RESOURCE-CONSTRAINT AND SCALABLE DATA DISTRIBUTION MANAGEMENT FOR HIGH LEVEL ARCHITECTURE

by

PANKAJ GUPTA

B.S. Govt. College of Engineering, Pune, University of Pune, India, 1997
M.S. University of Central Florida, Orlando, USA, 2006

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2007

Major Professor: Ratan K. Guha

ABSTRACT

In this dissertation, we present an efficient algorithm, called *P-Pruning algorithm*, for data distribution management problem in High Level Architecture. High Level Architecture (HLA) presents a framework for modeling and simulation within the Department of Defense (DoD) and forms the basis of IEEE 1516 standard. The goal of this architecture is to interoperate multiple simulations and facilitate the reuse of simulation components. Data Distribution Management (DDM) is one of the six components in HLA that is responsible for limiting and controlling the data exchanged in a simulation and reducing the processing requirements of federates. DDM is also an important problem in the parallel and distributed computing domain, especially in large-scale distributed modeling and simulation applications, where control on data exchange among the simulated entities is required.

We present a performance-evaluation simulation study of the *P-Pruning* algorithm against three techniques: region-matching, fixed-grid, and dynamic-grid DDM algorithms. The *P-Pruning* algorithm is faster than region-matching, fixed-grid, and dynamic-grid DDM algorithms as it avoid the quadratic computation step involved in other algorithms. The simulation results show that the P-Pruning DDM algorithm uses memory at run-time more efficiently and requires less number of multicast groups as compared to the three algorithms. To increase the scalability of P-Pruning algorithm, we develop a resource-efficient enhancement for the *P-Pruning algorithm*. We also present a performance evaluation study of this resource-efficient algorithm in a memory-constraint environment. The *Memory-Constraint* P-Pruning algorithm deploys I/O efficient data-

structures for optimized memory access at run-time. The simulation results show that the *Memory-Constraint P-Pruning* DDM algorithm is faster than the P-Pruning algorithm and utilizes memory at run-time more efficiently. It is suitable for high performance distributed simulation applications as it improves the scalability of the P-Pruning algorithm by several order in terms of number of federates. We analyze the computation complexity of the *P-Pruning* algorithm using average-case analysis. We have also extended the P-Pruning algorithm to three-dimensional routing space. In addition, we present the P-Pruning algorithm for dynamic conditions where the distribution of federated is changing at run-time. The dynamic P-Pruning algorithm investigates the changes among federates regions and rebuilds all the affected multicast groups.

We have also integrated the P-Pruning algorithm with FDK, an implementation of the HLA architecture. The integration involves the design and implementation of the communicator module for mapping federate interest regions. We provide a modular overview of P-Pruning algorithm components and describe the functional flow for creating multicast groups during simulation. We investigate the deficiencies in DDM implementation under FDK and suggest an approach to overcome them using P-Pruning algorithm. We have enhanced FDK from its existing HLA 1.3 specification by using IEEE 1516 standard for DDM implementation. We provide the system setup instructions and communication routines for running the integrated on a network of machines. We also describe implementation details involved in integration of P-Pruning algorithm with FDK and provide results of our experiences.

To

my parents,

my grandparents,

Sequeira family,

my teachers,

and

guruji.

Manaojavam Maarutatulyavegam
Jitendriyam buddhimataam varishtham
Vaataatmajam Vaanarayayoothamukhyam
Sri Ramadootam Saranam prapadye

ACKNOWLEDGMENTS

First and foremost, I would like to express my most sincere gratitude to my advisor Dr. Ratan Guha for his continuous guidance and encouragement throughout my Ph.D. I would probably have not successfully completed Ph.D. without his unwavering support and kindness. I would also like to thank members of my research committee - Dr. Mostafa Bassiouni, Dr. Sheau-Dong Lang, and Dr. Michael Proctor for their assistance and guidance in completing this work.

I am grateful to all the people in Computer Science Department for helping me during my time here: Denise Tjon Ket Tjong, Jenny Shen, Linda Lockey, Rob Traub, and Heather Oakes. I would like to acknowledge the help of Parallel and Distributed Computing group at Georgia Institute of Technology for providing me the source code of FDK software. I would also like to acknowledge the financial support from Dr. Guha, UCF School of Electrical Engineering and Computer Science, Dr. Narsingh Deo, Dr. Guru Prasad, Aximetric Inc., Dr. Ivan Garibay, and UCF Office of Research & Commercialization during my doctoral program. I am also grateful to Dr. Katherine Morse, Dr. Mikel Petty, and Dr. Bernard Zeigler for providing helpful research directions.

