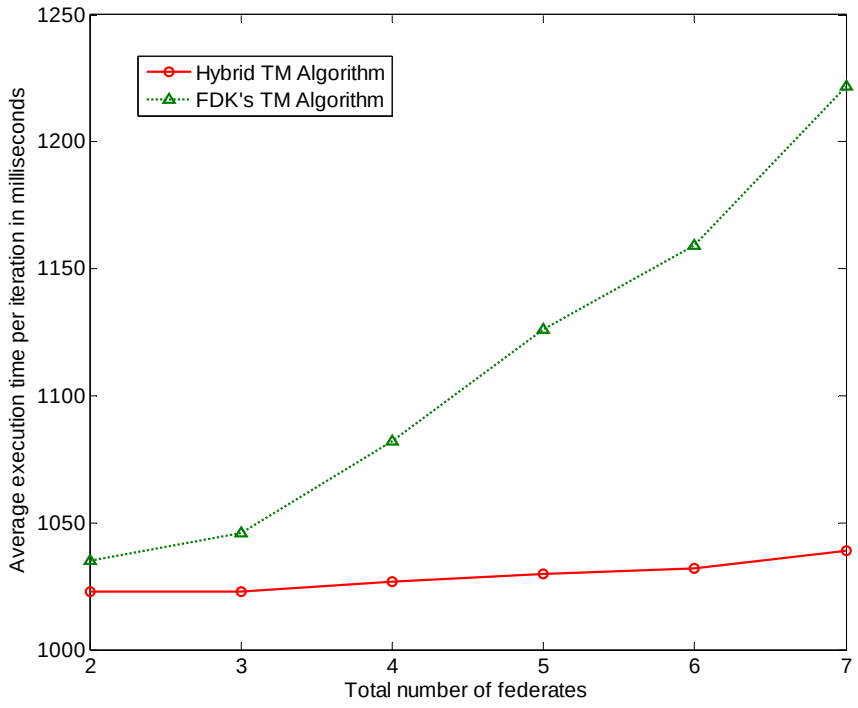


Figure 4.4.5: Simulation execution performance of F1 with respect to the total number of federates in a federation on the LAN (LH=10)

On the WAN, Federate1 and its LS were run at the Birmingham site while all the other federates and their LS were run at the NTU site. The performance results are shown in Figures 4.4.6, 4.4.7 and 4.4.8. It can be seen that the effect of redundant GALT calculations is enlarged when the execution is on the WAN. This makes FDK's TM algorithm much less scalable than the hybrid algorithm even with a small lookahead. As the lookahead is increased, the performance of FDK's TM algorithm does not vary much, while the performance of the hybrid algorithm improves further as the "time creep" problem is relieved.

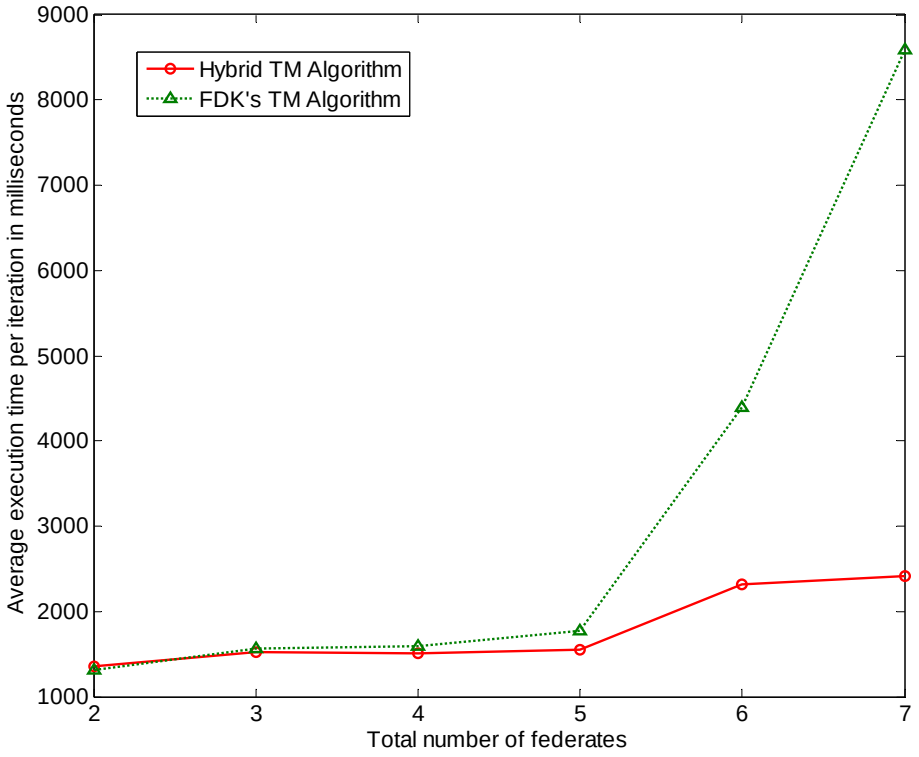


Figure 4.4.6: Simulation execution performance of F1 with respect to the total number of federates in a federation on the WAN (LH=2)

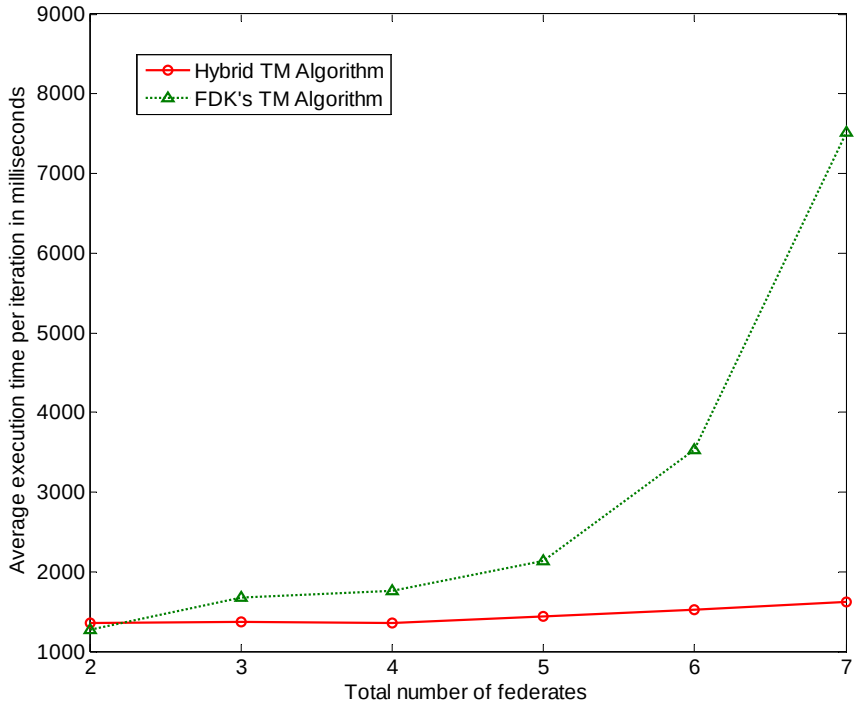


Figure 4.4.7: Simulation execution performance of F1 with respect to the total number of federates in a federation on the WAN (LH=6)

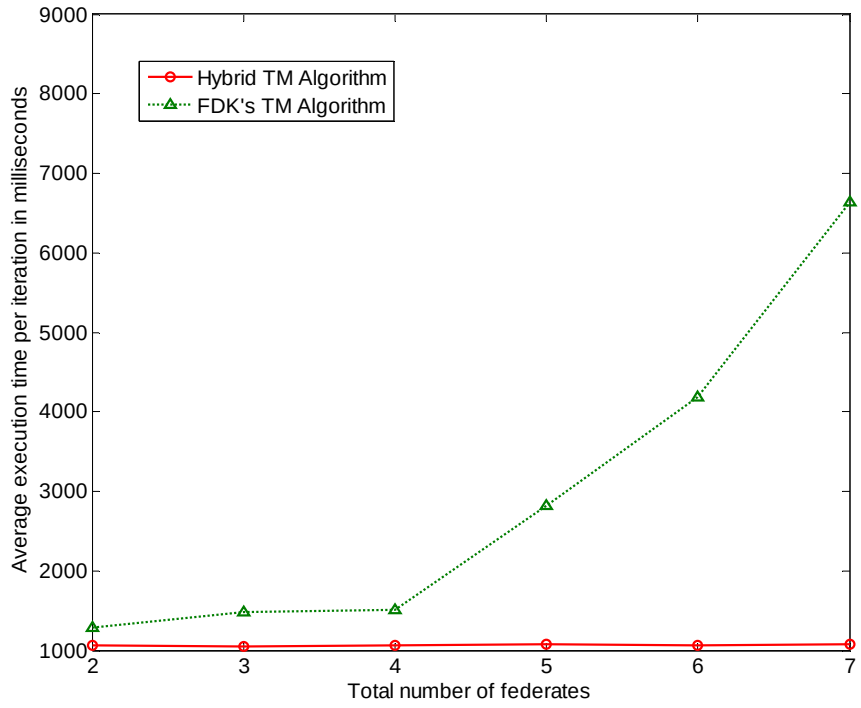


Figure 4.4.8: Simulation execution performance of F1 with respect to the total number of federates in a federation on the WAN (LH=10)

4.5 Summary

Traditional TM algorithms can be either synchronous or asynchronous. In general, synchronous algorithms are based on conditional information while asynchronous algorithms are based on unconditional information. Synchronous algorithms are not suitable for federations with low time-resolution federates which send conditional information with a low frequency; asynchronous algorithms have the “time creep” problem when the lookahead is small. To resolve the drawbacks of each algorithm by taking the advantages of the other, this chapter combines the two algorithms together and describes a hybrid algorithm based on both conditional and unconditional information. The idea is presented using HLA-specific terminology, but it applies to general parallel and distributed simulation. The three algorithms have been implemented into the SOHR framework, and both experimental results on the LAN and on the WAN show that the hybrid algorithm effectively benefits from the advantages of both synchronous and asynchronous algorithms and has the best performance among the three algorithms in a federation with a variable-processing-time federate. Since processing time of federates is normally uncertain especially in a Grid environment, the hybrid algorithm is a good choice for our Grid-based SOHR framework.

To compare the proposed hybrid algorithm with FDK's TM algorithm, we have also implemented FDK's TM algorithm into the SOHR framework. An analysis of the algorithm and experimental results show that FDK's TM algorithm completely prevents the “time creep” problem, which may still occur in the hybrid algorithm. However, due to redundant GALT calculations of FDK's TM algorithm, it is generally less scalable than the hybrid algorithm with respect to the total number of federates in a federation.

The implementation of multiple TM algorithms in SOHR shows its plug-and-play paradigm for an HLA RTI implementation.

Chapter 5

Efficient Sort-Based DDM Matching Algorithms for HLA Applications with a Large Spatial Environment

To achieve the third objective of this project as defined in Subsection 1.3.3, we first focus on the DDM matching process. In this chapter, we first describe an efficient sort-based matching algorithm for HLA applications with a large spatial environment, and then extend it to an algorithm which incrementally matches for dynamic region modifications. In Chapter 6, we will discuss the DDM implementation in SOHR.

5.1 Necessary and Sufficient Conditions for Region Overlapping

Before going into the details of the new algorithms, the necessary and sufficient conditions for an update region and a subscription region to overlap with each other are examined. As introduced in Section 1.1.2.4, an update region and a subscription region overlap if and only if they have common dimensions and their ranges overlap on each common dimension. According to the HLA 1516 specification [IEEE1516], a range is specified as a continuous semi-open interval [LB, UB) on a dimension with the minimum possible difference of 1 between LB and UB. The difference between LB and UB is defined as the range size on the dimension in this thesis. Consider one update region U and one

subscription region S. Dimension X is a common dimension of U and S. U's range on dimension X is [U_{LB}, U_{UB}) and S's range on dimension X is [S_{LB}, S_{UB}). There are three cases of comparisons between the two ranges as shown in Figure 5.1.1. When U_{LB} >= S_{UB} (Case 1), there is no overlap. When S_{LB} >= U_{UB} (Case 3), there is no overlap either. When U_{UB} > S_{LB} and S_{UB} > U_{LB} (Case 2), the two ranges overlap with each other. So the necessary and sufficient conditions for the two ranges to overlap with each other are the satisfaction of both Inequalities 5.1.1 and 5.1.2.

$$U_{UB} > S_{LB} \qquad (5.1.1)$$

$$S_{UB} > U_{LB} \qquad (5.1.2)$$

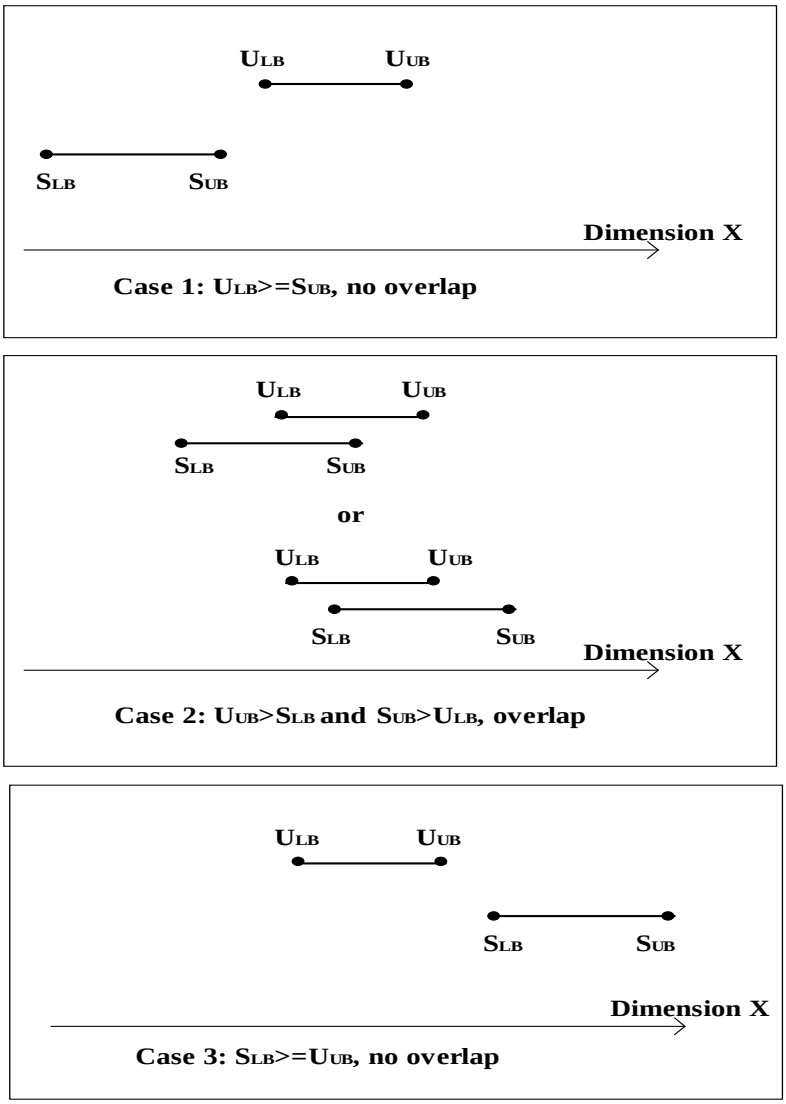


Figure 5.1.1: Overlapping condition of two ranges

5.2 An Efficient Sort-Based DDM Matching Algorithm

5.2.1 Algorithm Description

The efficient sort-based algorithm is based on the necessary and sufficient conditions for range overlapping derived in Subsection 5.1. In order to derive overlapping information for an update region U, we need to look for all subscription regions which satisfy Inequalities 5.1.1 and 5.1.2 on each common dimension. If the S_{LB} and S_{UB} of all subscription regions are separately sorted per dimension, the searching process is simplified. In HLA applications with a large spatial environment, each region range size of a federate on any dimension is normally confined by its influencing or sensing range. An assumption is made that the maximum update region range size maxURRS and the maximum subscription region range size maxSRRS on any dimension can be calculated based on the information of all regions. Note that maxURRS and maxSRRS can vary dynamically based on region modifications. The following conditions should be satisfied:

$$S_{LB} >= S_{UB}\text{-maxSRRS} \qquad (5.2.1)$$

$$U_{LB} >= U_{UB}\text{-maxURRS} \qquad (5.2.2)$$

From Inequalities 5.1.2, 5.2.1 and 5.2.2, we have:

$S_{LB} >= S_{UB}$-maxSRRS

$> U_{LB}$-maxSRRS

$> U_{UB}$-maxURRS-maxSRRS

With Inequality 5.1.1, we thus have:

$$U_{UB}\text{-maxURRS-maxSRRS} < S_{LB} < U_{UB} \qquad (5.2.3)$$

Similarly, we can derive:

$$U_{LB} < S_{UB} < U_{LB}+\text{maxURRS}+\text{maxSRRS} \qquad (5.2.4)$$

As shown in Figure 5.2.1, we map U_{UB} onto the S_{LB} sorted list and then scan backwards until $S_{LB} <= U_{UB}$-maxURRS-maxSRRS. In this way, a list of subscription regions SList1 is obtained in the shaded range of Figure 5.2.1(a). Similarly, we map U_{LB} onto the S_{UB} sorted list and then scan forwards until $S_{UB} >= U_{LB}$+maxURRS+maxSRRS. In this way, a list of

subscription regions Slist2 is obtained in the shaded range of Figure 5.2.1(b). The intersection of SList1 and SList2 contains all the subscription regions with ranges overlapping with U on this dimension. For simplicity of description, we assume all regions are specified using the same set of dimensions in this Chapter. The above process needs to be repeated for each dimension and the final intersection of all these SList1, SList2 is the list of subscription regions overlapping with the update region U. For scenarios where regions have different sets of dimensions, the efficient sort-based algorithm can be easily adapted.