I would like to thank my friends in Orlando whom I was lucky to meet and share my experiences: Olcay Kursun, Yayati Kasralikar, Shahabuddin Muhammad, Sudipta Rakshit, Kiran Anna, Abhishek Karnik, and many more. I thank you all for your companionship, best wishes, and support.

Last, and certainly the most, I thank my parents and grandparents for their patience, love, and encouragement. I attribute all my success to their blessings and prayers. I am forever indebted to the Sequeira family in Mumbai, India who provided the educational foundation that enabled me to come so far. I was very lucky to meet and share experiences in Orlando through company of great people like Guruji Jerrybandan, Uncle Nishan Mahabir, and all members of Om-Tat-Sat family. Finally, I owe this thesis to the unconditional love, encouragement and support of my wife, Kamini. This long, roller-coaster journey would not have ended successfully without her steadfast company.

I acknowledge the support of the Army Research Office under grants DAAD19-01-1-0502, and the National Science Foundation under the Grant EIA 0086251. The views and conclusions herein are those of the author and do not represent the official policies of the funding agencies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS & ACRONYMS

a	Dimension of each grid cell
P	Set of publisher regions
S	Set of subscriber regions
F	Set of Federates
G	Set of Grid Cells
n	Number of Federates
R	Length of Routing Space on X-axis
Fed_ID	Identifier for each federate
ListX	Array for storing information on publisher and subscriber regions
List_ID	Identifier for each element of ListX
E(x)	Expected value of *x*
Pr(x)	Probability of variable *x*
AWACS	Airborne Warning and Control System
DDM	Data Distribution Management
DIS	Distributed Interactive Simulation
DMMC	Dual Mode Multicast
DMSO	Defense Modeling and Simulation Office
FDK	Federated Simulation Development Kit
HLA	High Level Architecture
JSTAR	Joint Surveillance Target Attack Radar System
MCG	Multicast Group
RTI	Runtime Infrastructure
STOW	Synthetic Theater of War
SRTP	Selectively Reliable Transmission Protocol
SPEEDES	Synchronous Parallel Emulation Environment for Discrete Event Simulation

CHAPTER ONE: INTRODUCTION

Distributed Simulation is a cost-effective technique for system studies in research, modeling, and training. The High Level Architecture (HLA) presents a framework for modeling and simulation within the Department of Defense (DoD). The goal of this architecture is to interoperate multiple simulations and facilitate the reuse of simulation components. HLA allows interconnection of simulations, devices, and human operators in a common federation. It builds on composability, letting designer construct simulations from pre-built components. Each computer-based simulation system is called a *federate* and the group of interoperating systems is called a *federation*. HLA specifications—incorporated as IEEE 1516 standard—were developed to provide reusability and interoperability.

The HLA Run-Time Infrastructure (RTI) provides a set of services used to interconnect simulation during a federation execution. These RTI services are grouped into the six categories: federation management, declaration management, object management, ownership management, time management, and data distribution management. RTI provides a degree of portability (across computing platforms, operating systems, and communication systems) and simulation interoperability. RTI is also responsible for information exchange during the execution. It allows federates to join and resign, declare their intent to publish information, send information about objects, attributes and interaction, and synchronize time.

A distributed simulation consists of a collection of autonomous simulators, or federates, that are interconnected using RTI software. RTI implements relevant services

required by the federated simulation environment. The most important services for the purposes of this discussion fall into two basic categories: *Time Management* and *Data Distribution*. The time management services ensure that the simulation time in each of the simulator instances stays synchronized with the others, and the data distribution services allow for the transitioning of event messages from one simulator to another.

Data Distribution Management (DDM) services extend Declaration management services using routing space and regions in HLA. In distributed simulation environment, every action takes place on a simulator that may affect or may be of interest to another simulator, requires a message. In a large-scale distributed simulation, such as those encountered in defense applications, simulating many objects that are of interest to other objects can result in increased communication across a network, on the scale of $O(n^2)$. Data Distribution Management is responsible for limiting and controlling the data exchanged in a simulation. It also aims at reducing the processing requirements of simulation hosts, or federates, by communicating updates regarding interactions and state information only to federates that require them.