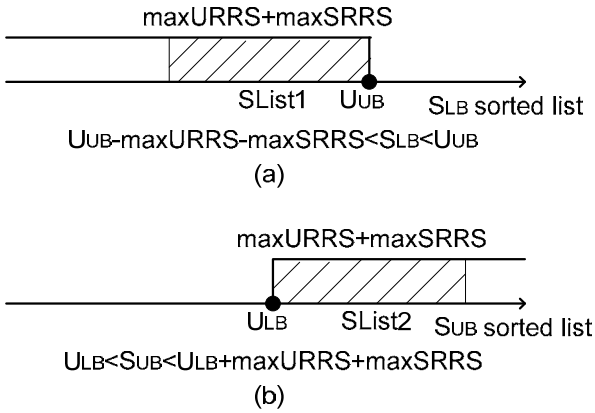


Figure 5.2.1: Searching process of overlapping regions

The data structure of the efficient sort-based algorithm is shown in Figure 5.2.2. We denote the number of update and subscription regions as numOfUR and numOfSR respectively. The array regions keeps information of all regions. D denotes the number of dimensions. For each dimension, there is one separate copy of four BoundLists and two long integers. The URLBList keeps lower bounds of all update regions in a list. Similarly, the URUBList keeps upper bounds of all update regions; the SRLBList keeps lower bounds of all subscription regions; the SRUBList keeps upper bounds of all subscription regions. Each item of a BoundList has two elements: a region handle (i.e., an index to the regions array) and a bound value. The maxURRS keeps the maximum range size of update regions. The maxSRRS keeps the maximum range size of subscription regions. The information of a region consists of a region handle, the specification of a range on each dimension, two position arrays and an integer vector. For an update region U and any dimension d, LBPosInLBList[d] remembers the position of U's lower bound on

Dimension d in the bound list URLBList[d]; UBPosInUBList[d] remembers the position of U's upper bound on Dimension d in the bound list URUBList[d]. Similarly for a subscription region S and any dimension d, LBPosInLBList[d] remembers the position of S's lower bound on Dimension d in the bound list SRLBList[d]; UBPosInUBList[d] remembers the position of S's upper bound on Dimension d in the bound list SRUBList[d]. The OLInfo is an integer vector which keeps the handles of all overlapping regions for this region. If this region is an update region, its OLInfo remembers its overlapping subscription regions; if this region is a subscription region, its OLInfo remembers its overlapping update regions.

```
Region regions[numOfUR+numOfSR];      // all regions
BoundList URLBList[D];    // update region lower bound list
BoundList URUBList[D];    // update region upper bound list
BoundList SRLBList[D];    // subscription region lower bound list
BoundList SRUBList[D];    // subscription region upper bound list
long maxURRS[D];   // max update region range size
long maxSRRS[D];   // max subscription region range size

class Region
{
        int regionHandle;
        Range ranges[D];
        int LBPosInLBList[D];       // LB position in LB list
        int UBPosInUBList[D];       // UB position in UB list
        IntVector OLInfo;           // overlapping information
}
```

Figure 5.2.2: Data structure of the efficient sort-based algorithm

The static matching of all regions by the efficient sort-based algorithm is shown in Figure 5.2.3. The algorithm first inserts all region bounds into the respective dimensions' BoundLists and calculates the maxURRS and maxSRRS for each dimension (Steps 1-16). Next, it heapsorts all BoundLists in ascending order (Steps 17-23). Then, the position arrays of all regions are initialized (Steps 24-30). Finally, the overlapping information of all regions can be derived by going through all update regions (Steps 31-55).

```
1.  for each update region U
2.      for each dimension d
3.      {
4.          insert (U,ULB[d]) into URLBList[d];
5.          insert (U,UUB[d]) into URUBList[d];
6.          if (UUB[d]-ULB[d]>maxURRS[d])
7.                  maxURRS[d]=UUB[d]-ULB[d];
8.      }

9.  for each subscription region S
10.     for each dimension d
11.     {
12.         insert (S,SLB[d]) into SRLBList[d];
13.         insert (S,SUB[d]) into SRUBList[d];
14.         if (SUB[d]-SLB[d]>maxSRRS[d])
15.                 maxSRRS[d]=SUB[d]-SLB[d];
16.     }

17. for each dimension d
18. {
19.     heapsort URLBList[d];
20.     heapsort URUBList[d];
21.     heapsort SRLBList[d];
22.     heapsort SRUBList[d];
23. }

24. for each dimension d
25. {
26.       initialize LBPosInLBList[d] of all update regions by scanning URLBList[d] once;
27.       initialize UBPosInUBList[d] of all update regions by scanning URUBList[d] once;
28.       initialize LBPosInLBList[d] of all subscription regions by scanning SRLBList[d] once;
29.       initialize UBPosInUBList[d] of all subscription regions by scanning SRUBList[d] once;
30. }

31. for each update region U
32. {
33.     for each dimension d
34.     {
35.       SList1[d]=Ø;
36.       binarysearch in SRLBList[d] for the last position P1 with SRLBList[d][P1].value<UUB[d];
37.       for p from P1 down to 0
          {
38.         if  SRLBList[d][p].value > UUB[d]-maxURRS[d]-maxSRRS[d]
39.               add SRLBList[d][p].handle to SList1;
40          else
41                break;
42        }
43.       SList2[d]=Ø;
44.       binarysearch in SRUBList[d] for the first position P2 with SRUBList[d][P2].value>ULB[d];
45.       for p from P2 up to numOfSR-1
46.       {
47.         if SRUBList[d].[p].value < ULB[d]+maxURRS[d]+maxSRRS[d]
48.               add SRUBList[d][p].handle to Slist 2;
49.         else
50.               break;
51.       }
52.     }
53.     find the intersection of all SList1[d] and SList2[d];
54.     for all S in the intersection, update OLInfo of U and S to indicate overlap;
55. }
```

Figure 5.2.3: Static matching of all regions by the efficient sort-based algorithm

```
1.  if (R is an update region)
2.  {
3.     for each subscription region S in regions[R].OLInfo
4.        remove R from regions[S].OLInfo;
5.     clear regions[R].OLInfo;

6.     for each dimension d
7.     {
8.        if RUB[d]-RLB[d]>maxURRS[d]
9.           maxURRS[d]=RUB[d]-RLB[d];

10.       if new RUB[d]<old RUB[d]
11.          for p from regions[R].UBPosInUBList[d]-1 down to 0
12.          {
13.             if URUBList[d][p].value>new RUB[d]
14.             {
15.                URUBList[d][p+1].handle=URUBList[d][p].handle;
16.                URUBList[d][p+1].value=URUBList[d][p].value;
17.                regions[URUBList[d][p].handle].UBPosInUBList[d]++;
18.             }
19.             else
20.             {
21.                URUBList[d][p].handle=R;
22.                URUBList[d][p].value=new RUB[d];
23.                regions[R].UBPosInUBList[d]=p;
24.                break;
25.             }
26.          }
27.       if new RUB[d]>old RUB[d]
28.          scan right similarly to Steps 11-26;
29.       if new RLB[d]<old RLB[d]
30.          scan left similarly to Steps 11-26;
31.       if new RLB[d]>old RLB[d]
32.          scan right similarly to Steps 11-26;

33.       calculate SList1[d] and SList2[d] similarly to Steps 35-51 in Figure 5.2.3;
34.    }
35.    update overlapping information similarly to Steps 53 and 54 in Figure 5.2.3;
36. }

37. else          // R is a subscription region
38. { similar to above .....}
```

Figure 5.2.4: Dynamic matching of one region modification by the efficient sort-based algorithm

As mentioned in Subsection 2.3.4, Raczy's sort-based matching algorithm proposed in [RAC05] cannot deal with dynamic matching of a selective region modification without repeating the static matching of all regions, which is very costly. Our efficient sort-based matching algorithm is applicable to dynamic matching of region modifications as shown in Figure 5.2.4. When a modification of Region R is received, it first determines whether

R is an update region or subscription region (Step 1). If R is an update region, the original overlapping information related to R is removed (Steps 3-5); the new overlapping information for R can be found (Steps 6-35) in almost the same way as the static matching algorithm shown in Figure 5.2.3 except that the URUBList and URLBList on each dimension need to be adjusted to make sure that they remain sorted after the region modification (Steps 10-32). For any dimension d, the adjustment on URUBList[d] starts from R's old UBPosInUBList[d] (Step 11); if R_{UB}[d] is decreased (Step 10), we scan left to look for its new position on URUBList[d] (Step 11); if R_{UB}[d] is increased (Step 27), we scan right to look for its new position (Step 28). Similarly, the adjustment on URLBList[d] starts from R's old LBPosInLBList[d] (Steps 29-32). Since R's UBPosInUBList[d] and LBPosInLBList[d] are stored in its region information, they can be directly fetched. This is an improvement over the version of the algorithm in [PAN07] which uses a binary search. If R is a subscription region, the procedure is symmetric to the update region case (Steps 37 and 38).

5.2.2 Theoretical Storage and Computational Complexity Analysis

The major parameters used in the algorithm's complexity analysis are as follows.

- The number of update regions: N.
- The number of subscription regions: for simplicity, assumed to be the same as the number of update regions (N).
- The number of dimensions: D. D is assumed to be 2 to simplify the analysis.
- The upper bound for each dimension is D_{UB}. Each dimension extends over [0, D_{UB}].
- As described above, in a large spatial environment, a federate's region range size on any dimension is normally confined by its influencing or sensing range. Parameter maxRS is defined as the maximum range size of any region on any dimension. The range size of a region on any dimension is assumed to be uniformly distributed over [1, maxRS].

- In a large spatial environment, each moving entity has a maximum speed and its region modification is restricted by this. Parameter maxBS is defined as the maximum shift of a region's bound with regard to its previous value on any dimension. The shift value is assumed to be uniformly distributed over [0, maxBS]. The shift direction of a region is random.

To analyze the storage complexity of the efficient sort-based algorithm, Figure 5.2.2 is reviewed. From the perspective of a particular region, the probability that another region overlaps with it is $O(maxRS/D_{UB})^D$. Then the average number of overlapping regions for a given region is $O(maxRS/D_{UB})^2 \times N$. So OLInfo of all regions require $O((maxRS/D_{UB})^2 \times N^2)$ storage. Since all the other information of all regions require O(N) storage, the total storage for all regions (i.e., the array, regions) is $O((maxRS/D_{UB})^2 \times N^2)$. The total storage for BoundLists is O(N). The information of the maximum range sizes requires O(1) storage. To sum up, the total storage complexity of the efficient sort-based algorithm is $O((maxRS/D_{UB})^2 \times N^2)$. Though it is quadratic storage complexity with respect to N, the actual storage requirement depends on the ratio of $maxRS/D_{UB}$ which is normally very small in HLA applications with a large spatial environment.

To analyze the computational complexity of the efficient sort-based algorithm for static matching of all regions, Figure 5.2.3 is reviewed. The process of inserting the regions' bounds into the respective BoundLists and calculating the maximum range sizes (Steps 1-16) requires O(N) computation. The process of heapsorting all BoundLists (Steps 17-23) requires $O(N \times logN)$ computation. The process of initializing LBPosInLBList and UBPosInUBList of all regions (Steps 24-30) requires O(N) computation. Both Steps 36 and 44 require O(logN) computation using a binary search. The code sections of Steps 37-42 and of Steps 45-51 require $O(maxRS/D_{UB} \times N)$ computation. Step 53 performs processing which is dependent on the sizes of the SLists and requires $O(maxRS/D_{UB} \times N)$ computation. The computational complexity of Step 54 is proportional to the average number of overlapping subscription regions for an update region, which is

$O((maxRS/D_{UB})^2 \times N)$. The ratio of $maxRS/D_{UB}$ is smaller than or equal to 1, so the computational complexity of Steps 33-54 is $O(maxRS/D_{UB} \times N)$. Since it is repeated for each update region (Step 31), the computational complexity of Steps 31-55 is $O(maxRS/D_{UB} \times N^2)$. So the total computational complexity of the efficient sort-based algorithm for matching all regions is $O(maxRS/D_{UB} \times N^2)$. Though it is quadratic with respect to N, the actual computational cost depends on the ratio of $maxRS/D_{UB}$ which is normally very small in HLA applications with a large spatial environment.

To analyze the computational complexity of the efficient sort-based algorithm for dynamic matching of one region modification, Figure 5.2.4 is reviewed. Steps 3-4 remove old overlapping information of relevant subscription regions. There are on average $O((maxRS/D_{UB})^2 \times N)$ subscription regions which overlap with the old position of the update region R and R needs to be deleted from their overlapping vectors. The computational complexity for the deletion of R from an overlapping vector is proportional to the vector size, which is $O((maxRS/D_{UB})^2 \times N)$. So the computational complexity of Steps 3-4 is $O((maxRS/D_{UB})^4 \times N^2)$. Steps 5 and 8-10 require O(1) computation. The computational complexity of Steps 11-26 depends on the difference between the new and old upper bound values. The larger the difference, the larger the portion of the URUBList that needs to be shifted to maintain the ascending order. Based on the assumed maxBS, Steps 11-26 require $O((maxBS)/D_{UB} \times N)$ computation. Similarly, Steps 27-32 require $O(maxBS/D_{UB} \times N)$ computation. Steps 33 and 35 are the same as the corresponding processing done in the static matching algorithm for one update region and their computational complexity is the same as analyzed in the last paragraph, which is $O(maxRS/D_{UB} \times N)$. To sum up, the total computational complexity of the efficient sort-based algorithm for dynamic matching of one region modification is $O((maxRS/D_{UB})^4 \times N^2)+O(maxRS/D_{UB} \times N)+O(maxBS/D_{UB} \times N)$. Though it is quadratic with respect to N at first sight, the actual computational cost depends on the ratios of $maxRS/D_{UB}$ and $maxBS/D_{UB}$, both of which are very small in HLA applications with a large spatial environment. With a small $maxRS/D_{UB}$ ratio (e.g., 0.001), the second

component of the total computational complexity dominates the first one unless N increases to $(D_{UB}/maxRS)^3$ (e.g., 10^9), so the total computational complexity can be approximated as $O(maxRS/D_{UB}\times N)+O(maxBS/D_{UB}\times N)$ which is linear.

5.3 An Extended Efficient Sort-Based DDM Matching Algorithm

In many spatial applications, it is too expensive for a federate to commit region modifications in each time step (or frame) as this causes too much computation cost for matching as well as communication overhead [MOR97]. A simple strategy is to commit region modifications periodically [MOR97], [BAS98]. Another strategy is to use a space-driven approach [BAS98], where a federate commits a region modification only when the position of the region has changed over a distance threshold since its last modification. So, it is normally the case that, in each time step, only a small subset of all regions in a federation are actually modified. To deal with this efficiently, in general, a static DDM matching algorithm is initially used at the beginning of a federation execution; during the execution, a dynamic matching algorithm is used in each time step to match for the modified regions only. Since the static matching cost is amortized over execution time, the dynamic matching performance is the key factor determining the DDM matching efficiency.

As discussed in Subsection 5.2.2, the dynamic matching performance of the efficient sort-based algorithm depends on the ratios of $maxRS/D_{UB}$ and $maxBS/D_{UB}$, both of which are very small in a large spatial environment. In most federation scenarios, maxBS is much smaller than maxRS as an entity's moving speed normally means it moves a much smaller distance than its sensing range. For example, in a typical federation scenario, the moving speed of a ground unit can be 10 meters per second on a dimension while the subscription region range size can be 5000 meters on a dimension [PET97]. If a region modification is periodically committed every 10 seconds, maxBS is equal to 100 meters,

which is much smaller than maxRS, which is equal to 5000 meters. If the efficient sort-based algorithm can be extended so that the dynamic matching performance becomes dependant only on the maxBS/D_{UB} ratio, it will become a more efficient algorithm at dynamic matching.

In this subsection, we describe such an extension. The extended version needs more storage space and its static matching cost is more than that of the original version. However, its dynamic matching becomes incremental in the sense that, the algorithm calculates only the difference between the new matching results and the old matching results for a modified region. In this way, its dynamic computational complexity is reduced to O(maxBS/D_{UB}×N), which is independent of maxRS/D_{UB}.

5.3.1 Algorithm Description

As derived in Subsection 5.1, the necessary and sufficient conditions for two regions to overlap with each other is the satisfaction of both Inequalities 5.1.1 and 5.1.2 on each common dimension. If there is a change of overlapping status between two regions, this must be due to the status change of their Inequality 5.1.1 or 5.1.2 on one or more of the common dimensions. So a status change of two regions' Inequality 5.1.1 or 5.1.2 on a dimension can be used to trigger the rematching process between the two regions. Suppose an update region U is modified. For detecting the status change of Inequality 5.1.1 relating U_{UB} and the lower bound of any subscription region (S_{LB}) on a dimension, we map the old U_{UB} and new U_{UB} onto the S_{LB} sorted list as shown in Figure 5.3.1. The set of subscription regions (SRSet) on the left side of U_{UB} on the S_{LB} sorted list have their respective Inequality 5.1.1 relationship with U satisfied. If U_{UB} shifts to the right as shown in Figure 5.3.1(a), the subscription regions in the shaded range are added to SRSet. If U_{UB} shifts to the left as shown in Figure 5.3.1(b), the subscription regions in the shaded range are eliminated from SRSet. In either case, the matching process is triggered for rechecking the overlapping status between U and any subscription region in the shaded range. The rechecking procedure between U and any subscription region is simply based on the

requirement of the satisfaction of both Inequalities 5.1.1 and 5.1.2 on all of their common dimensions. Similarly for Inequality 5.1.2 relating LB of an update region (U_{LB}) and UB of any subscription region (S_{UB}), a status change is detected and the corresponding matching process is triggered. The above incremental dynamic matching procedure is for a particular dimension and it needs to be repeated for each dimension.

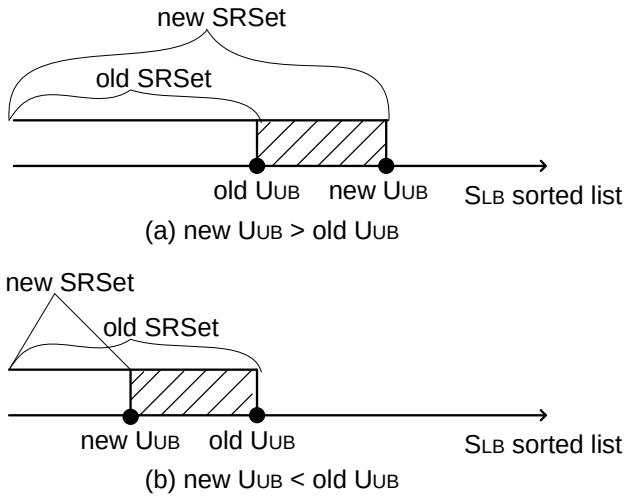


Figure 5.3.1: Status change detection of Inequality 5.1.1

The data structure of the extended efficient sort-based algorithm is shown in Figure 5.3.2. It is quite similar to the data structure of the efficient sort-based algorithm with the differences highlighted in bold. For access efficiency, the overlapping status on any dimension d is maintained as an N-by-N bit-matrix OLInfoMatrixPerDimension[d], of which the row denotes the update region and the column denotes the subscription region. The overall overlapping status on all dimensions is maintained as an N-by-N bit-matrix OLInfoMatrix. Another two position arrays, namely UBPosInLBList and LBPosInUBList, are added to the definition of a region. For an update region U and any dimension d, UBPosInLBList[d] remembers the position of the last entry in the bound list SRLBList[d] whose value is smaller than U's upper bound on dimension d (note: when no such entry can be found, UBPosInLBList[d] is set to -1 assuming the list index starts from 0); LBPosInUBList[d] remembers the position of the first entry in the bound list SRUBList[d] whose value is larger than U's lower bound on dimension d (note: when no such entry can

be found, LBPosInUBList[d] is set to the size of SRUBList[d] assuming the list index starts from 0). Similarly for a subscription region S and any dimension d, UBPosInLBList[d] remembers the position of the last entry in the bound list URLBList[d] whose value is smaller than S's upper bound on dimension d; LBPosInUBList[d] remembers the position of the first entry in the bound list URUBList[d] whose value is larger than S's lower bound on dimension d.

```
Region regions[numOfUR+numOfSR];      // all regions
BitMatrix OLInfoMatrix;  // overlapping info matrix
// overlapping info matrix per dimension
BitMatrix OLInfoMatrixPerDimension[D];
BoundList URLBList[D];    // update region lower bound list
BoundList URUBList[D];    // update region upper bound list
BoundList SRLBList[D];    // subscription region lower bound list
BoundList SRUBList[D];    // subscription region upper bound list
long maxURRS[D];   // max update region range size
long maxSRRS[D];   // max subscription region range size

class Region
{
      int regionHandle;
      Range ranges[D];
      int LBPosInLBList[D];       // LB position in LB list
      int UBPosInUBList[D];       // UB position in UB list
      int UBPosInLBList[D];   // UB position in LB list
      int LBPosInUBList[D];   // LB position in UB list
}
```

Figure 5.3.2: Data structure of the extended eficient sort-based algorithm

The static matching of all regions by the extended efficient sort-based algorithm is shown in Figure 5.3.3. It is quite similar to the static matching mechanism of the efficient sort-based algorithm with the differences highlighted in bold. Steps 31-40 initialize the two new position arrays, LBPosInUBList and UBPosInLBList, of all regions. Steps 51 and 52 initialize the OLInfoMatrixPerDimension array. Step 55 initializes OLInfoMatrix.

```
Steps 1-30 are the same as Steps 1-30 in Figure 5.2.3;

31. for each update region U
32.       for each dimension d
33.       {
34.         binarysearch in SRUBList[d] for the first position P1 with SRUBList[d][P1].value>ULB[d];
35.         regions[U].LBPosInUBList[d]=P1;
36.         binarysearch in SRLBList[d] for the last position P2 with SRLBList[d][P2].value<UUB[d];
37.         regions[U].UBPosInLBList[d]=P2;
38.       }
39. for each subscription region S
40.       similar to Steps 32-38;

41. for each update region U
42. {
43.    for each dimension d
44.    {
45.       SList1[d]=Ø;
46.       P1=regions[U].UBPosInLBList[d];
47.       similar to Steps 37-42 in Figure 5.2.3;
48.       SList2[d]=Ø;
49.       P2=regions[U].LBPosInUBList[d];
50.       similar to Steps 45-51 in Figure 5.2.3;
51.       find the intersection of SList1 and SList2;
52.       for all S in the intersection, update OLInfoMatrixPerDimension[d] to indicate overlap;
53.    }
54.    find the intersection of the list intersections found for all dimensions in Step 51;
55.    for all S in this intersection, update OLInfoMatrix to indicate overlap;
56. }
```

Figure 5.3.3: Static matching of all regions by the extended efficient sort-based algorithm

The dynamic matching of one region modification by the extended efficient sort-based algorithm is shown in Figure 5.3.4. Step 4 is the same as Steps 8-32 in Figure 5.2.4. Steps 5-21 are the incremental matching based on the status change of Inequality 5.1.1. If the new upper bound value of the modified region R is smaller than its old value (Step 5), we scan left on the SRLBList[d] from the old $R_{UB}[d]$ position (i.e., regions[R].UBPosInLBList[d]) to look for its new position. For each scanned subscription region, its overlapping status with R on the current dimension d is rechecked using Inequalities 5.1.1 and 5.1.2 (Step 11). If the overlapping status on dimension d has changed, OLInfoMatrixPerDimension[d] is updated and the overlapping status of the two regions on all the other dimensions are read to check for any change on the overall overlapping status (Step 12). If the new upper bound value of R is larger than its old value, we scan right (Steps 20 and 21). Similarly, Steps 22-25 are the incremental matching based on the status change of Inequality 5.1.2. The dynamic matching mechanism of our extended efficient sort-based algorithm is similar to the sweep and prune algorithm introduced in Subsection 2.3.4.