1.1 Motivation and Goals

DDM is important not only as a crucial service in HLA/RTI, but also as an important problem in the parallel and distributed simulation domain. For a sequential simulation, all the simulated entities can exist on single machine and can have direct access to state information and events. However, for a distributed simulation environment, especially in large-scale simulation such as those in defense applications, control on data exchange among the simulated entities is required.

Data distribution techniques are also important in diverse computing applications such as Web server infrastructure [20], load-balancing schemes [21], Web services [22] and parallel computing ([23], [24], [25], [27], [26], [28], [29], [30]). The general nature of the problem in these domains is similar to those encountered in distributed simulation. Hence, advances in DDM research are applicable to wide areas of computing and simulation.

1.2 Dissertation Organization

The rest of the dissertation is organized as follows. Chapter 2 provides a background on the importance of data distribution techniques, concepts of routing space, and a review of related work in DDM research. Chapter 3 provides an algorithmic overview of current DDM algorithms. The P-Pruning algorithm for DDM matching problem with its three sub-procedures and an illustration is presented in Chapter 4. We also analyze the computational complexity of P-Pruning algorithm, distribution of federates within the routing space, and size of multicast groups in Chapter 4. The performance-evaluation of P-Pruning algorithm with other DDM algorithms with implementation details and simulation results are discussed in Chapter 5. Chapter 6 highlights the resource constraint issues in data distribution management and proposes memory-efficient enhancements in P-Pruning algorithm. It also includes the performance-evaluation study details and simulation results. The integration of P-Pruning algorithm with FDK software is discussed in Chapter 7. Chapter 8 identifies the directions for further research. It lays out future directions for development of distributed DDM approach on cluster computers and testing with real-world data. Finally, Chapter 9 presents the concluding remarks.

1.3 Research Contributions

The objective of this research work is to design efficient and scalable strategies for Data Distribution Management in resource-constraint well as distributed environment. We have proposed a new algorithm, called *P-Pruning algorithm*, for the DDM region matching problem. We have also shown that this algorithm performs better than several DDM strategies using average-case complexity analysis and through performance evaluation experiments. We have also developed a resource-efficient method for DDM. We also investigate scalability issues and develop a scalable approach for DDM in distributed environment such as cluster computers. Throughout the dissertation, we have used IEEE 1516 specifications for representing the federates, and their publisher and subscriber regions.

CHAPTER TWO: BACKGROUND AND RELATED WORK

In this Chapter, we provide a background on HLA, different implementation of RTI in commercial and academic institutions, and DDM. Examples of RTI implementations are the FDK software developed at the Georgia Institute of Technology and the Light-Weight RTI built at George Mason University. We also highlight the importance of DDM problem in the parallel and distributed simulation domain and the underlying concept of routing space. Finally, we present an overview of current state-of-the-art research work in DDM.

2.1 HLA Architecture

The High Level Architecture (HLA) was developed by the Department of Defense (DoD) under the leadership of the Defense Modeling and Simulation Office (DMSO). It is an architecture for reuse and interoperation of simulation components within the DoD. The HLA is intended to provide a structure for reuse of capabilities available in different simulations, thereby reducing the cost and time required to create a synthetic environment for a new purpose and providing developers the option of distributed collaborative development of complex simulation applications. In recent years, HLA has been applied across a wide range of simulation application areas, including education and training, analysis, engineering, and even entertainment, at different levels of resolution. HLA specifications do not require a particular implementation, or use of a programming language. There are numerous implementations of HLA in industry and academic research institutions. Appendix A provides an overview of different implementations of HLA with their key characteristics.