The DDM matching process should be carried out only between update regions and subscription regions. The necessary and sufficient conditions for region overlapping, i.e. Inequalities 5.1.1 and 5.1.2, indicate that the value relationships between a U_{UB} and a S_{LB} or between a U_{LB} and a S_{UB} are important. To enable sort-based matching, the value relationships between bounds of the same type (i.e., U_{LB}, U_{UB}, S_{LB} or S_{UB}) are also necessary. However, the value relationship between a U_{LB} and a S_{LB} or between a U_{UB} and a S_{UB} or between a U_{LB} and a U_{UB} or between a S_{LB} and a S_{UB} does not affect matching results. The sweep and prune algorithm maintains all types of bounds on a dimension in a single sorted list, so half of the swaps during its insertion sort are due to value relationships which do not affect matching results and are thus wasteful. The extended efficient sort-based algorithm maintains four sorted lists per dimension and thus does not have wasteful processing, so it should be more efficient than the sweep and prune algorithm.

```
1. if (R is an update region)
2.     for each dimension d
3.     {
4.         same as Steps 8-32 in Figure 5.2.4;
5.         if new RUB[d]<old RUB[d]
6.             for p from regions[R].UBPosInLBList[d] down to 0
7.             {
8.                 if SRLBList[d][p].value≥new RUB[d]
9.                 {
10.                    regions[SRLBList[d][p].handle].LBPosInUBList[d]++;
11.                    recheck the overlapping status between R and SRLBList[d][p].handle;
12.                    update OLInfoMatrixPerDimension[d] and OLInfoMatrix accordingly;
13.                }
14.                else
15.                {
16.                    regions[R].LBPosInUBList[d]=p;
17.                    break;
18.                }
19.            }
20.        if new RUB[d]>old RUB[d]
21.            scan right similarly to Steps 6-19;
22.        if new RLB[d]<old RLB[d]
23.            scan left similarly to Steps 6-19;
24.        if new RLB[d]>old RLB[d]
25.            scan right similarly to Steps 6-19;
26.    }

29. else        // R is a subscription region
30. { similar to above .....}
```

Figure 5.3.4: Dynamic matching of one region modification by the extended efficient sort-based algorithm

5.3.2 Theoretical Storage and Computational Complexity Analysis

The same parameters used in Subsection 5.2.2 are used for the theoretical storage and computational complexity analysis of the extended efficient sort-based algorithm. To analyze the storage complexity, Figure 5.3.2 is reviewed. The region information (i.e., the array, regions) requires O(N) storage. The OLInfoMatrix and OLInfoMatrixPerDimension require $O(N^2)$ storage. All BoundLists need O(N) storage. The information of the maximum range sizes requires O(1) storage. To sum up, the total storage complexity of the extended efficient sort-based algorithm is $O(N^2)$. The extended efficient sort-based algorithm requires more storage than the efficient sort-based algorithm and Raczy's sort-based algorithm as it keeps (D+1) N-by-N bit-matrixes.

To analyze the computational complexity of the extended efficient sort-based algorithm for static matching of all regions, Figure 5.3.3 is reviewed. Steps 1-30 require O(N× logN) computation. The process of initializing LBPosInUBList and UBPosInLBList of all regions (Steps 31-40) requires O(N×logN) computation. Both Steps 46 and 49 require O(1) computation. Both Steps 47 and 50 require $O(\text{maxRS}/D_{UB}\times N)$ computation. Steps 51-55 require $O(\text{maxRS}/D_{UB}\times N)$ computation. So the computational complexity of Steps 43-55 is $O(\text{maxRS}/D_{UB}\times N)$. Since it is repeated for each update region (Step 41), the computational complexity of Steps 41-56 is $O(\text{maxRS}/D_{UB}\times N^2)$. So the total computational complexity of the extended efficient sort-based algorithm for matching all regions is $O(\text{maxRS}/D_{UB}\times N^2)$. Since the static matching of the extended efficient sort-based algorithm requires additional initialization of the bit-matrix per dimension and the UBPosInLBList and LBPosInUBList position arrays of all regions, its performance should be worse than the static matching performance of the efficient sort-based algorithm.

To analyze the computational complexity of the extended efficient sort-based algorithm

for dynamic matching of one region modification, Figure 5.3.4 is reviewed. Step 4 requires O(maxBS/D_{UB}×N) computation as analyzed in dynamic matching of the efficient sort-based algorithm. Steps 5, 20, 22 and 24 require O(1) computation. The computational complexity of Steps 6-19, 21, 23 and 25 depend on the difference between the new and old bound values and thus is O(maxBS/D_{UB}×N). So the computational complexity of Steps 4-25 is O(maxBS/D_{UB}×N). Steps 4-25 are repeated on each dimension, so the total computational complexity of the extended efficient sort-based algorithm for dynamic matching of one region modification is O(maxBS/D_{UB}×N). Since it only depends on the maxBS/D_{UB} ratio, it should be more efficient than the dynamic matching computational performance of the efficient sort-based algorithm in most federation scenarios.

Though the static matching computational performance of the extended efficient sort-based algorithm is worse than that of the efficient sort-based algorithm, the cost is amortized over simulation execution time as the dynamic matching computational performance is better.

5.4 Experiments and Results

In [RAC05], Raczy's sort-based algorithm has been shown to be the best DDM matching algorithm in many situations compared with the region-based and the hybrid algorithms. To show that our proposed algorithms can be even more efficient and to verify the correctness of the theoretical conclusions made in Subsections 5.2.2 and 5.3.2, four Java programs have been written to respectively execute the region-based, Raczy's sort-based, our efficient sort-based and extended efficient sort-based algorithms. Each program performs static matching of all regions once, then performs dynamic matching of a region modification for a large number of iterations and calculates the average dynamic matching time per iteration. As the matching in the grid-based approach (see Subsection 2.3.2) is not exact and its matching is relatively trivial compared with exact matching algorithms, we do not include the grid-based approach for matching comparison in this chapter. However, the grid-based approach is included in Chapter 6 when comparing overall

performance of different DDM approaches.

The same assumptions as described in Subsection 5.2.2 are made. The initial position of each region is uniform-randomly generated in the routing space. For the dynamic matching, a random subscription or update region is modified in one iteration. The long data type is utilized in the bit-matrix implementation in both Raczy's sort-based and our extended efficient sort-based algorithms. The row update operation of the bit-matrix is based on the long data type, so the performance of Raczy's sort-based algorithm is optimized. Since only the performance of the DDM matching is measured, all programs were run on a single PC with dual Intel core 2.66 GHz CPUs, 2GB of RAM and a Windows XP OS.

5.4.1 Matching Performance with respect to N

An HLA application with a large spatial environment was considered with D_{UB}=20000, maxRS=20 and maxBS=1. With N extending from 3000 to 30000 in incremental steps of 3000, the total execution time of each algorithm for static matching of all regions is shown in Figure 5.4.1 and the average execution time of each algorithm for dynamic matching of one region modification is shown in Figure 5.4.2. With regard to storage requirements, an interesting aspect found is that, with Sun Java JVM's default heap size of 64 megabytes for Windows [SUN], Raczy's sort-based algorithm fails to run for N>=24000 due to the lack of memory. With N equal to 24000, the N-by-N bit-matrix of overlapping information requires 24000×24000 bits which is larger than the default heap size. As our extended efficient sort-based algorithm maintains more bit-matrices, it fails to run for N>=12000. To enable the execution of both these two algorithms, the JVM's heap size was increased accordingly when carrying out the experiments. In contrast, with the default heap size, our efficient sort-based algorithm works well for the whole N range. All these results agree with our previous theoretical storage complexity analysis.

Figure 5.4.1(a) shows the static matching computational performance with respect to N

for all algorithms. For clarity of comparison, Figure 5.4.1(b) shows the same information for Raczy's sort-based, our efficient sort-based and extended efficient sort-based algorithms with a smaller time scale. Results show that the static matching time of each algorithm increases non-linearly with an increasing N. The region-based algorithm performs much worse than the other algorithms as expected. The efficient sort-based algorithm has the best performance due to the small maxRS/D_{UB} ratio (0.001) in the experimental large spatial environment. The extended efficient sort-based algorithm has worse performance than the efficient sort-based and Raczy's sort-based algorithms as its static matching requires more data initialization for later dynamic matching than the efficient sort-based algorithm.

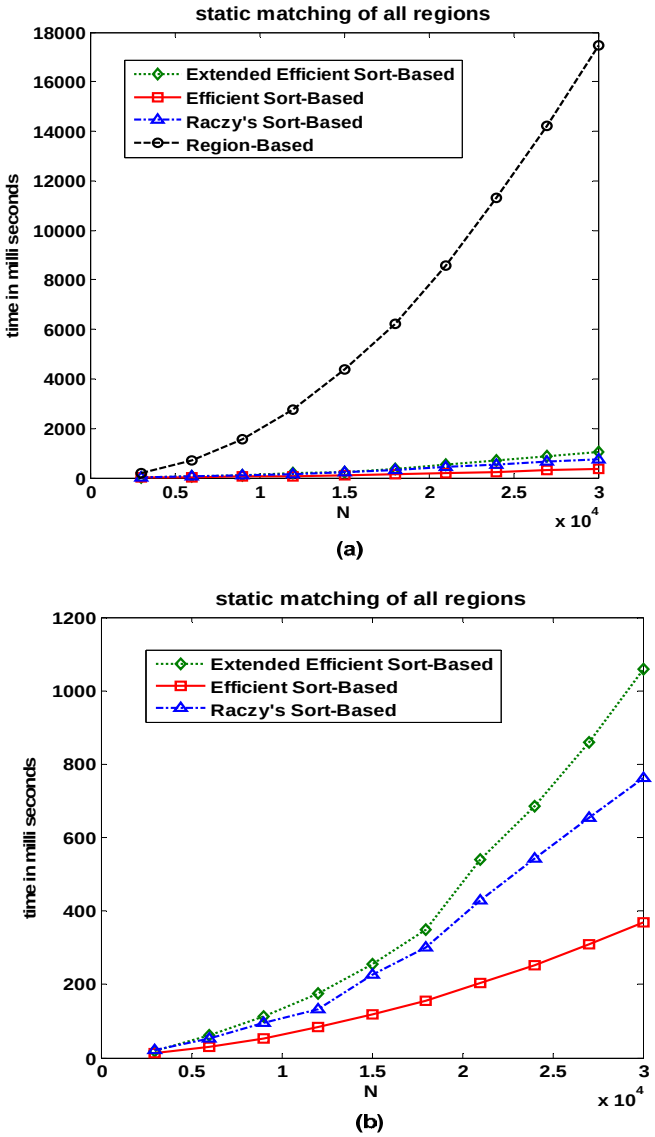


Figure 5.4.1: Static matching computational performance of all regions with respect to N

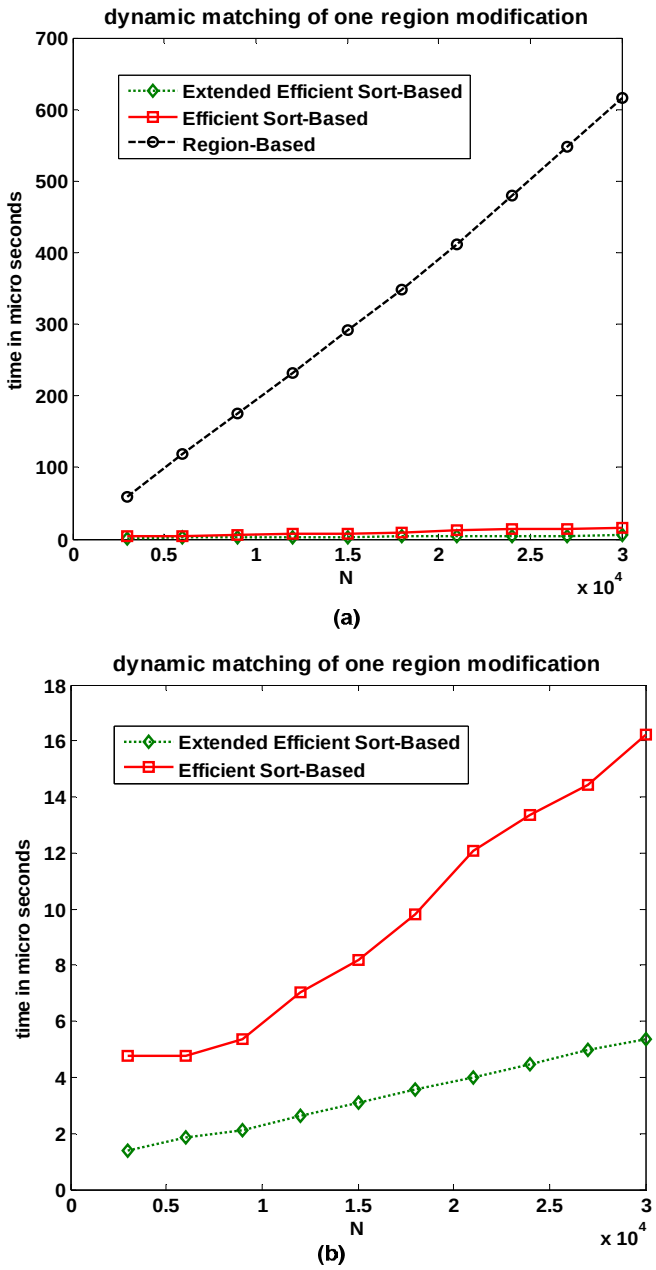


Figure 5.4.2: Dynamic matching computational performance of one region modification with respect to N

Figure 5.4.2(a) shows the dynamic matching computational performance with respect to N for all algorithms except Raczy's sort-based algorithm. For clarity of comparison, Figure 5.4.2(b) shows the same information for our efficient sort-based and extended efficient sort-based algorithms with a smaller time scale. Since Raczy's sort-based algorithm cannot dynamically deal with a region modification without reprocessing all regions, an assumption is made that its dynamic matching time of one region modification is the same

as its static matching time of all regions. Its performance is thus much worse than all the other algorithms, so its performance results are not shown in Figure 5.4.2. Results in Figure 5.4.2 show that the dynamic matching time of each of the three algorithms increases almost linearly with an increasing N. The efficient sort-based algorithm performs much better than the region-based algorithm due to the small maxRS/D_{UB} and maxBS/D_{UB} ratios in the experimental large spatial environment. The extended efficient sort-based algorithm has the best performance, because its dynamic matching computational performance only depends on the maxBS/D_{UB} ratio (0.00005), which is much smaller than the maxRS/D_{UB} ratio (0.001).

5.4.2 Matching Performance with respect to maxRS/D_{UB} ratio

To test the matching performance with respect to the maxRS/D_{UB} ratio, an experiment was carried out with N=18000, D_{UB}=20000 and maxBS=1 while extending maxRS from 20 to 200 in incremental steps of 20. The static matching computational performance of all regions is shown in Figure 5.4.3 and the dynamic matching computational performance of one region modification is shown in Figure 5.4.4.

Figure 5.4.3(a) shows the static matching computational performance with respect to the maxRS/D_{UB} ratio for all algorithms. For clarity of comparison, Figure 5.4.3(b) shows the same information for Raczy's sort-based, our efficient sort-based and extended efficient sort-based algorithms with a smaller time scale. Results show that both region-based and Raczy's sort-based algorithms are not affected much by the maxRS/D_{UB} ratio while the static matching time of both our efficient sort-based and extended efficient sort-based algorithms increases almost linearly with an increasing maxRS/D_{UB} ratio. When the ratio is smaller than or equal to 0.002, the efficient sort-based algorithm has the best performance. When the ratio increases to 0.004 or larger, Raczy's sort-based algorithm becomes the best. The two curves for Raczy's sort-based and the efficient sort-based algorithms intersect at a ratio close to 0.003. The extended efficient sort-based algorithm has worse performance than the efficient sort-based and Raczy's sort-based algorithms as

its static matching requires more data initialization for later dynamic matching than the efficient sort-based algorithm. The region-based algorithm performs much worse than all the other algorithms.

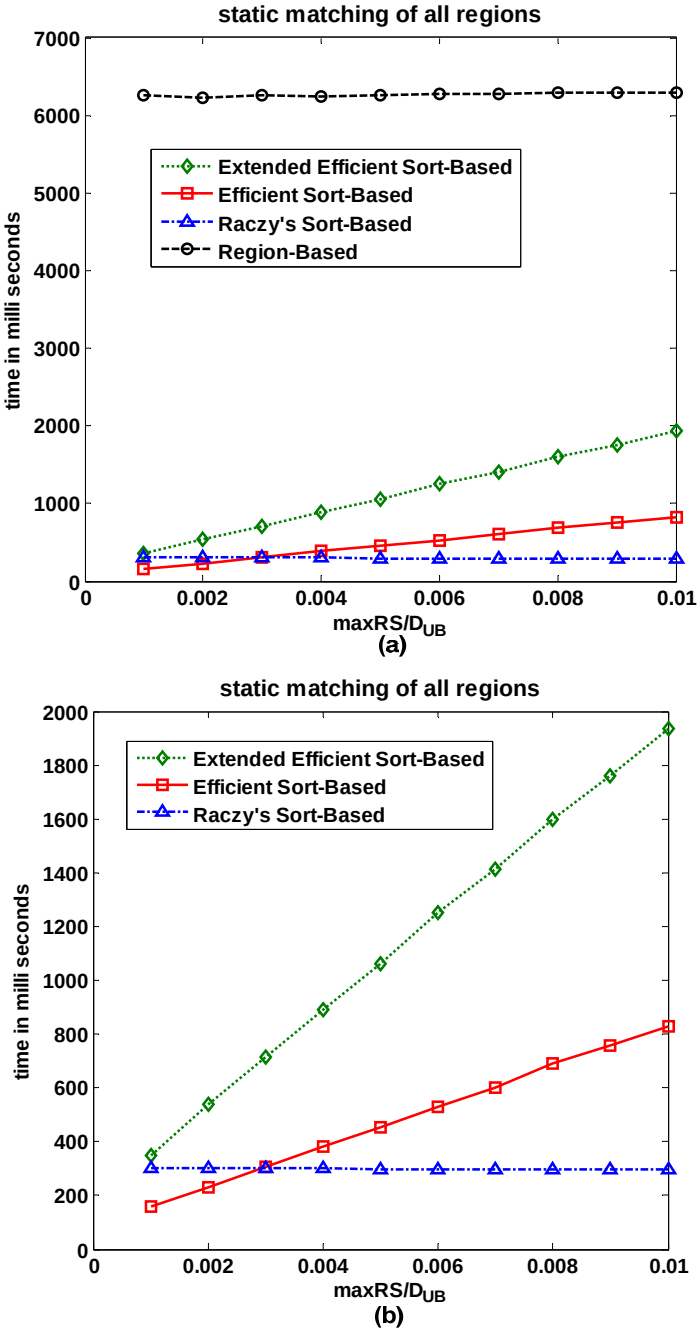


Figure 5.4.3: Static matching computational performance of all regions with respect to maxRS/D_{UB} ratio

Figure 5.4.4 shows the dynamic matching computational performance with respect to the maxRS/D_{UB} ratio for all algorithms except Raczy's sort-based algorithm. Since the dynamic matching of one region modification by Raczy's sort-based algorithm is the same as its static matching of all regions, its results are not shown in Figure 5.4.4. Results in Figure 5.4.4 show that the dynamic matching time of the efficient sort-based algorithm increases almost linearly with an increasing maxRS/D_{UB} ratio while the performance of both the region-based and extended efficient sort-based algorithms is not affected much by the maxRS/D_{UB} ratio. The extended efficient sort-based algorithm has the best performance due to the small maxBS/D_{UB} ratio (0.00005).

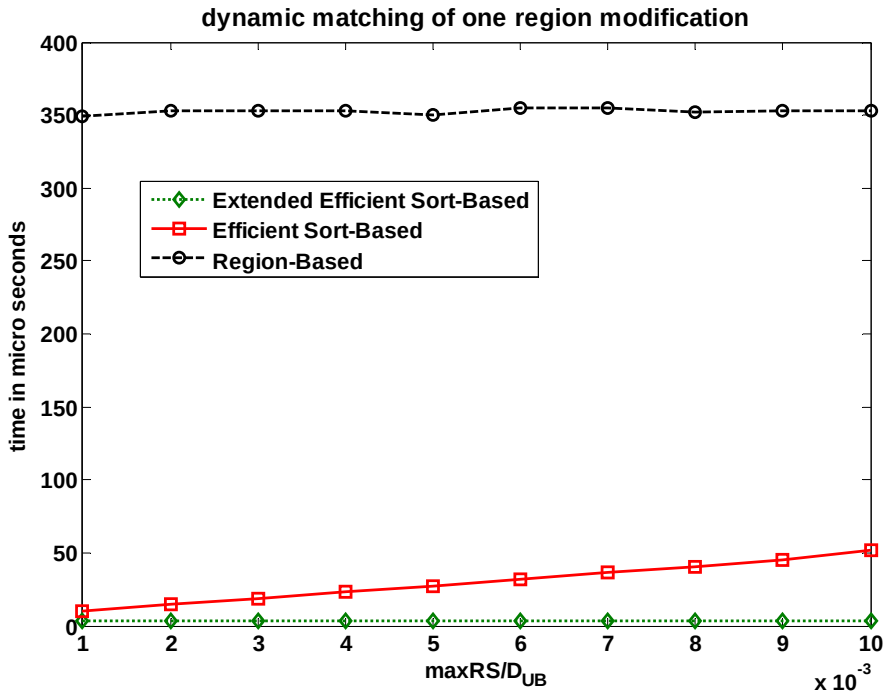


Figure 5.4.4: Dynamic matching computational performance of one region modification with respect to maxRS/D_{UB} ratio

5.4.3 Matching Performance with respect to maxBS/D_{UB} ratio

To test the matching performance with respect to the maxBS/D_{UB} ratio, an experiment was carried out with N=18000, D_{UB}=20000 and maxRS=20 while extending maxBS from 1 to 10 in incremental steps of 1. The static matching of all algorithms is not affected by the maxBS/D_{UB} ratio. Figure 5.4.5(a) shows the dynamic matching computational

performance with respect to the maxBS/D_{UB} ratio for all algorithms except Raczy's sort-based algorithm. For clarity of comparison, Figure 5.4.5(b) shows the same information for our efficient sort-based and extended efficient sort-based algorithms with a smaller time scale. Since the dynamic matching of one region modification by Raczy's sort-based algorithm is the same as its static matching of all regions, its results are not shown in Figure 5.4.5.

Results in Figure 5.4.5 show that the dynamic matching time of both the efficient sort-based and extended efficient sort-based algorithms increase almost linearly with an increasing maxBS/D_{UB} ratio. In the efficient sort-based algorithm, the maxBS/D_{UB} factor only exists in adjusting the sorted BoundLists (Steps 10-32 in Figure 5.2.4). However, in the extended efficient sort-based algorithm, the maxBS/D_{UB} factor exists in adjusting the sorted BoundLists (Step 4 in Figure 5.3.4) and the incremental matching (Steps 5-29 in Figure 5.3.4). So, the extended algorithm has a larger slope with respect to the maxBS/D_{UB} ratio than the original algorithm as shown in Figure 5.4.5. When the ratio is smaller than or equal to 3×10^{-4}, the extended efficient sort-based algorithm has the best performance. When the ratio increases to 4×10^{-4} or larger, the efficient sort-based algorithm becomes the best. Although the region-based algorithm is not affected much by the maxBS/D_{UB} ratio, it performs much worse than both the efficient sort-based and extended efficient sort-based algorithms.

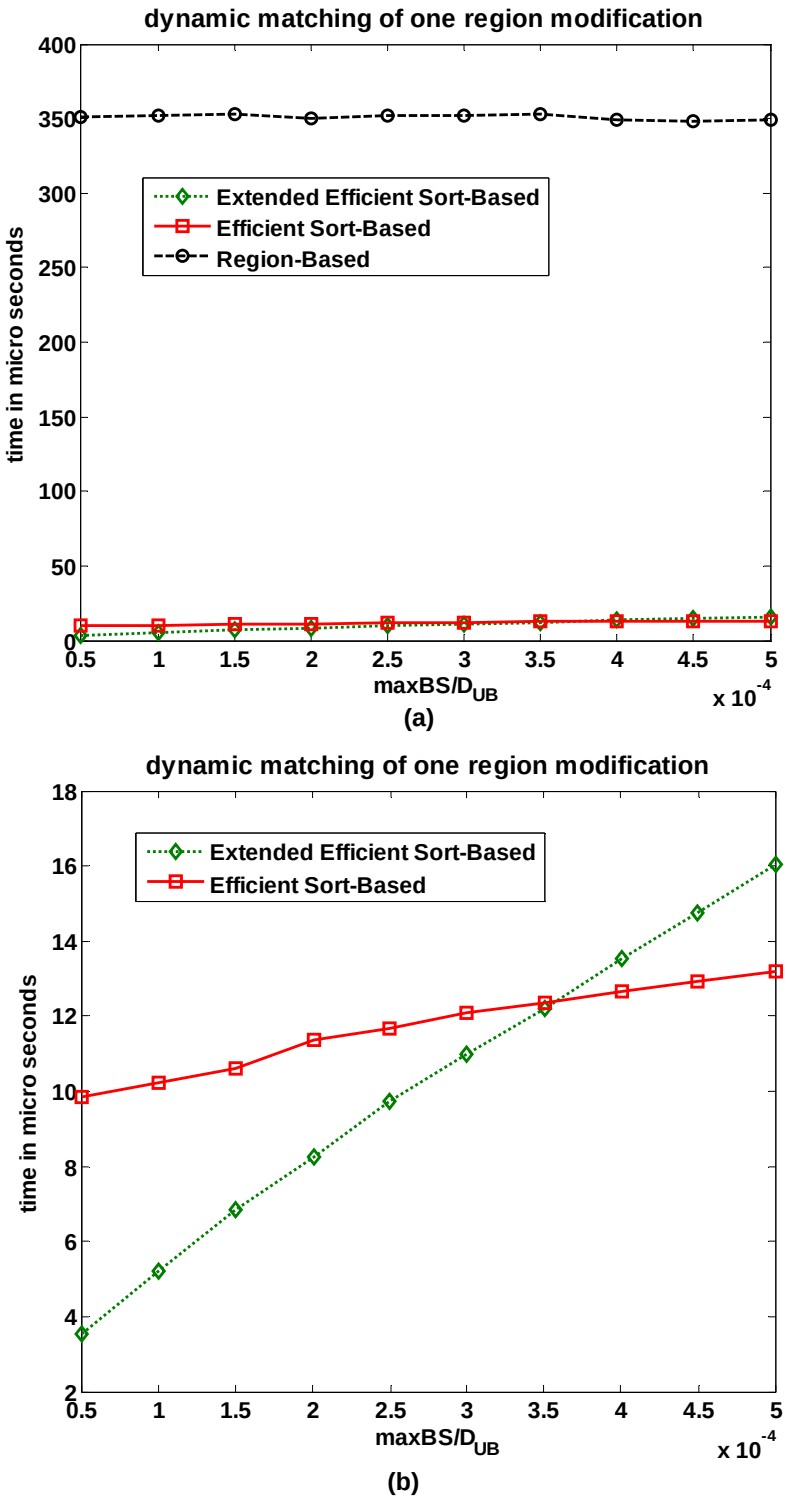


Figure 5.4.5: Dynamic matching computational performance of one region modification with respect to maxBS/D_{UB} ratio

5.5 Summary

In this Chapter, we first describe an efficient sort-based matching algorithm for HLA applications with a large spatial environment. Theoretical analysis concludes that, both the storage complexity and the static matching performance of the efficient sort-based algorithm are dependent on the maxRS/D_{UB} ratio while its dynamic matching performance is dependent on both the maxRS/D_{UB} and maxBS/D_{UB} ratios. In a large spatial environment, the efficient sort-based algorithm should have good storage and computational scalability because the two ratios are normally very small.

Next, we extend the efficient sort-based algorithm to one which incrementally matches for dynamic region modifications. Theoretical analysis concludes that, the extended efficient sort-based algorithm needs more storage space and more static matching time than the efficient sort-based algorithm. However, its dynamic matching computational performance is only dependant on the maxBS/D_{UB} ratio and thus is generally more efficient than the dynamic matching of the efficient sort-based algorithm. The extended efficient sort-based algorithm should be a good choice for dynamic matching in an application with a small maxBS/D_{UB} ratio without regard to the maxRS/D_{UB} ratio.

Experiments were carried out to test the matching performance of the two proposed algorithms with respect to the individual factors, namely the number of regions, the maxRS/D_{UB} ratio and the maxBS/D_{UB} ratio. Their experimental results are compared with the results of the region-based and Raczy's sort-based algorithms. All the experimental results agree with the theoretical conclusions.

Chapter

6

Implementation of Data Distribution Management Services in SOHR

To demonstrate the plug-and-play paradigm of SOHR, this chapter discusses the implementation of two DDM approaches, the grid-based approach and the extended efficient sort-based approach, in SOHR. Experiments have been carried out to compare their performance in different scenarios. As all the algorithms compared in Chapter 5 are exact matching algorithms, their communication performance will be the same if implemented in SOHR. For this reason, we only implemented the extended efficient sort-based algorithm and compared it with the grid-based approach.

6.1 DDM Implementation

We have implemented the grid-based approach and the extended efficient sort-based approach in SOHR. Both the two approaches have been implemented in a distributed manner, so that the major DDM work is done by LRI's DDM module while the DDMS is simply responsible for generating region handles and keeping an association between federates and their respective regions. The association information is fetched by a newly joining federate for its LRI's DDM module initialization. As the matching process only derives the overlapping information between regions, the association information is necessary for a sending federate to derive the list of receiving federates for its data transmission (connecting). The association information is updated whenever a new region

is created or an old region is deleted during the whole federation execution. To minimize latencies, data messages are sent in a peer-to-peer way for the routing process in our implementations.

6.1.1 Grid-Based DDM Implementation

The grid-based DDM approach was implemented in SOHR by implementing a Grid-Based DDM Module in the LRI structure and a new DDMS as shown in Figure 6.1.1. For each grid cell, the Grid-Based DDM Module maintains a list of remote subscription regions which overlap with the cell. When an attribute update or interaction is requested to be sent out with a local update region, the data message is sent to all federates associated with regions in the remote subscription lists of the cells with which the local update region overlaps (routing). An example is shown in Figure 6.1.2. There are nine grid cells with their respective remote subscription lists. U is a local update region while S1, S2 and S3 are remote subscription regions. When an attribute update is requested to be sent out with U, the regions associated with regions in the union of C5 and C8's remote subscription region list, i.e. S1 and S2, will receive the attribute update. As the grid-based matching is not exact, there will be irrelevant data messages sent, e.g. U and S1 in Figure 6.1.2. To further enable a receiver-side filtering mechanism, the sending update region information is piggybacked in the data message. It is checked against the subscription region information at the receiver side before the data message can be delivered to the receiving federate.

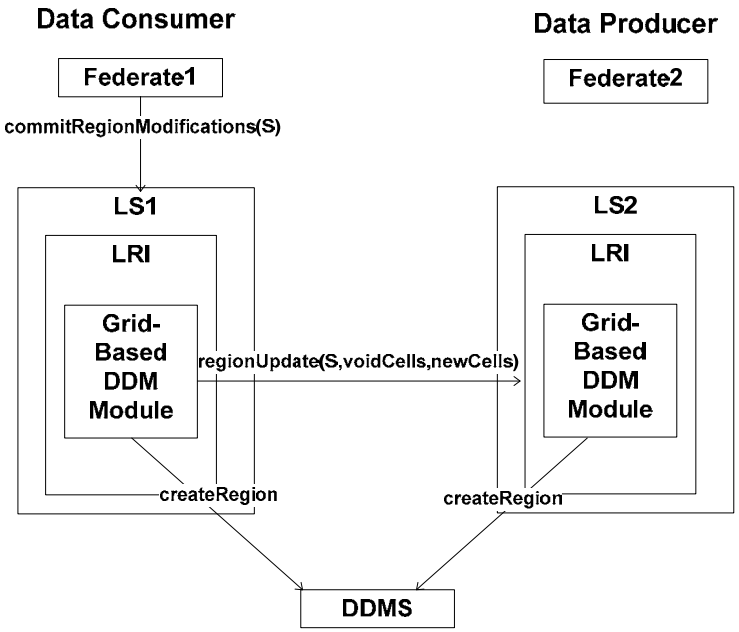


Figure 6.1.1: Grid-based DDM implementation

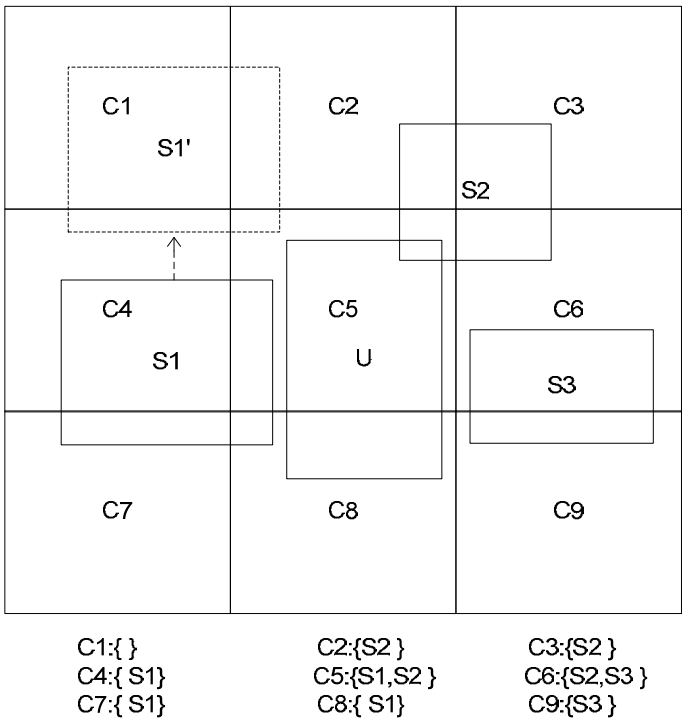


Figure 6.1.2: An example for the grid based DDM implementation

As shown in Figure 6.1.1, when a subscription region modification is committed (declaring) at the data consumer side, if there is any overlapping status change, other federates' corresponding Grid-Based DDM Modules need to be informed through a region

update message. The message should specify the modified subscription region handle, the void cells which overlapped with the region before the modification but do not overlap with the region after the modification, the new cells which did not overlap with the region before the modification but overlap with the region after the modification. Upon receiving a region update message, the Grid-Based DDM Module at the data producer side removes the specified subscription region from the remote subscription region lists of the void cells and adds the specified subscription region into the remote subscription region lists of the new cells. Each of these list updates is referred to as a List Update (LU) operation. For example, in Figure 6.1.2, if S1 changes its position to S1', the region update message is in the form of regionUpdate(S1,{C7,C8},{C1,C2}). This triggers four LU operations.

6.1.2 Extended Efficient Sort-Based DDM Implementation

The extended efficient sort-based DDM approach was implemented in a similar way to the implementation of the grid-based approach. Figure 6.1.3 shows the situation after DDM Module instances have been created. A sender-side filtering mechanism is adopted, so that the boundary information of subscription regions is sent between federates' LRIs (declaring) and the Extended Efficient Sort-Based DDM Module is responsible for matching based on the received boundary information. The matching is based on the extended efficient sort-based matching algorithm described in Subsection 5.3. Since its matching is exact, further receiver-side filtering is not needed for data message sending (routing).

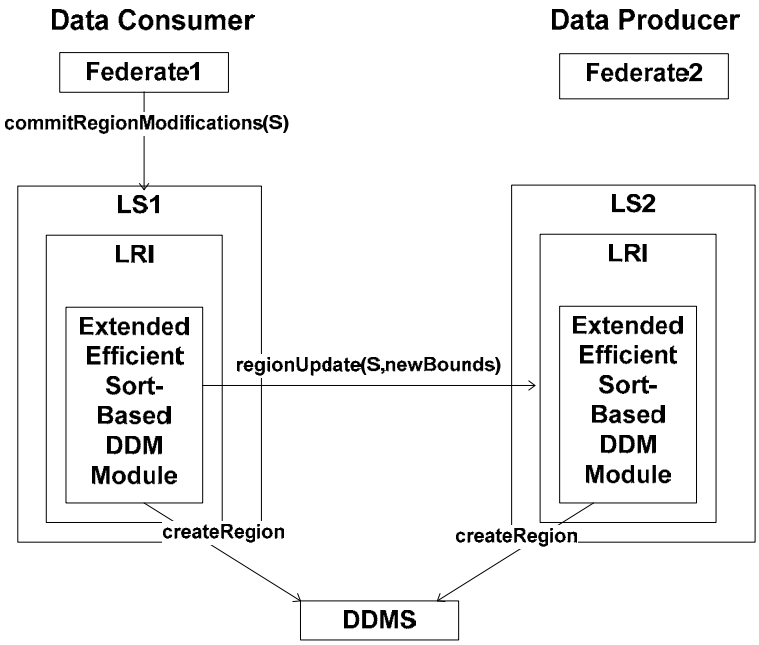


Figure 6.1.3: Extended efficient sort-based DDM implementation

6.2 Experiments and Results

To compare the performance of the two DDM implementations, experiments were carried out with a time-stepped federation of moving tanks. It contains two federates each of which simulates 50 moving tanks in a 40km×40km two dimensional routing space. In each time step of 1 minute simulation time, a tank has a certain probability of moving in a random direction (north, south, east or west). If a tank moves in a time step, it moves with a constant speed. Our experiments simulated highly dynamic scenarios. We did not use a specific mobility model for tanks because it may sacrifice the generality of the experimental conclusions. A tank has an update region and a subscription region surrounding its current position, and it sends its status update though an attribute update message in each time step. The whole federation lasts for simulation time of 100 minutes. All the following experiments were executed in a cluster with a 10Gb/s infiniBand connection. Each node of the cluster is installed with two dual core Xeon 3.0GHz CPUs, a 4GB RAM and a Redhat Enterprise Linux 4 OS. One federate and its LS were executed on one node; the other federate and its LS were executed on another node. All the other SOHR services were executed on a third node.

6.2.1 Performance with Basic Setup

The basic experimental setup has the tank moving probability equal to 0.25, the tank moving speed equal to 1000 meters per minute, the update region size equal to 200m×200m and the subscription region size equal to 2000m×2000m. The performance results are shown in Figure 6.2.1.

DDM Approach	RUM	LU	AUM	AUMVP	Message Total	Time (S)
Extended Efficient	2483	-	2504	100%	4987	122
Grid-Based 1×1	0	0	10000	26.04%	10000	145
Grid-Based 2×2	225	421	10000	29.17%	10225	146
Grid-Based 4×4	466	1895	9970	27.97%	10436	149
Grid-Based 8×8	927	7331	7913	34.02%	8840	143
Grid-Based 10×10	1195	11394	7193	38.57%	8388	139
Grid-Based 20×20	1283	43986	4695	52.03%	5978	129
Grid-Based 30×30	2436	97674	3876	62.46%	6312	135
Grid-Based 40×40	2446	172546	3926	70.12%	6372	145
Grid-Based 50×50	2431	267940	3436	76.51%	5867	150
Grid-Based 60×60	2449	385283	3574	77.95%	6023	164

RUM: Region Update Message LU: List Update AUM: Attribute Update Message
AUMVP: Attribute Update Message Valid Percentage

Figure 6.2.1: Simulation results of the basic setup

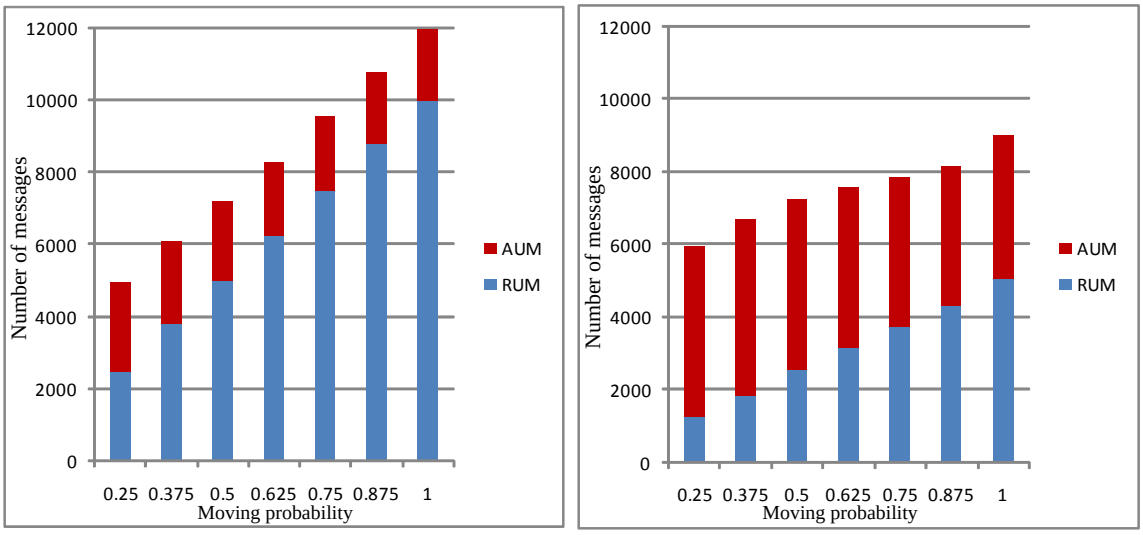
For the extended efficient sort-based approach, the number of Region Update Messages (RUM) is close to 2500, which is as expected since the moving probability is 0.25 and there are 100 tanks and 100 time steps of 1 minute. Since its matching is exact, the Attribute Update Message Valid Percentage (AUMVP) is 100%.