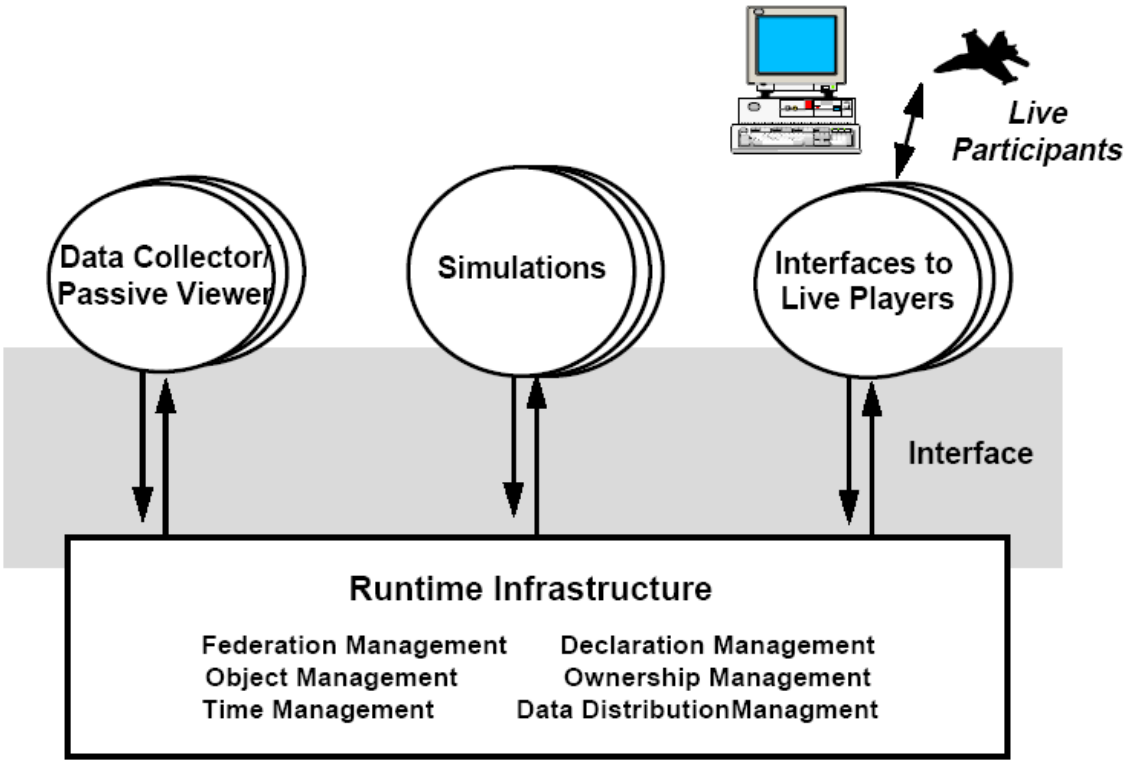


Figure 1. Components in HLA RTI

The HLA is defined by three concepts:

- Object Model Template
- Runtime Infrastructure
- HLA compliance rules

The RTI and compliance rules are uniform across all HLA-compliant simulations. However, each group of interacting simulations, or federates, must define a basis for the exchange of data and events between simulations. The format and content of this basis is defined by the Object Model Templates. The Object Model Templates are sued to describe the objects that will exist in the federation.

The key components of HLA federation are illustrated in Figure 1. The federation is composed of individual simulations or federates who form the first key component of HLA. A federate can be a computer simulation, a manned simulator, or supporting utility (such as a viewer or data collector), or an interface to a live player or instrumented facility. The HLA requires all federates incorporate specified capabilities to allow the objects in the simulation through the exchange of data supported by services implemented in the RTI. The second key component of HLA is the RTI. The RTI is functionally equivalent to a distributed operating system for the federation. It provides a set of general purpose services that support federate-to-federate interactions, federation management functions. All interactions among the federates are executed through the RTI. The third component in HLA federation is the interface to RTI. The HLA runtime interface specification provides a standard method for federates to interact with the RTI, to invoke the RTI services for supporting runtime interactions among federates and to respond to requests from the RTI. This interface is implementation independent and free of the specific object models and data exchange requirements of any federation.

The HLA supports passive collection of simulation data and monitoring of simulation activities. Also, HLA supports interfaces to live participants, such as instrumented platforms or live systems. Live participants interact with the simulated world through an interface that feeds the representation of the live world into the simulated world and projects data from the simulated world back to the live world. Formally, the HLA is defined by three components: the interface specification, the object model template, and the HLA rules.