As the performance of the grid-based approach depends on the chosen size of the grid cells, we have varied the number of grid cells from 1×1 to 60×60. The results show that the number of RUM increases with an increasing number of cells, and becomes saturated when the number of cells approaches 30×30. This is because, as the size of a grid cell decreases towards the distance moved by a tank in one time step, the possibility that a tank movement causes a change of its subscription region's overlapping grid cells becomes larger and finally approaches 1. Other points to note with an increasing number

of cells include the increasing number of List Update (LU) operations, the decreasing number of exchanged Attribute Update Messages (AUM) and the increasing AUMVP. As the number of cells increases to 20×20, the total number of messages, which is the summation of RUM and AUM, decreases due to the decreasing AUM. This causes the total simulation time to decrease. As the number of cells increases further, the total number of messages may become lower (e.g. the 50×50 case) due to the increasing AUMVP and is expected to approach the total number of messages of the extended efficient sort-based approach. However, the simulation time increases due to the increasing number of list update operations.

In the basic experiment setup, the extended efficient sort -based approach is always better than the grid-based approach and the grid-based approach has the best performance when the number of cells is set to 20×20. In the following experiments, the number of cells is chosen to be 20×20 for the grid-based approach.

6.2.2 Performance with respect to Moving Probability



(a) Extended Efficient Sort-Based

(b) Grid-Based

Figure 6.2.2: Total number of messages with respect to moving probability

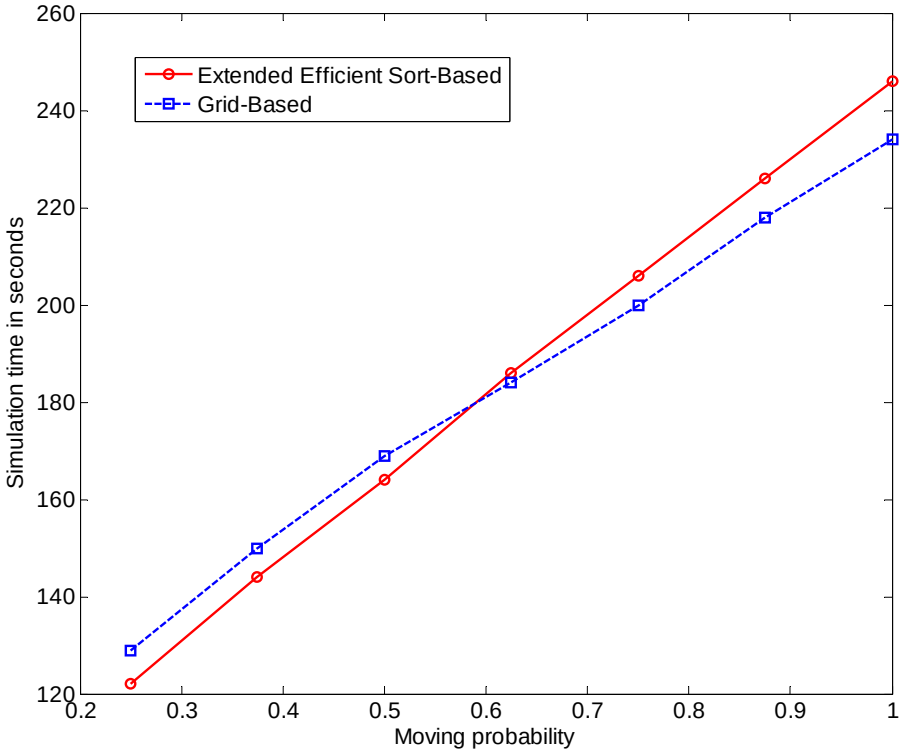
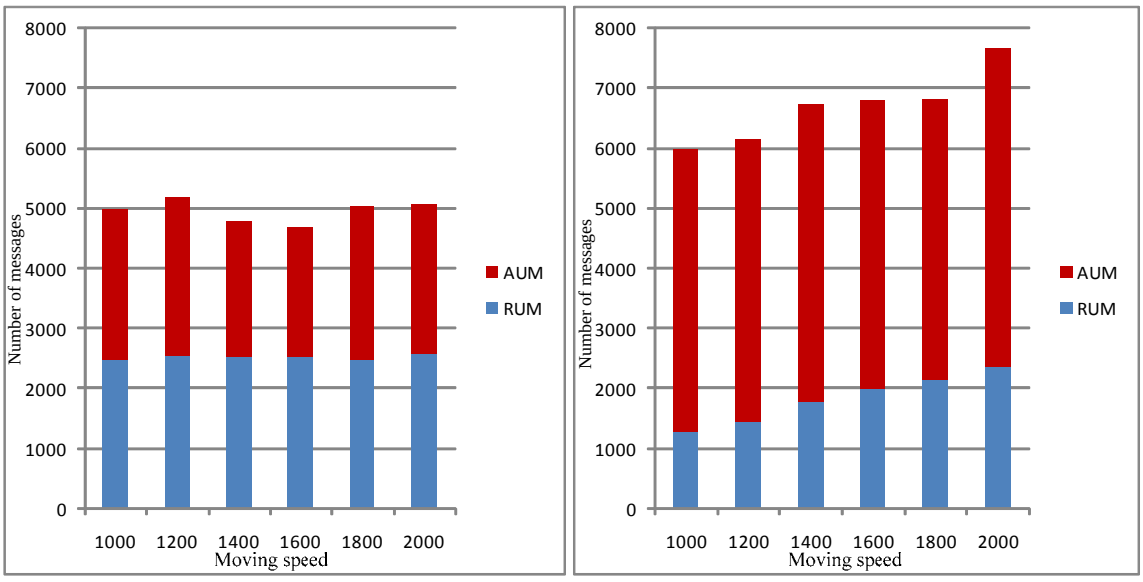


Figure 6.2.3: Simulation time with respect to moving probability

To analyze how the tank moving probability affects the performance of the two DDM implementations, we have increased it from 0.25 to 1 in increment steps of 0.125 based on the basic experiment setup in Subsection 6.2.1. The performance in terms of messages is shown in Figure 6.2.2 and the performance in terms of the simulation time is shown in Figure 6.2.3. For both approaches, the total number of messages and simulation time increase with an increasing moving probability mainly due to the increasing number of RUM exchanged. For the grid-based approach, an increase in the number of RUM messages increases the number of LU operations performed, which also has some effect on increasing the simulation time. However, the extended efficient sort-based approach is affected more by the increasing moving probability as it exchanges more RUM, so the grid-based approach begins to outperform the extended efficient sort-based approach when the moving probability is increased to 0.625 in both figures.

6.2.3 Performance with respect to Moving Speed

To analyze the performance of the two DDM implementations with respect to the tank moving speed, we have varied the moving speed from 1000 meters per minute to 2000 meters per minute based on the basic experiment setup in Subsection 6.2.1. Figure 6.2.4 shows the performance in terms of the number of messages and Figure 6.2.5 shows the performance in terms of the simulation time. As we can see, the number of RUM messages of the extended efficient sort-based approach does not vary much with an increasing moving speed as a RUM message always needs to be sent for each subscription region modification no matter how much the region moves. As the moving speed does not have much effect on the number of AUM messages, the performance of the extended efficient sort-based approach does not vary much. In contrast, the total number of messages of the grid-based approach increases, as an increasing moving speed increases the possibility that a tank movement changes its subscription region's overlapping grid cells and thus increases the number of RUM exchanged. For this reason and the fact that an increasing number of RUM messages also increases the number of LU operations, the simulation time of the grid-based approach increases with an increasing moving speed.



(a) Extended Efficient Sort-Based (b) Grid-Based

Figure 6.2.4: Total number of messages with respect to moving speed

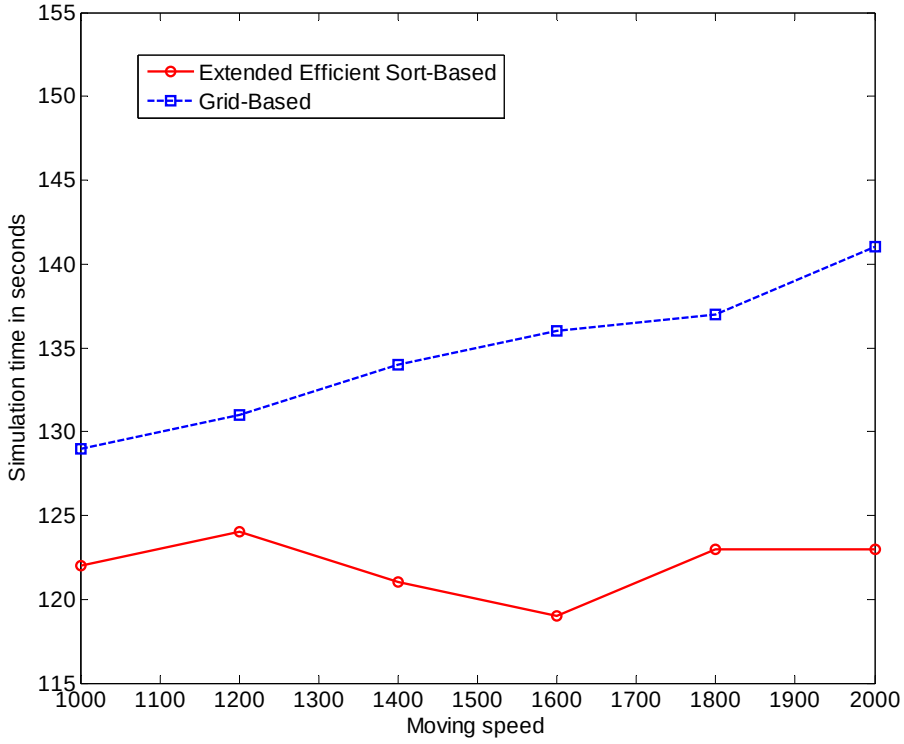


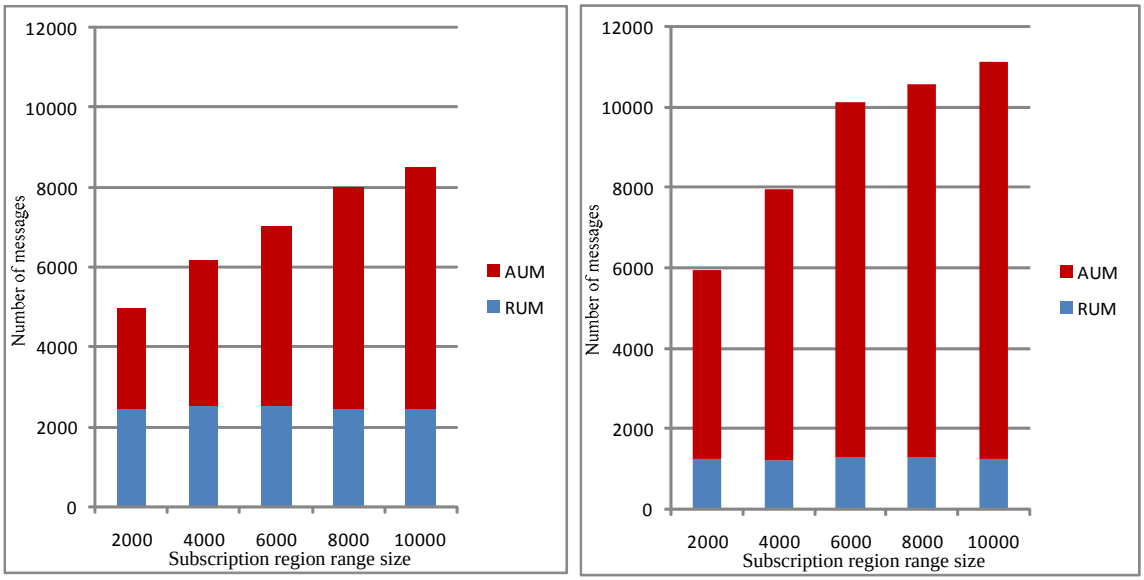
Figure 6.2.5: Simulation time with respect to moving speed

6.2.4 Performance with respect to Subscription Region Size

Based on the basic experiment setup in Subsection 6.2.1, we have also increased the subscription region range size from 2000 meters to 10000 meters in increment steps of 2000 meters. The performance in terms of the number of messages is shown in Figure 6.2.6 and the performance in terms of simulation time is shown in Figure 6.2.7. As we can see, for both approaches, the total number of messages and simulation time increase with an increasing subscription region range size mainly due to the increasing number of AUM sent. As the grid-based approach sends more AUM due to its inexact matching, its total number of messages is affected more by the subscription region size. The increasing subscription region range size also increases the number of LU operations performed by the grid-based approach. For these two reasons, the simulation time of the grid-based approach is affected more by the increasing subscription region range size.

Of the three parameters we have varied, the moving probability affects the simulation

time most. This is because, a tank movement involves several RTI service invocations, including the requests of setting the range bounds for each dimension and committing region modifications, and each request is a separate Grid service invocation in SOHR.



(a) Extended Efficient Sort-Based (b) Grid-Based

Figure 6.2.6: Total number of messages with respect to subscription region range size

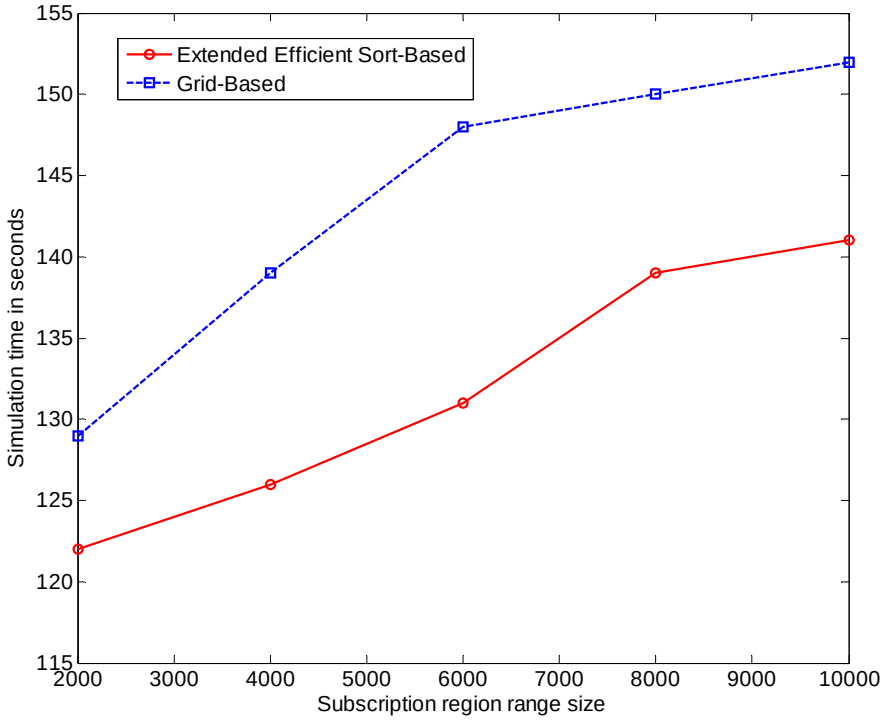


Figure 6.2.7: Simulation time with respect to subscription region range size

6.3 Summary

To demonstrate the plug-and-play paradigm of SOHR, this chapter discusses the distributed implementation of two DDM approaches, the grid-based approach and an extended efficient sort-based approach, in SOHR. Experiments have also been carried out to compare their performance in different scenarios. Results show that the extended efficient sort-based approach is more effective at reducing the number of AUM sent, but at the expense of extra RUM. If regions are modified frequently as in Subsection 6.2.2, the grid-based approach can give better performance with a well chosen number of cells. For different scenarios, the plug-and-play paradigm of SOHR always enables us to choose the approach with the better performance.

Chapter

7

Multi-User Gaming on the Grid using SOHR

To show the advantages of SOHR as claimed in Subsection 3.4, this chapter discusses a case study of SOHR to support multi-user gaming on the Grid. As discussed in Subsection 2.4, the HLA is promising at supporting multi-user gaming on the internet. However, this usually requires particular prior security setup (port opening) across administrative domains according to the specific RTI used. Since Grid service invocations, which can traverse firewalls, are used for communications in SOHR, this means SOHR is very suitable for multi-user gaming as users are able to join a game more conveniently on the Internet. This is illustrated with a multi-user maze game in this chapter. Specifically, an existing multi-user maze game [CAI02b, HEI99] has been modified to use SOHR as its communication infrastructure. Experiments have been carried out to analyze how DDM can be utilized to improve the communication efficiency of the game. As various DDM approaches exist, the paper also shows how the plug-and-play paradigm of SOHR enables us to choose the best approach for a specific scenario. The SOHR-based maze game has two versions, with and without TM usage. They are separately introduced in two subsections as follows.

7.1 SOHR-Based Multi-User Maze Game without TM

"You-Build-It" is an extensible multi-user virtual world engine which enables the creation of a new virtual world by changing properties files and map files [HEI99]. One

application is an interactive multi-user maze game where each player navigates in a maze and tries to collect as many cherry items as possible. Cherry items are generated dynamically at random positions. A player is represented as an avatar and has a 3D representation of its view in the maze. A player is able to discover nearby players within a viewing range. If a player goes close enough to another player, it may steal cherries held by the other player. Figure 7.1.1 shows the view of a player. The player sees two nearby other players and one cherry item in the view.

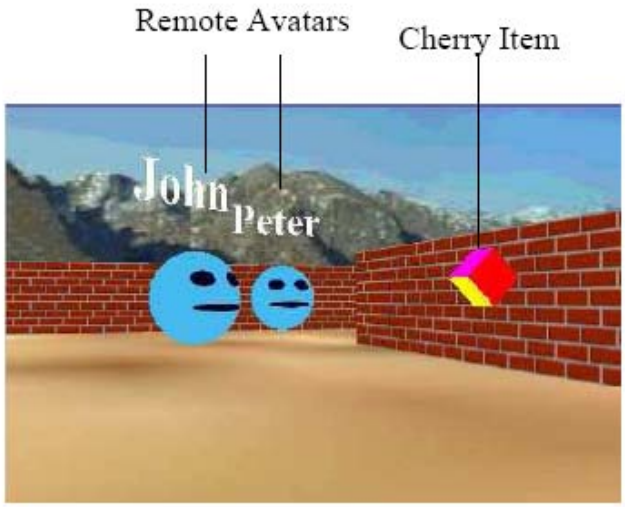


Figure 7.1.1: Game display

7.1.1 Modified Maze Game Architecture

The maze game is modified to use SOHR as its communication infrastructure in a fully distributed manner as shown in Figure 7.1.2. A player constitutes a federate and keeps a copy of the static maze environment sized 80×80 units. There is a cherry item server federate which is responsible for dynamically creating cherry items at random locations in the maze environment. Initially, we aim at producing a version of the game with the best responsiveness, so TM is not used here. This means that each federate in Figure 7.1.2 is unregulating and unconstrained.

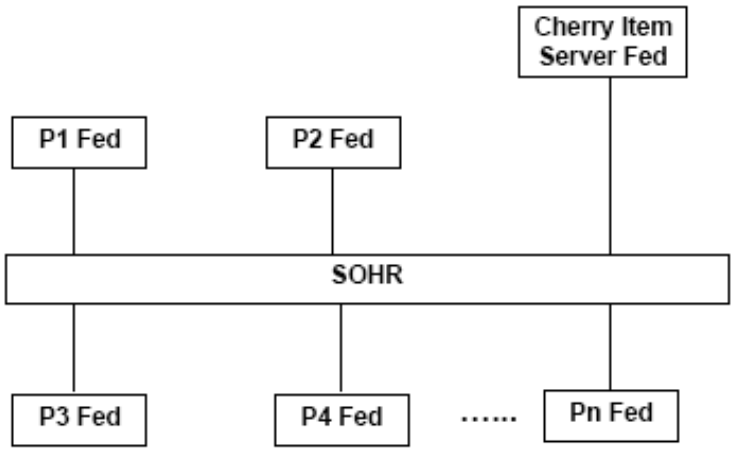


Figure 7.1.2: Modified maze game architecture

A player has a viewing range of 20×20 units with its center point being the player's current position. A player is able to see other players within its viewing range. Since the status updates between players are frequently exchanged, DDM can be used to improve the communication efficiency. The viewing range can then be used as the subscription region of a player and the update region can be simply set to an area of 1×1 unit at the current position of the player. Each player federate has a running thread that keeps updating its current status to other players as shown in Figure 7.1.3. In each iteration, the player commits region modifications if DDM is used and its regions are modified, and then sends its current status update, including its current position and orientation, to other players through an attribute values update. The execution time per iteration is a determining factor for the smoothness of the game and is used as a key performance indicator for our experiments in Subsection 7.1.3.