2.1.1 RTI

The RTI is a collection of software that provides commonly required services to simulation system. It is also intended to provide a measure of portability (across computing platforms, operating systems, and communication systems) and simulation interoperability. There are six classes of services in RTI: Federation Management, Declaration Management, Object Management, Ownership Management, Time Management, and Data Distribution Management. Federation Management services offer basic functions required to create and operate a federation. *Declaration management* services support efficient management of data exchange through the information provided by federates defining the data they will provide and will require during a federation execution. *Object management* services provide creation, deletion, identification and other services at the object level. *Ownership management* services support the dynamic transfer of ownership of object/attributes during an execution. *Time management* services support synchronization of simulation data exchanges. Finally, *data distribution management* services support the efficient routing of data among federates during the course of a federation execution.

The HLA rules define the principles of HLA in terms of responsibilities that federates and federations must uphold. Each federation has a Federation Object Model (FOM), which is a common object model for the data exchanged between federates in a federation. OMT is the meta-model for all FOMs. The Interface Specification defines the standard services and interfaces to be used between the federates and the RTI. While the HLA is an architecture, not software, it is the RTI software that facilitates the interaction

between federates using a common FOM. Actions of a federate, such as registering an object, updating attributes of an object or sending an interaction, are defined as HLA events. Each federate has a RTI Ambassador and a Federate Ambassador. During simulation, a federate employs RTI Ambassador (*RTIAmb*) methods to generate events and the Federate Ambassador (*FedAmb*) acts as a callback module to receive events originating from other federates.

2.1.2 Time Management

There are two principal components to the HLA time management (TM) services. First, a time stamp ordered (TSO) message delivery service guarantees that successive messages delivered to each federate have non-decreasing time stamps. Second, the time management services manage simulation time (termed logical time in the HLA) advances of each federate. Federates must explicitly request that their logical time be advanced by invoking an IFSpec service such as *Next Event Request, Time Advance Request, or Flush Queue Request*. The RTI only grants the advance via the *Time Advance Grant* service (callback) when it can guarantee that no TSO messages will later be delivered with a time stamp smaller than the granted advance time. In this way the RTI ensures federates never receives messages with time stamp less than their current logical time. In the HLA, time management is distinct from sending and receiving messages (events). Services such as *Update Attribute Values* and *Reflect Attribute Values* are used to send and receive messages, respectively.

2.1.3 HLA Implementations

FDK

Federated Simulations Development Kit (FDK) is HLA-based RTI software system developed at Georgia Institute of Technology ([1], [2], [3]). It contains composable modules for building run-time infrastructures (RTI) using which different simulations can be integrated together. RTI-Kit, a principal component of FDK, is a collection of libraries. It supports development of Run-Time Infrastructures for parallel and distributed simulation systems, especially federated simulation systems running on high performance computing platforms. FDK is designed so that RTI developers can pick and choose from the set of FDK modules that are most appropriate for developing their particular RTI implementation. Each library is designed so it can be used separately, or together with other RTI-Kit libraries, depending on the functionality required by the user. Because each library is designed as a stand-alone component, RTIs that are constructed using RTI-Kit are highly modular, with clear, well-defined (and documented) interfaces. These libraries can be embedded into existing RTIs to add new functionality. RTI developers can benefit from incorporating these ready-made modules, and avoid having to develop them on their own. FDK is a modular and reusable set of libraries designed to facilitate the development of RTIs for developing or integrating parallel and distributed simulation systems.

RTI Prototype

The RTI Prototype was developed during the mid-1990's at Massachusetts Institute of Technology (MIT) Lincoln Laboratories [4]. The HLA specification was still under development and undergoing revisions during this time period, so the creators of the RTI Prototype did not implement all of the HLA services (for example, Time Management was omitted) or follow the specification to the letter. Instead, their implementation was designed as a proof of concept for HLA, with the intent of also providing feedback that might impact the development of the HLA itself. The API for the RTI is specified in the following in languages: IDL, C++, Java, and Ada. IDL stands for Interface Definition Language, a part of the CORBA specification. The RTI Prototype uses the IDL API specification, and was developed using C++ and the Iona ORBIX implementation of CORBA. The approach to Data Distribution Management (DDM) used in the RTI Prototype is known as the Fixed Grid-based Approach, which we discuss in detail in the Chapter 2 about different approaches to DDM.