A Java KeyListener is used to detect the moving commands from the user. If no key is pressed, the avatar stays unchanged at the current position. If the left or the right arrow button is pressed, the avatar will turn left or right to a certain degree. If the up or down arrow button is pressed, the avatar will be moved forward or backward a distance of 0.05 units. To reduce the computational and communication overhead of DDM, a player federate commits region modifications in a space-driven manner with a distance threshold

of 1 distance unit. So, the probability of committing region modifications in an action step by a player federate is not high. In contrast, a player needs to send its status update in each action step. In addition to updating the status, the status update also acts as a heartbeat of a player to other players. When the player (A) goes into the viewing range of another player (B), A's first status update received by B causes B to add A to its list of remote avatars. To keep itself in B's list, A is required to keep sending its status update. When A goes outside of the viewing range of B, B has to detect this and then deletes A from its list. B realizes this by using another thread which keeps checking whether there is a status update from A in the last period of time.

```
run()
{
    while(true)
    {
        if(DDM is used && regions are modified)
            commitRegionModifications;
        sendStatusUpdates;
    }
}
```

Figure 7.1.3: Updating thread

The communications related to cherry items, such as cherry creation, collection and stealing, are through interactions. As there is competition for cherry collection between players, in that multiple players may try to collect the same cherry at the same time, the item server is responsible for this conflict resolution. A player tries to collect a cherry by sending a request to the item server. The item server sends a positive reply only when the cherry is available. Similar conflicts exist for cherry stealing between players and they are solved in a similar way using the request-and-reply mechanism. Interactions for cherry item collection and stealing are not exchanged so frequently compared with the attribute updates, so we do not use DDM for them for simplicity.

7.1.2 Auto Player

```
run()
{
    while(true)
    {
        if(a cherry is in viewing range)
        {
            while(the avatar is not facing the cherry)
            {
                turn a certain degree to the cherry;
                sleep(sleeptime);
            }
            while(the cherry is still avaialble)
            {
                move forward by 0.05;
                sleep(sleeptime);
            }
        }
        with a probability of 0.1
            while(turning is not finished)
            {
                turn a certain degree;
                sleep(sleeptime);
            }
        for(int i=0;i<=99;i++)
        {
            move forward by 0.05;
            sleep(sleeptime);
        }
    }
}
```

Figure 7.1.4: Auto-navigating thread

In order to make the game more interesting and allow for performance evaluation, we have implemented auto players which auto-navigate in the maze environment. It is realized through an auto-navigating thread as shown in Figure 7.1.4. In each iteration, a player checks whether there is a cherry in its viewing range. If there is, it tries to walk towards the cherry and collect it. After that, there is a probability of 0.1 for the player to turn left or right. Then the player keeps walking straight for 100 steps of 0.05 units. During the execution of the above loop iteration, the player may collide with maze walls

or other players. Whenever there is a collision, the player rotates for a certain degree to circumvent the collision. The walking and turning (rotating) are in tiny steps and a short sleep time is used for each tiny step. In Subsection 7.1.3, we will show how the sleep time affects the performance of our experiments.

7.1.3 Experiments and Results

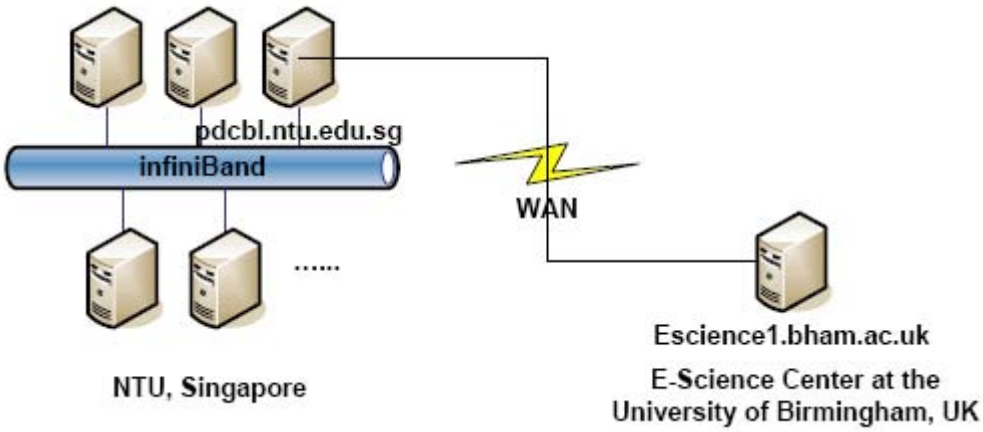


Figure 7.1.5: Experiment configuration

To demonstrate the SOHR-based maze game across administrative domains on the Grid and analyze its performance, experiments were carried out between a cluster at Nanyang Technological University (NTU), Singapore, and the E-Science Center at the University of Birmingham, UK, as shown in Figure 7.1.5. The cluster has a head node (pdcbl) and 15 subnodes connected by a 10Gb/s infiniBand connection and each node is installed with two dual core Xeon 3.0GHz CPUs, a 4GB RAM and a Redhat Enterprise Linux 4 OS. The only externally accessible node of the cluster is its head node (pdcbl). One machine was used on the UK side and it is installed with two Xeon 3GHz CPUs, 2GB RAM and Redhat Enterprise Linux AS 3 OS. The RTI Index Service, all management services and one LS were deployed at the head node of the cluster. Another LS was deployed at the machine on the UK side. 16 auto players were used in the experiments, namely P1-P16. P1-P15 were executed on separate sub nodes in the cluster while P16 was executed on the machine on the UK side. The Cherry Item Server federate was executed on the head node of the cluster. With the TCP port 8080 opened on both sides, the game was executed

successfully. As the TCP port 8080, known as the HTTP Alternative port, is generally open across administrative domains, users are able to join the SOHR-based game more conveniently than traditional games.

7.1.3.1 Performance with Sleep Time Equal to 50 Milliseconds

The experiments were carried out both without DDM and with different DDM approaches for the status updates between players. The DDM approaches used include the grid-based approach and the extended efficient sort-based approach implemented in SOHR as discussed in Chapter 6. As the performance of the grid-based approach depends on the size of a grid cell, we have also varied the number of grid cells of the grid-based approach from 2×2 to 80×80. The sleep time as introduced in Subsection 7.1.2 is fixed to be 50 milliseconds, which has been visually found to be a good value to allow human players to compete with auto players. The performance was measured for 1000 iterations of the updating loop in Figure 7.1.3 and the results are shown in Table 7.1.1.

Table 7.1.1: Performance results

DDM Approach	AAUM(P1-P16)	ARUM(P1-P15)	ATPI(S)(P1-P15)	RUM(P16)	ATPI(S)(P16)
Non DDM	15000	0	0.735	0	1.229
EESB	958	1231	0.148	2475	0.341
Grid-Based 2×2	6782	196	0.446	240	0.838
Grid-Based 4×4	2960	112	0.171	165	0.37
Grid-Based 8×8	1579	137	0.109	420	0.343
Grid-Based 16×16	1235	237	0.108	525	0.267
Grid-Based 40×40	1129	699	0.13	1965	0.471
Grid-Based 80×80	825	1760	0.202	3360	0.598

AAUM: Average Attribute Update Message sent out (A)RUM: (Average) Region Update Message sent out
ATPI(S): Average Time Per Iteration in Seconds

The communication efficiency of different approaches can be compared based on two factors, the number of Attribute Update Messages (AUM) sent and the number of Region Update Messages (RUM) sent. The communication efficiency directly affects the Average execution Time Per Iteration (ATPI) of the updating loop and in turn reflects the smoothness of the game. As only P16 was executed on the UK side and all of its message sending to other federates is across the WAN, on average, it needs more time to complete an iteration of the updating loop and thus its ATPI is listed in a separate column. With a

larger ATPI, P16 moves more in an iteration of the updating loop compared with other players and thus has larger probability to change its overlapping grid cells in the grid-based DDM approach. So it sends more RUM and its RUM results are listed in a separate column. Compared with the non DDM case, though the Extended Efficient Sort-Based approach (EESB) needs to send RUM whenever a region modification is committed, the Average number of AUM sent by a player (AAUM) is significantly reduced and thus the ATPI is also significantly reduced. Compared with the EESB approach, the grid-based approach only sends RUM when the overlapping grid cells change upon a region modification commitment, but it sends more AUM due to its inexact matching. A general property of the grid-based approach is observed that, with an increasing number of grid cells, the number of AUM is reduced and the number of RUM is generally increased [TAN00, RAK96, AYA00]. The only exception occurs when the number of grid cells equals to 2×2 where more RUM are exchanged than the 4×4 case. When the number of grid cells is changed from 4×4 to 2×2, there are two factors which affect the number of RUM exchanged. The first factor is the increasing size of a grid cell which reduces the number of RUM. The second factor is the increasing ATPI as a result of an increasing number of AUM exchanged. With a large ATPI, players move more per iteration and have a large probability of changing the overlapping grid cells, so they send more RUM. The two factors compete with each other and the second factor dominates in our scenario. When the number of grid cells is 16×16, the performance of the grid-based approach is optimized and better than the performance of the EESB approach. This is because the fast moving pace of the players causes too many RUM exchanged by the EESB approach. In Subsection 7.1.3.2, we will analyze how the moving pace affects the performance of both DDM approaches.

7.1.3.2 Performance with respect to Sleep Time

To vary the moving pace of the players, we have varied the sleep time from 50 milliseconds to 150 milliseconds in increasing steps of 25 milliseconds. For the grid-based approach, the number of grid cells is fixed to 16×16. The performance results in terms of

the number of messages are shown in Figure 7.1.6 and the performance results in terms of the execution time (ATPI) are shown in Figure 7.1.7. With an increasing sleeping time, the number of AUM exchanged does not vary for both DDM approaches while the number of RUM exchanged is reduced. As the grid-based approach exchanges much less RUM than AUM, the reduction of RUM does not affect its performance too much. So the ATPI results of the grid-based approach do not vary much with respect to the sleep time. As the EESB approach exchanges much more RUM than the grid-based approach, its reduction effect is more significant and thus makes it ATPI decrease. When the sleep time is increased up to 125 milliseconds, the EESB approach begins to outperform the grid-based approach as shown in Figure 7.1.7.

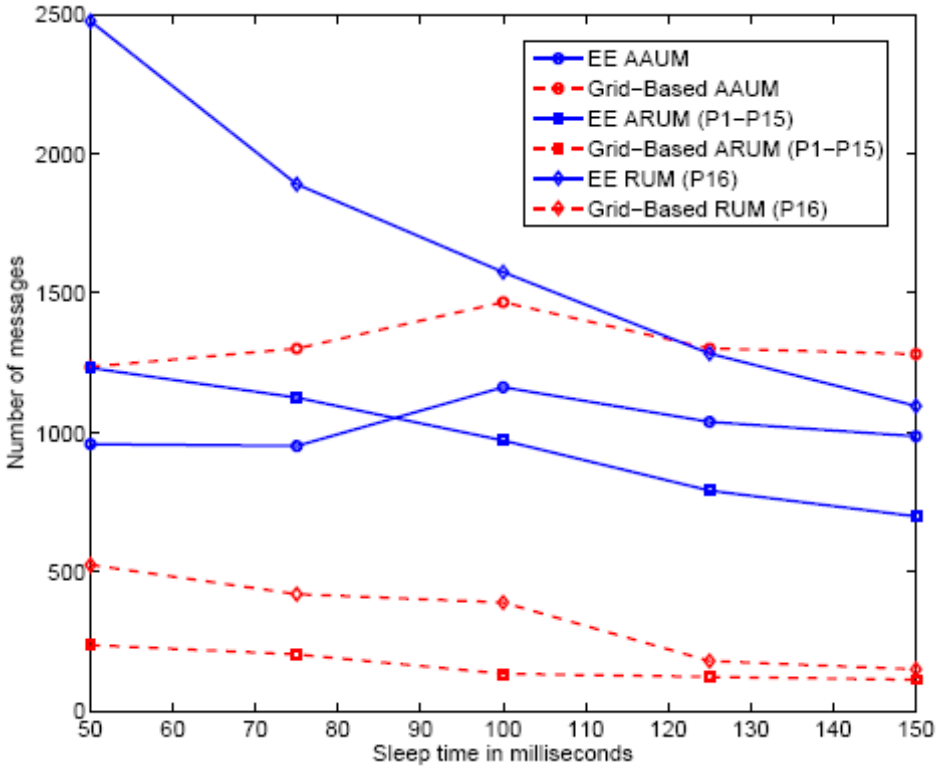


Figure 7.1.6: The number of messages with respect to the sleep time

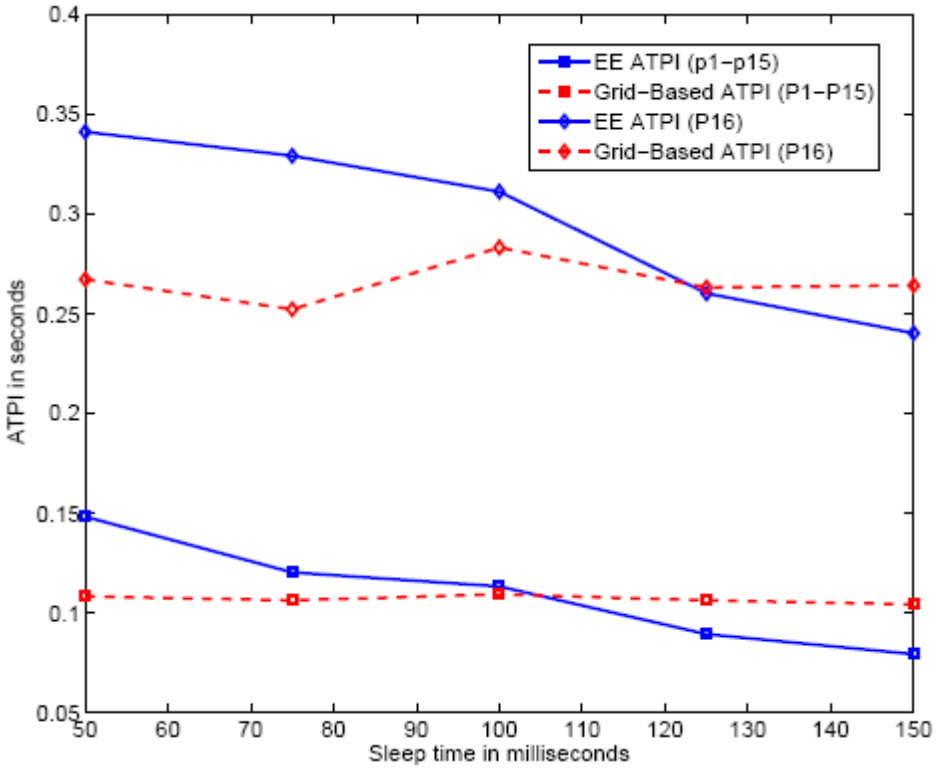


Figure 7.1.7: ATPI with respect to the sleep time

7.2 SOHR-Based Multi-User Maze Game with TM

The version of the game without TM introduced in Subsection 7.1 is not fair to all players as some players (e.g., P1-P15 in Subsection 7.1.3) may provide their status updates more frequently and experience less message latencies than other players (e.g., P16 in Subsection 7.1.3). This causes different responsiveness to be felt by different players, which is not fair. In order to provide a version of the game which is fair to all players, the HLA TM service group can be used to coordinate their behavior so that all players usually perform the same number of actions between synchronizations. Though the variation in responsiveness between different players will be reduced, the whole game responsiveness will be degraded as the use of TM introduces a synchronization overhead.

7.2.1 Modified Maze Game Architecture

The maze game architecture with TM is similar to the one without TM as shown in Figure 7.1.2. However, each federate is now both regulating and constrained. The lookahead of

each federate is set to be 5. All event messages, including the object attribute values update messages for player status updates and interaction messages for cherry creation, collection and stealing, become time-stamped. Figure 7.2.1 shows the running thread that a player federate keeps to update its current status to others. A player first performs ten action steps. Similar to the description in Subsection 7.1.1, a player moves with tiny steps controlled by the arrow keys. In each action step, the player can stay static or rotate for a certain degree or translate a certain distance. After ten action steps, the player federate requests its simulation time increase of 10 using the RTI service Time Advance Request (TAR). The request will later be granted with the RTI callback service Time Advance Grant (TAG). This is one iteration of the updating thread and it is continually repeated. The execution time per iteration is a determining factor for the smoothness of the game and is used as a key performance indicator for our experiments in Subsection 7.2.3. Similar to the player federates, the cherry item server federate repeatedly requests its simulation time increase using TAR to keep the simulation time of the whole federation increasing.

```
run()
{
        while(true)
        {
                tenActionSteps;
                if (DDM is used && regions are modified)
                        commitRegionModifications;
                sendTimeStampedStatusUpdates;
                TAR(logicalTime+10);
                inTimeAdvancingState=true;
                while(inTimeAdvancingState==true)
                        evokeMultipleCallbacks(0.01,0.02);

        }
}

void timeAdvanceGrant(LogicalTime theTime)
{
        inTimeAdvancingState=false;
        logicalTime=theTime;
}
```

Figure 7.2.1: Updating thread with TM

7.2.2 Auto Player

```
tenActionSteps()
{
        if(a cherry is in viewing range)
        {
                if(the avatar is not facing the cherry)
                        for(int i=0;i<10;i++)
                        {
                                turn a certain degree to the cherry;
                                sleep(sleeptime);
                                if(the avatar is facing the cherry)
                                        break;
                        }
                else
                        for(int i=0;i<10;i++)
                        {
                                move forward by movingdistance;
                                sleep(sleeptime);
                        }
        }
        else
        {
                if(random.nextFloat()<0.01)
                        for(int i=0;i<10;i++)
                        {
                                turn a certain degree;
                                sleep(sleeptime);
                        }
                else
                        for(int i=0;i<10;i++)
                        {
                                move forward by movingdistance;
                                sleep(sleeptime);
                        }
        }
}
```

Figure 7.2.2: Autonavigation

In order to make the game more interesting and allow for performance evaluation, we have implemented auto-navigating players as in the previous version of the game. The major task is to automate the “tenActionSteps” in Figure 7.2.1 and the solution is shown in Figure 7.2.2. A player checks whether there is a cherry in its viewing range. If there is, it tries to make ten steps of turning to the cherry until it faces the cherry or make ten steps of moving forward. If there is no cherry in its viewing range, it has 0.01 probability of

making ten steps of turning left or right and has 0.99 probability of making ten steps of moving forward. During the execution of the above autonavigation, the player may collide with maze walls or other players. Whenever there is a collision, the player rotates for a certain degree to circumvent the collision. The turning and moving forward are in tiny steps and a short sleep time is used for each tiny step. In each tiny step of moving forward, a parameter, namely movingdistance, determines how far the movement is. In Subsection 7.2.3, we will show how the parameter affects the performance of our experiments.

7.2.3 Experiments and Results

The same experiment configuration was used as the one in Subsection 7.1.3. However, all the player federates and the cherry item server federate are replaced with their respective TM versions. The experiment aims at measuring how the moving pace of the players affects the game performance in terms of the Average Time per Iteration (ATPI) of the updating thread loop in Figure 7.2.1. There are two ways of varying the moving pace of the players. The first one is to vary the sleep time in Figure 7.2.2 as we did in Subsection 7.1.3.2. However, for this TM version of the game, the auto-navigating code is embedded in the updating thread as shown in Figure 7.2.1, so the varying of the sleep time directly affects the ATPI of the updating thread loop. We decided not to vary the sleep time and keep it to be 50 milliseconds. The second way of varying the moving pace of the players is to vary the movingdistance parameter introduced in Figure 7.2.2. We have varied the movingdistance parameter from 0.01 unit to 0.05 unit in increasing steps of 0.01 unit. The experiment was carried out for 1000 iterations of the updating loop in Figure 7.2.1 both without DDM and with different DDM approaches for the status updates between players. The DDM approaches used include the grid-based approach and the extended efficient sort-based approach implemented in SOHR. For the grid-based approach, the number of grid cells is fixed to be 16×16. The performance results in terms of the average number of messages sent out by a player federate are shown in Figure 7.2.3 and the performance results in terms of the execution time (ATPI) of a player federate are shown in Figure 7.2.4.

As shown in Figure 7.2.3, for all approaches with and without DDM, the Average number of Time Information Messages (ATIM)[1] sent out by a player federate is constant at 16000 irrespective of the moving distance as there are 1000 iterations and 17 federates including both player federates and the cherry item server federate. Both the Extended Efficient Sort-Based DDM approach (EESB) and the grid-based DDM approach greatly reduce the average number of AUM sent out by a player federate. The reduction effect of the EESB approach is more significant due to its exact matching. The moving distance does not have an effect on the number of AUM for all approaches. In contrast, the average number of RUM sent out by a player federate increases as the moving distance increases for both the EESB and the grid-based approaches. As the EESB approach exchanges more RUM than the grid-based approach, its increasing effect is much more significant.

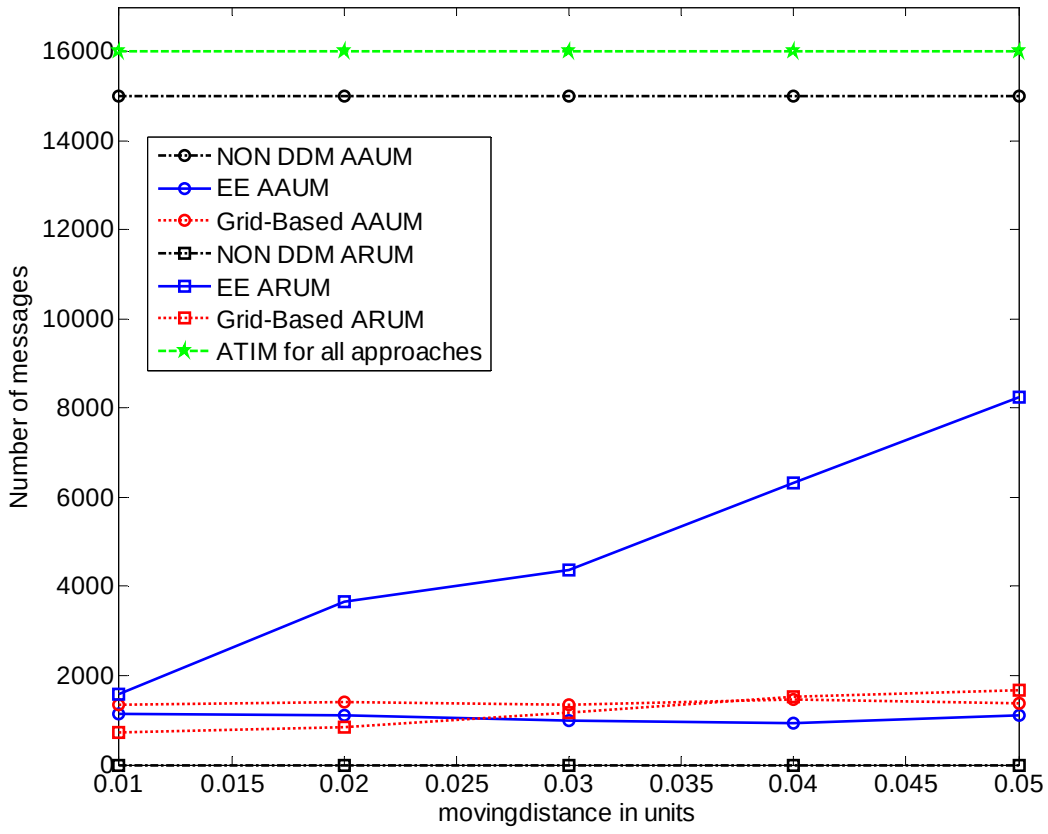


Figure 7.2.3: The number of messages with respect to the moving distance

[1] In this scenario, the Time Information Message (TIM) is in the form of UI as TAR is used for requesting time advancement.

In contrast to the version of the game without TM discussed in Subsection 7.1.3, the use of DDM has little improvement on the execution time (ATPI) of the game as shown in Figure 7.2.4. This is because of the TM synchronization requirement. As shown in Figure 7.2.1, in each iteration of the updating thread, a player federate may send out an RUM if DDM is used, sends out an AUM, sends out a TIM and then waits for the time advancement grant. As the time advancement grant can be delayed by any federate in the game, the execution time of one iteration is determined by the slowest federate. In the non DDM approach, the slowest federate just sends two messages, an AUM and a TIM. In the EESB or the grid-based approach, the slowest federate may send two messages (AUM and TIM) or three messages (RUM, AUM and TIM). This means the use of DDM does not improve the ATPI of the version of the game with TM. Moreover, the increase of the moving distance makes the ATPI of the EESB approach much larger than the ATPI of the non DDM approach. This is because the increase of the moving distance significantly increases the number of RUM of the EESB approach as shown in Figure 7.2.3. Thus, the probability that the slowest federate sends three messages in one iteration is thus significantly increased.

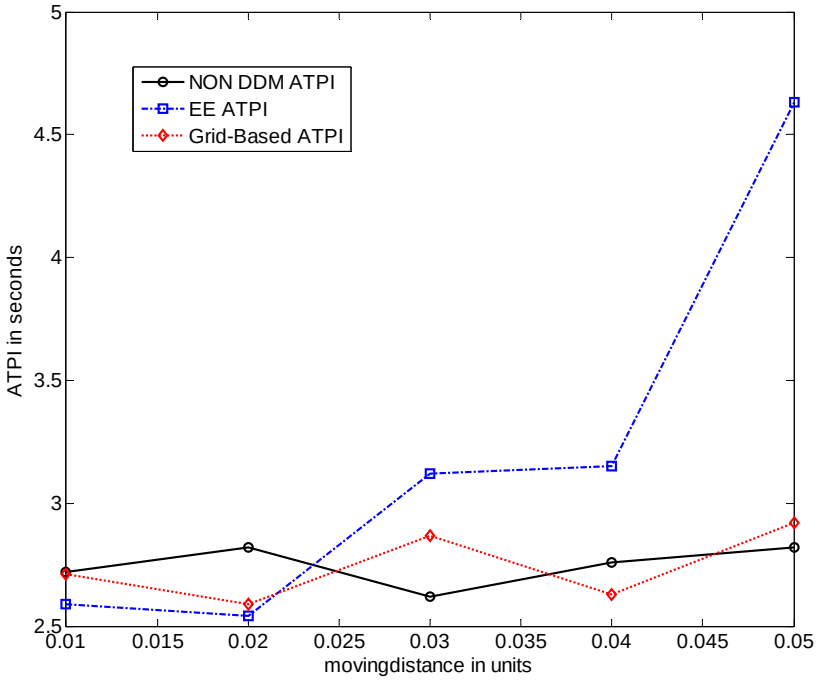


Figure 7.2.4: ATPI with respect to the moving distance

7.3 Summary

To show the advantages of SOHR as claimed in Subsection 3.4, this chapter has discussed a case study of SOHR to support multi-user gaming on the Grid. The SOHR-based maze game has been developed in two versions, with and without TM. Without any prior security setup, both of them have been successfully executed on the WAN between Singapore and UK through the HTTP Alternative TCP port 8080. This means that users are able to join a game conveniently on the Internet.

Experiments have shown that the HLA DDM service group is able to significantly improve the performance of the version of the game without TM. The EESB and the grid-based DDM approaches have been compared with regard to the performance improvement. The results show that when players move at a slow pace, the EESB approach is more efficient due to its exact matching; when players move at a fast pace, the overhead of the EESB approach in exchanging more region updates becomes prominent so that the grid-based approach is more efficient. This generally means the EESB approach is more suitable for slow-paced games while the grid-based approach is more suitable for fast-paced games. For different scenarios, the plug-and-play paradigm of SOHR always enables us to choose the approach with the better performance. However, neither the EESB nor the grid-based approach is able to improve the performance of the version of the game with TM due to the TM synchronization requirement.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Distributed simulation was developed to promote the interoperability and reusability of simulation applications and to link geographically dispersed simulation components [FUJ00]. The High Level Architecture (HLA) has been approved as the IEEE standard for distributed simulation in September 2000 [IEEE1516, IEEE1516.1, IEEE1516.2]. While the HLA defines the rules [IEEE1516], interface specification [IEEE1516.1] and Object Model Template (OMT) [IEEE1516.2], the Run Time Infrastructure (RTI) provides the actual implementation of the HLA standard.

Grid computing was proposed for distributed resource sharing among dynamic collections of individuals, institutions, and virtual organizations [FOS01b]. Within a Service Oriented Architecture (SOA), the Open Grid Services Architecture (OGSA) defines a set of core services to provide the capabilities and behaviors that address key concerns in Grid Systems [FOS01a, FOS05]. Among Grid middlewares, Globus Toolkit (GT) [GT], which provides an implementation of the OGSA, is the de facto standard middleware for Grid computing.

This project is driven by the demand to create a service oriented distributed simulation framework which provides the basic functionalities of a distributed simulation middleware as Grid services. To follow the standards of both distributed simulation and Grid computing, this project has developed a Service Oriented HLA RTI (SOHR) framework which implements an HLA RTI using Grid services based on GT4. The detailed design of SOHR is described in Chapter 3 with its major benefits summarized in Subsection 3.4. A prototype of SOHR has been implemented with a subset of the HLA specification and latency benchmark tests have been carried out both on the LAN and on the WAN. The experiment results show that SOHR's performance on GT4.1.1 is much better than its performance on GT4.0.2 due to the persistent TCP/IP connection support by GT4.1.1. SOHR's performance on GT4.1.1 is comparable with the performance of the DMSO RTI especially on the WAN.

As SOHR creates a plug-and-play paradigm for an RTI implementation, an algorithm for an HLA service group can be implemented and plugged into SOHR easily. This project therefore also aims to develop efficient algorithms for the HLA TM and DDM service groups and incorporate them into SOHR.

The HLA TM service group ensures a TSO message delivery sequence and correct time advancement of each federate in a federation. Traditional TM algorithms can be either synchronous or asynchronous. In general, synchronous algorithms are based on conditional information while asynchronous algorithms are based on unconditional information. However, synchronous algorithms are not suitable for federations with low time-resolution federates which send conditional information with a low frequency; asynchronous algorithms have the "time creep" problem when the lookahead is small. To resolve the drawbacks of each algorithm by taking the advantages of the other, Chapter 4 combines the two algorithms together to give a hybrid algorithm based on both conditional and unconditional information. The idea is described using HLA-specific terminology, but it applies to general parallel and distributed simulation. With the plug-and-play paradigm of SOHR, we have implemented the three TM algorithms into SOHR. Both experimental

results on the LAN and on the WAN show that the hybrid algorithm effectively benefits from the advantages of both synchronous and asynchronous algorithms and has the best performance among the three algorithms in a federation with a variable-processing-time federate. Since processing time of federates is normally uncertain especially in a Grid environment, the hybrid algorithm is a good choice for our Grid-based SOHR framework.

Chapter 4 also presents a comparison between the hybrid algorithm and FDK's TM algorithm. FDK's TM algorithm has been implemented into the SOHR framework in a similar way to the other three TM algorithms. An analysis of the algorithm and experimental results show that FDK's TM algorithm completely prevents the “time creep” problem, which may still occur in the hybrid algorithm. However, due to redundant GALT calculations of FDK's TM algorithm, it is generally less scalable than the hybrid algorithm with respect to the total number of federates in a federation.

As communication efficiency is one of the major concerns in this project because of the high overheads of Grid service invocations, another focus is on the HLA DDM service group. This has led to the development of an efficient sort-based DDM matching algorithm which is described in Chapter 5. According to the theoretical analysis, both the storage complexity and the static matching performance of the efficient sort-based algorithm are dependant on the maxRS/D_{UB} ratio while its dynamic matching performance is dependant on both the maxRS/D_{UB} and maxBS/D_{UB} ratios. In a large spatial environment, the efficient sort-based algorithm should have good storage and computational scalability because the two ratios are normally very small.

Next, Chapter 5 extends the efficient sort-based algorithm to one which incrementally matches for dynamic region modifications. Theoretical analysis concludes that, the extended efficient sort-based algorithm needs more storage space and more static matching time than the efficient sort-based algorithm. However, its dynamic matching computational performance is only dependant on the maxBS/D_{UB} ratio and thus is generally more efficient

than the dynamic matching of the efficient sort-based algorithm. The extended efficient sort-based algorithm should be a good choice for dynamic matching in an application with a small maxBS/D_{UB} ratio without regard to the maxRS/D_{UB} ratio.

Experiments were carried out to test the performance of the two proposed DDM matching algorithms with respect to individual factors, namely the number of regions, the maxRS/D_{UB} ratio and the maxBS/D_{UB} ratio. Their experimental results are compared with the results of the region-based and Raczy's sort-based algorithms. All the experimental results agree with the theoretical conclusions.

With the plug-and-play paradigm of SOHR, two DDM approaches, the grid-based approach and the extended efficient sort-based approach, have been implemented in SOHR as discussed in Chapter 6. Experiments have been carried out to compare their performance in different scenarios. Results show that the extended efficient sort-based approach is more effective at reducing the number of AUM sent, but at the expense of extra RUM. If regions are modified frequently as in Subsection 6.2.2, the grid-based approach can give better performance with a well chosen number of grid cells.

To show the advantages of SOHR as claimed in Subsection 3.4, Chapter 7 presents a case study of SOHR to support multi-user gaming on the Grid. Due to the reuse and interoperability supported by the HLA and communication optimization supported by the HLA DDM service group, the HLA is promising at supporting multi-user gaming on the internet. However, this usually requires particular prior security setup (port opening) across administrative domains according to the specific RTI used. With Grid service invocations, which can traverse firewalls, used for its communication, SOHR is very suitable for multi-user gaming as users are able to join a game more conveniently on the internet. This has been illustrated with the integration of an existing multi-user interactive maze game with SOHR. The SOHR-based maze game has been developed in two versions, with and without TM. Without any prior security setup, both of them have been successfully

executed on the WAN between Singapore and UK through the HTTP Alternative TCP port 8080.

Experiments have shown that the HLA DDM service group is able to significantly improve the performance of the game version without TM. The extended efficient sort-based (EESB) and the grid-based DDM approaches have been compared for their performance improvement. The results show that when players move at a slow pace, the EESB approach is more efficient due to its exact matching; when players move at a fast pace, the overhead of the EESB approach in exchanging more region information becomes prominent so that the grid-based approach is more efficient. For different scenarios, the plug-and-play paradigm of SOHR always enables us to choose the approach with the better performance. However, neither the EESB nor the grid-based approach is able to improve the performance of the version of the game with TM due to the TM synchronization requirement.

Thus, all the four objectives of this project as defined in Subsection 1.3 have been accomplished.

8.2 Future Work

As discussed in Subsection 3.2.2, the LS design of the SOHR framework is not fully service-oriented as the different modules of the LRI structure are implemented as separate Java classes and bundled together. This design decision was made for performance reasons because of the close communication between the different modules.

To create a fully Service-Oriented Architecture (SOA) for the SOHR framework, we need to decompose the LRI structure so that its different modules are implemented in different Grid services as shown in Figure 8.2.1. The whole SOHR architecture can be divided into three levels. The lowest level is the system level and it comprises a single RTI Index Service. The middle level is the federation level and it comprises the various management services for different HLA service groups. Relying on the WSRF framework, a federation

creates a single resource instance for each type of the management services. This is consistent with the management services we have discussed in Subsection 3.1. The top level is the federate level and it comprises different module services and LSs. Relying on the WSRF framework, a federate creates a single resource instance of LS (LRI) for communication purposes and the creation of an LRI then causes the creation of a single resource instance for each type of module service.

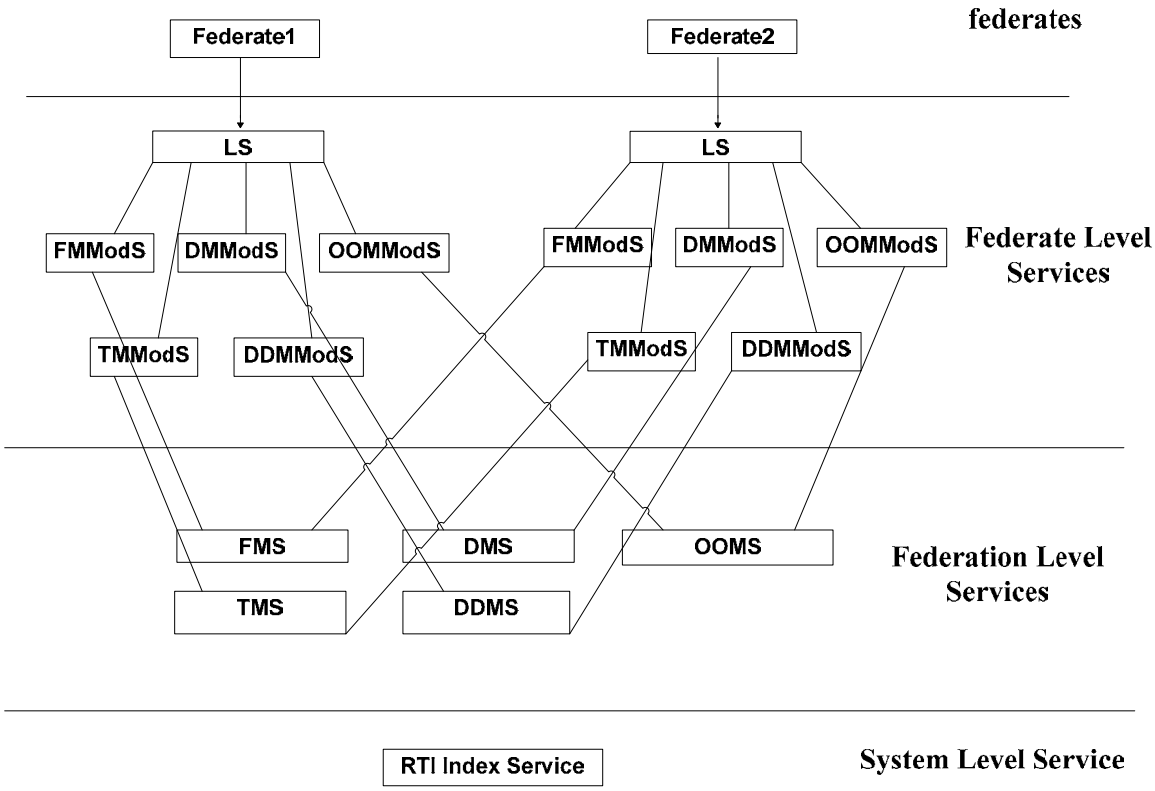


Figure 8.2.1: A fully SOA for the SOHR framework

The resource instance of a particular module service assumes the same functionalities as the corresponding particular LRI module in the original SOHR framework. It communicates with an underlying management service and its peers of other federates through Grid service invocations to provide the functionalities of the corresponding HLA service group (e.g., interactions are sent from OOMModS to OOMModS). An LRI simply passes the request made by its federate to the corresponding module service for processing.

Because the communications between the module services for a federate are frequent and the multi-hop feature of the new architecture inevitably increases the messaging latency (e.g. object attribute updates and interactions), its performance will be worse than the original architecture. However, the new architecture is fully service-oriented so that it provides better support for our plug-and-play paradigm. For example, a new DDM approach can be plugged into the new SOHR architecture simply by implementing and deploying a new DDMModS and DDMS based on the new approach. With the registration service provided by the RTI Index Service, the LRI of a federate can discover them and connect to them when the new approach is chosen for use in a federation. This is more convenient than the way a new module is generated in the original SOHR framework as described in Subsection 3.2.2.

As future work, the new fully SOA for SOHR will be investigated, implemented and evaluated. It will be compared with the original SOHR architecture in terms of both performance as well as convenience of the plug-and-play paradigm in real applications. Efforts will also be made on improving its performance.

Appendix A: Author's Publications

Journal Paper:

[1] **K. Pan**, S. J. Turner, W. Cai and Z. Li, "A Hybrid HLA Time Management Algorithm Based on Both Conditional and Unconditional Information". *Simulation*, Vol. 85, No. 9, 2009, pp. 559-573.

Book Chapter:

[1] **K. Pan**, S. J. Turner, W. Cai and Z. Li, "Design and Performance Evaluation of a Service-Oriented HLA RTI on the Grid". Grid Computing: Infrastructure, Service, and Applications, Chapter 18, Eds. L. Wang, W. Jie and J. Chen, ISBN: 978-1420067668, CRC Press, April 2009.

Conference Paper:

[1] **K. Pan**, S. J. Turner, W. Cai and Z. Li, "An Efficient Sort-Based DDM Matching Algorithm for HLA Applications with a Large Spatial Environment". In *Proceedings of the 21st ACM/IEEE/SCS International Workshop on Principles of Advanced and Distributed Simulation*, June 2007, pp. 70-82.

[2] **K. Pan**, S. J. Turner, W. Cai and Z. Li, "A Service Oriented HLA RTI on the Grid". In *Proceedings of the 5th IEEE International Conference on Web Services*, July 2007, pp. 984-992.

[3] **K. Pan,** S. J. Turner, W. Cai and Z. Li, "A Hybrid HLA Time Management Algorithm Based on Both Conditional and Unconditional information". In *Proceedings of 22nd*

ACM/IEEE/SCS Interational Workshop on Principles of Advanced and Distributed Simulation, 2008, pp. 203-211. **(Best Paper Candidate)**

[4] **K. Pan,** S. J. Turner, W. Cai and Z. Li, "Implementation of Data Distribution Management Services in a Service Oriented HLA RTI". In *Proceedings of Winter Simulation Conference* 2009, December 2009, pp. 203-215.

[5] **K. Pan,** S. J. Turner, W. Cai and Z. Li, "Multi-User Gaming on the Grid using a Service Oriented HLA RTI". In *Proceedings of the 13th ACM/IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 2009, pp. 48-56. **(Best Paper Award)**

[6] Z. Li, W. Cai, S. J. Turner and **K. Pan**, "Federate Migration in a Service Oriented HLA RTI". In *Proceedings of the 11th ACM/IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 2007, pp. 113-121.

[7] Z. Li, W. Cai, S. J. Turner and **K. Pan**, "Improving Performance by Replicating Simulations with Alternative Synchronization Approaches". In *Proceedings of Winter Simulation Conference 2008*, December 2008, pp. 1112-1120.

[8] S. J. E. Taylor, N. Mustafee, S. J. Turner, **K. Pan** and S. Strassburger**,** "Commercial-Off-The-Shelf Simulation Package Interoperability: Issues and Futures". In *Proceedings of Winter Simulation Conference 2009*, December 2009, pp. 1027-1038.

Bibliography

[ALSP94] A. L. Wilson and R. M. Weatherly, "The Aggregate Level Simulation Protocol: an Evolving System". In *Proceedings of the 1994 Winter Simulation Conference*, December 1994, pp. 781-787.

[AYA89] R. Ayani, "A Parallel Simulation Scheme Based on the Distance between Objects". In *Proceedings of SCS Multiple Conference on Distributed Simulation*, 1989, pp. 113-118.

[AYA00] R. Ayani, F. Moradi and G. Tan, "Optimizing Cell-size in Grid-Based DDM". In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*, 2000, pp. 93-100.

[BAS98] M. Bassiouni, M. H. Chiu, M. Loper and M. Garnsey, "Relevance Filtering for Distributed Interactive Simulation". *Computer Systems Science and Engineering*, Vol. 13, No. 1, 1998, pp. 39-47.

[BAU01] N. E. Baughman and B. N. Levine, "Cheat-Proof Playout for Centralized and Distributed Online Games". In *Proceedings of the INFOCOM 2001*, 2001, pp. 104-113.

[BET01] P. Bettnet and M. Terrano, "GDC2001: 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond". *Technical report, Ensemble Studios*, 2001.

[BHA08] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan and X. Zhuang, "Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games". In *Proceedings of ACM SIGCOMM 2008*, 2008, pp. 389-400.

[BOU03] A. Boukerche and C. Dzermajko, "Alternative Approaches to Data Distribution Management in Large-Scale Distributed Simulation

Systems". In *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, 2003, pp. 555-562.

[BOU05] A. Boukerche, N. J. Mcgraw and R. B. Araujo, "A Grid-Filtered Region-based Approach to Support Synchronization in Large-Scale Distributed Interactive Virtual Environments". In *Proceedings of the 2005 International Conference on Parallel Processing Workshops*, 2005, pp. 525-530.

[BRY77] R. E. Bryant, "Simulation of Packet Communication Architecture Computer Systems". Master's thesis, Massachusetts Institute of Technology, 1977.

[BUTFLY] Butterfly.net. Inc. "The Butterfly Grid: A Distributed Platform for Online Games". Available via http://www.butterfly.net/platform/.

[CAI02a] W. Cai, S. J. Turner and H. Zhao, "A Load Management System for Running HLA-based Distributed Simulations over the Grid". In *Proceedings of the 6th IEEE International Workshop on Distributed Simulation and Real Time Applications*, October 2002, pp. 7-14.

[CAI02b] W. Cai, P. Xavier, S. J. Turner and B. Lee, "A Scalable Architecture for Supporting Interactive Games on the Internet". In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*, 2002, pp. 60-67.

[CAI05] W. Cai, Z. Yuan, M. Y. H. Low and S. J. Turner, "Federate Migration in HLA-based Simulation". *Future Generation Computer Systems*, Vol. 21, No. 1, January 2005, pp. 87-95.

[CAR97] C. D. Carothers, R. M. Fujimoto, R. M. Weatherly and A. L. Wilson, "Design and Implementation of HLA Time Management In the RTI Version F.0". In *Proceedings of the 1997 Winter Simulation*

Conference, December 1997, pp. 373-380.

[CHA79] K. M. Chandy and J. Misra. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs". *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 5, 1979, pp. 440-452.

[CHA89] K. M. Chandy and R. Sherman, "The Conditional Event Approach to Distributed Simulation". In *Proceedings of SCS Multiple Conference on Distributed Simulation*, 1989, pp. 93-99.

[CHA06] x. Chai, H. Yu, Z. Du, B. Hou and B. Li, "Research and Application on Service Oriented Infrastructure for Networkitized M&S". In Proceedings of *the 6th IEEE International Symposium on Cluster Computing and the Grid, International Workshop on Distributed Simulation on the Grid*, 2006.

[CHE06] X. Chen, W. Cai, S. J. Turner and Y. Wang, "SOAr-DSGrid: Service-Oriented Architecture for Distributed Simulation on the Grid". In *Proceedings of the 20th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*, 2006, pp. 65-73.

[CHE08] D. Chen, G. K. Theodoropoulos, S. J. Turner, W. Cai, R. Minson and Y. Zhang, "Large Scale Agent-Based Simulation on the Grid". *Future Generation Computer Systems*, Vol. 24, Issue 7, 2008, pp. 658-671.

[CHO05] K. Choi, T. Lee and C. Jeong, "RTI Execution Environment Using Open Grid Service Architecture". In *Proceedings of the 5th International Conference on Computational Science*, 2005, pp. 866-869.

[COH95] J. D. Cohen, M. C. Lin, D. Manocha and M. K. Ponamgi, "I-Collide: an Interactive and Exact Collision Detection System for Large-Scale

Environments". In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, 1995, pp. 189-196.

[CS] Valve Software. Counter Strike. Available via http://www.counter-strike.com/.

[DIO99] C. Diot and L. Gautier, "A Distributed Architecture for Multiplayer Interactive Applications on the Internet". *IEEE Networks Magazine*, 1999, 13(4):6-15..

[DIS95] IEEE 1278.1, "IEEE Standard for Distributed Interaction Simulation – Application Protocols". 1995.

[DMSO] Defense Modeling and Simulation Office (DMSO), "High Level Architecture RTI 1.3NG Programmer's Guide". Version 6, 2002.

[FAU98] R. Faucher, "Integrating HLA into the Spearhead Game". *MAK Technologies INC Cambridge MA*, report number: A220253, 1998.

[FDK] Georgia Tech. Federated Simulations Development Kit (FDK). Available via http://www.cc.gatech.edu/computing/pads/fdk.html.

[FIT04] J. B. Fitzgibbons, R. M. Fujimoto, D. Fellig, S. D. Kleban and A. J. Scholand, "IDSIM: An Extensible Framework for Interoperable Distributed Simulation". In *Proceedings of the IEEE International Conference on Web Services*, July 2004, pp. 532-539.

[FOS01a] I. Foster, C. Kesselman, J. Nick and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration". *Open Grid Services Infrastructure WG, Global Grid Forum*, June 2002. Available via http://www.globus.org/alliance/publications/papers/ogsa.pdf.

[FOS01b] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". *International Journal of High Performance Computing Applications*, Vol. 15, No. 3, August

2001, pp. 200-222.

[FOS05] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell and J. Von Reich, "The Open Grid Services Architecture, Version 1.0". *Open Grid Services Architecture WG, Global Grid Forum*, January 2005. Available via http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf.

[FOX05] G. Fox, A. Ho, S. Pallickara, M. Pierce and W. Wu, "Grids for the GiG and Real Time Simulations". In *Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real Time Applications*, October 2005, pp. 129-138.

[FUJ88] R. M. Fujimoto, "Lookahead in Parallel Discrete Event Simulation". In *Proceedings of the International Conference on Parallel Processing*, Vol. 3, 1988, pp. 34-41.

[FUJ98] R. M. Fujimoto, "Time Management in the High Level Architecture". *Simulation Special Issue on High Level Architecture*, Vol. 71, No. 6, 1998, pp. 388-400.

[FUJ00] R. M. Fujimoto, "Parallel and Distributed Simulation Systems". *Wiley InterScience*, January 2000.

[FUJ00b] R. M. Fujimoto, T. Mclean, K. Perumalla and I. Tacic, "Design of High Performance RTI Software". In *Proceedings of International Workshop on Distributed Simulation and Real-Time Applications*, 2000, pp. 89-96.

[GT] Globus Toolkit. Available via http://www.globus.org/.

[HEI99] C. Heistad and S. Pietrowicz, You-Build-It Virtual Reality. *NCSA Java3D Group, University of Illinois Urbana-Champaign*, 1999.

[HYE02] M. Hyett and R. Wuerfel, "Implementation of the Data Distribution Management Services in the RTI-NG". In *Proceedings of the 2002 Spring Simulation Interoperability Workshop*, March 2002, paper no. 02S-SIW-044.

[IEEE1516] IEEE 1516, "Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules". September 2000.

[IEEE1516.1] IEEE 1516.1, "Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification". September 2000.

[IEEE1516.2] IEEE 1516.2, "Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – HLA Object Model Template (OMT)". September 2000.

[IEEE1516.3] IEEE 1516.3, "High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)". March 2003.

[J2SE] J2SE 1.5.0 Tool Documentations For Windows. Available via http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/java.html.

[KNU04] B. Knutsson, H. Lu, W. Xu and B. Hopkins, "Peer-to-Peer Support for Massively Multiplayer Games". In *Proceedings of the INFOCOM 2004*, pp. 96-107.

[KUM05] P. Kumar, Q. Mehdi and N. Gough, "High Level Architecture for Distributed Agent Simulation in Computer Games". In *Proceedings of the 6th International Conference on Computer Games: Artificial Intelligence and Mobile Systems*, 2005, pp. 54-58.

[LEE06] M. Lees, B. Logan and G. K. Theodorpoulos, "Agents, Games and HLA". *Simulation Modeling Practice and Theory*, Vol. 14, Issue 6, 2006, pp 752-767.

[LI07] Z. Li, W. Cai, S. J. Turner and K. Pan, "Federate Migration in a Service Oriented HLA RTI". In *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 2007, pp. 113-121.

[LI08] Z. Li, W. Cai, S. J. Turner and K. Pan, "Improving Performance by Replicating Simulations with Alternative Synchronization Appraoches". In *Proceedings of Winter Simulation Conference 2008*, 2008, pp. 1112-1120.

[LIU05] E. S. Liu, M. K. Yip and G. Yu, "Scalable Interest Management for Multidimensional Routing Space". In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, 2005, pp. 82-85.

[LIU06] E. S. Liu, M. K. Yip and G. Yu, "Lucid Platform: Applying HLA DDM to Multiplayer Online Game Middelware". *ACM Computers in Entertainment*, Vol. 4, Issue 4, 2006.

[LIU09] E. S. Liu and G. K. Theodoropoulos, "An Approach for Parallel Interest Matching in Distributed Virtual Environments". In *Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, 2009, pp. 57-65.

[LUB89] B. D. Lubachevsky, "Efficient Distributed Event-Driven Simulations of Multiple-loop Networks". *Communication of the ACM*, Vol 32, Issue 1, 1989, pp. 111-123.

[LUCID] HKPU. Lucid Platform. Available via http://www.lucidplatform.com.

[MIN04] R. Minson and G. K. Theodoropooulos, "Distributing Repast Agent-Based Simulations with HLA". *In Proceedings of 2004 European Simulation Interoperability Workshop*, 2004, paper no.

04E-SIW-046.

[MOL06] B. Moller, "A First Look at the HLA Evolved Web Service API". In *Proceedings of the IEEE 2006 European Simulation Interoperability Workshop*, June 2006, paper no. 06E-SIW-061.

[MOL07] B. Mollder, B. Lofstrand and M. Karlsson, "An Overview of the HLA Evolved Modular FOMs". In *Proceedings of the 2007 Spring Simulation Interoperability Workshop*, 2007, paper no. 07S-SIW-108.

[MOR97] K. L. Morse and J. S. Steinman, "Data Distribution Management in the HLA Multidimensional Regions and Physically Correct Filtering". In *Proceedings of the 1997 Spring Simulation Interoperability Workshop*, March 1997, paper no. 97S-SIW-052.

[MOR01] K. L. Morse and M. D. Petty, "Data Distribution Management Migration from DoD 1.3 to IEEE 1516". In *Proceedings of the 5th IEEE International Workshop on Distributed Simulation and Real Time Applications*, August 2001, pp. 58-65.

[MOR03] K. L. Morse, D. Drake and R. Brunton, "Web Enabling an RTI – an XMSF Profile". In *Proceedings of the IEEE 2003 European Simulation Interoperability Workshop*, June 2003, paper no. 03E-SIW-046.

[NIC93] D. M. Nicol, "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations". *Journal of the Association for Computing Machinery*, Vol. 40, Issue 2, 1993, pp. 304-333.

[PAN07] K. Pan, S. J. Turner, W. Cai and Z. Li, "An Efficient Sort-Based DDM Matching Algorithm for HLA Applications with a Large Spatial Environment". In *Proceedings of the 21st ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*,

2007, 70-82.

[PET97] M. D. Petty and A. Mukherjee, "Experimental Comparison of d-Rectangle Intersection Algorithms Applied to HLA Data Distribution". In *Proceedings of the 1997 Fall Simulation Interoperability Workshop*, 1997.

[PET2004] M. D. Petty and K. L. Morse, "The Computational Complexity of the High Level Architecture Data Distribution Management Matching and Connecting Processes". *Simulation Modeling Practice and Theory*, Vol. 12, Issues 3-4, July 2004, pp. 217-237.

[PUL05] J. M. Pullen, R. Brunton, D. Brutzman, D. Drake, M. Hieb, K. L. Morse and A. Tolk, "Using Web Services to Integrate Heterogeneous Simulations in a Grid Environment". *Future Generation Computer Systems*, Vol. 21, No. 1, January 2005, pp. 97-106.

[QUAKE] ID Software. Quake III. Available via http://planetquake.gamespy.com/quake3/.

[RAC02] C. Raczy, J. Yu, G. Tan, T. S. Chuan and R. Ayani, "Adaptive Data Distribution Management for HLA RTI". In *Proceedings of the IEEE 2002 European Simulation Interoperability Workshop*, June 2002, paper no. 02E-SIW-043.

[RAC05] C. Raczy, G. Tan and J. Yu, "A Sort-Based DMM Matching Algorithm for HLA". *ACM Transactions on Modeling and Computer Simulation*, Vol. 15, No. 1, January 2005, pp. 14-38.

[RAK96] S. J. Rak and D. J. Van Hook, "Evaluation of Grid-Based Relevance Filtering for Multicast Group Assignment". In *Proceedings of the Distributed Interactive Simulation*, March 1996, pp. 739-747.

[RYC06] K. Rycerz, "Grid-Based HLA Simulation Support". PhD thesis,

University van Amsterdam and AGH krakow, July 2006. Promotor: Prof. Dr. P. M. A. Sloot, Co-promotor: Dr. M. Bubak.

[SAM85] B. Samadi, "Distributed Simulation, Algorithms and Performance Analysis". Phd thesis, Computer Science Department, University of California, 1985.

[SAN08] A. Santoro and R. M. Fujimoto, "Offloading Data Distribution Management to Network Processors in HLA-Based Distributed Simulations". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, No. 3, March 2008, pp. 289-298.

[SIN99] S. Singhal and M. Zyda, "Networked Virtual Environments: Design and Implementation". *Addison-Wesley*, 1999.

[SOAP] Simple Object Access Protocol (SOAP). Available via http://www.w3.org/tr/soap/.

[SOHR] A Service Oriented HLA RTI (SOHR). Available via http://pdcc.ntu.edu.sg/sohr/.

[SOT] Borja Sotomayor, "The Globus Toolkit 4 Programmer's Tutorial". Available via http://gdp.globus.org/gt4-tutorial/.

[SUN] Sun-microsystems, "J2SE 1.5.0 Tool Documentations For Windows". Available via http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/java.html.

[TAN00] G. Tan, R. Ayani, Y. Zhang and F. Moradi, "Grid-Based Data Management in Distributed Simulation". In *Proceddings of the 33rd Annual Simulation Symposium*, 2000, pp. 7-13.

[TAN00] G. Tan, Y. Zhang and R. Ayani, "A Hybrid Approach to Data Distribution Management". In *Proceedings of the 4th IEEE International Workshop on Distributed Simulation and Real-Time*

Applications, August 2000, pp. 55-61.

[TUE03] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling and P. Vanderbilt, "Open Grid Services Infrastructure (OGSI) Version 1.0". *Open Grid Services Infrastructure WG, Global Grid Forum*, June 2003. Available via http://xml.coverpages.org/OGSI-SpecificationV110.pdf.

[VAN98] D. J. Van Hook and J. O. Calvin, "Data Distribution Management in RTI 1.3". In *Proceedings of the 1998 Spring Simulation Interoperability Workshop*, March 1998, paper no. 98S-SIW-206.

[VAN94] D. J. Van Hook, S. J. Rak and J. O. Calvin, "Approaches to Relevance Filtering". In Proceedings of the 11th Workshop on Standards for the Interoperability of Distributed Simulations, 1994.

[WAN06] X. Wang, "Synchronization Issues in Distributed Simulation Using Commercial Off-The-Shelf Simulation Packages". PhD thesis, Nanyang Technological University, January 2006. Promotor: Prof. Dr. Stephen John Turner.

[WOL97] H. Wolfson, S. B. Boswell, D. J. Van Hook and S. M. McGarry, "Reliable Multicast in the STOW RTI Prototype". In *Proceedings of the 1997 Spring Simulation Interoperability Workshop*, March 1997, paper no. 97S-SIW-119.

[WOO02] D. D. Wood, "Implementation of DDM in the MAK High Performance RTI". In *Proceedings of the 2002 Spring Simulation Interoperability Workshop*, 2002, paper no. 02S-SIW-056.

[WOW] Blizzard Entertainment. World of Warcraft. Available via http://www.worldofwarcraft.com/index.xml.

[WSDL] Web Services Description Language (WSDL) 1.1. Available via http://www.w3.org/TR/wsdl.

[WSRF] The WS-Resource Framework. Available via http://www.globus.org/wsrf/.

[WU04] W. Wu, Z. Zhou, S. Wang and Q. Zhao, "Aegis: A Simulation Grid Oriented to Large-Scale Distributed Simulation". In *Proceedings of the 3rd International Conference on Grid and Cooperative Computing*, 2004, pp. 413-422.

[XIA06] N. Li, Z. Xiao, L. Xu and X. Peng, "Research and Realization of Collaborative M&S Services in Simulation Grid". In Proceedings of *the 6th IEEE International Symposium on Cluster Computing and the Grid, International Workshop on Distributed Simulation on the Grid*, 2006.

[XIE05] Y. Xie, Y. M. Teo, W. Cai and S. J. Turner, "Service Provisioning for HLA-based Distributed Simulation on the Grid". In *Proceedings of the 19th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*, June 2005, pp. 282-291.

[XML] Extensible Markup Language (XML). Available via http://www.w3.org/XML/.

[XMSF] Extensible Modeling and Simulation Framework (XMSF) Project. Available via https://www.movesinstitute.org/xmsf/xmsf.html.

[XU06] L. Xu and X. Peng, "SSB: A Grid-Based Infrasturcture for HLA Systems". In Proceedings of *the 6th IEEE International Symposium on Cluster Computing and the Grid, International Workshop on Distributed Simulation on the Grid*, 2006.

[YVG] YVG Staff. Video game sales break records. Available via http://us.il.yimg.com/videogames.yahoo.com/feature/video-game-sal

es-break-records/1181404.

[ZHU06] S. Zhu, Z. Du and X. Chai, "GDSA: A Grid-Based Distributed Simulation Architecture". In Proceedings of *the 6th IEEE International Symposium on Cluster Computing and the Grid, International Workshop on Distributed Simulation on the Grid*, 2006.

[ZON04] W. Zong, Y. Wang, W. Cai and S. J. Turner, "Grid Services and Service Discovery for HLA-based Distributed Simulation". In *Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real Time Applications*, October 2004, pp. 116-124.