Before the RTI Prototype was tested in the Synthetic Theater of War (STOW) federation, laboratory experiments were conducted. The experimental results available deal with only two federates. These experiments seem to have been designed to verify that the RTI Prototype was indeed functioning properly, and not to show how it performs when each federate is simulating many objects which are moving around in the battlespace, as no such scenario was performed. When the STOW exercise was conducted, a data logger captured statistics, such as the total number of packets sent over the network. These statistics seem to be confined to the data-link level of the simulation, where as the HLA deals with a higher level of abstraction. Therefore, this data

unfortunately does not give direct insight into the performance of the RTI Prototype or the Data Distribution Management system.

RTI 1.3

The same group at Lincoln Labs that designed the RTI Prototype developed the RTI 1.3 [5]. The RTI 1.3 is the successor of the RTI Prototype and is so named because it implements version 1.3 of the HLA Specification. One of the main changes between the RTI Prototype and the RTI 1.3 was the Data Distribution Management strategy that was used. The RTI Prototype used a grid-based approach, whereas the RTI 1.3 employs a region-based method. The region-based method is described in detail in Chapter 2 on DDM algorithm overview. A distinguishing feature of the region-based DDM approach is the use of a single database to store information regarding the regions of interest declared by all federates in the federation. The database for regions, subscriptions, and publications used by the RTI 1.3 is called the Information Manager (IM). Unfortunately, no experimental results have been published concerning the performance of the RTI 1.3.

GMU Light-Weight RTI

The light-weight RTI developed at George Mason University (GMU) focuses on Declaration Management and the Data Distribution Management services [6]. Time Management and Ownership Management are not implemented, since these services were not the primary objective of the project. As a result, the light-weight RTI is best suited to real time simulations by federation of small to medium size. A useful feature of this RTI is that it can be interfaced to DIS simulations using a DIS to HLA translator

developed at GMU. The motivation behind its construction was to understand the HLA and to bring the earlier work done by those researchers into compatibility with the HLA. We shall now briefly describe the previous works out of which the light-weight RTI grew.

The light-weight RTI uses elements of the Dual Mode Multicast scheme and the Selectively Reliable Transmission Protocol. Dual Mode Multicast (DMMC) is a method of Data Distribution Management that was developed for use by systems adhering to the Distributed Interactive Simulation (DIS) protocol. DMMC uses a multi-level grid-based filtering scheme. An exercise-wide multicast group is used on the wide area network (WAN) and a fixed grid-based approach is used to determine multicast groups at level of the local area network (LAN).

Selectively Reliable Transmission Protocol (SRTP) is designed for applications, such as DIS and HLA, that need reliable multicast communications. SRTP runs in user space and forms a sublayer between an application and the Internet protocol stack of the operating system. SRTP operates in three modes: best effort multicast reliable multicast, reliable multicast, and lightweight reliable transaction-oriented unicast. The reliable multicast uses negative acknowledgement with NAK suppression mechanisms to avoid congestion at the sender. The major flaw in the light-weight RTI, according to its creators, is the poor runtime performance that is partly due to the use of SRTP, which is slower than UDP. Improving the performance of the light-weight RTI is the primary goal of the future work on this project.

RTI 1.3NG

Science Applications International Corporation (SAIC) developed RTI 1.3NG which was sponsored by Defense Modeling and Simulation Office (DMSO) based on competitive industry designs ([7], [8]). Like the RTI 1.3, the RTI 1.3NG also supports the HLA Specification 1.3. However, the RTI 1.3NG is intended to be a full implementation of all HLA services and will supersede the RTI 1.3, which is no longer being supported by DMSO ([9], [10], [11], [12], [14], [15], [16]). As yet, no literature has been published regarding the design, implementation, or performance of the RTI 1.3NG, which is still under development.

MAK RTI

MAK Technologies, which is based in Cambridge, Massachusetts, and is a leading provider of simulation networking software, developed the MAK RTI ([17], [18], [19]). It supports the HLA Specification 1.3, and is link-compatible with DMSO RTI 1.3. Like the other RTI implementations that we have discussed, the MAK RTI does not implement all HLA services. The MAK RTI can be downloaded at no charge, and runs on the Windows 95/98/NT, Solaris, IRIX, and Red Hat Linux platforms.

HPC-RTI

RAM Labs, based in San Diego, California, is developing an RTI designed for use in high performance computing (HPC) environments. The RTI-HPC is integrated with the Synchronous Parallel Emulation Environment for Discrete Event Simulation (SPEEDES). SPEEDES is a government-owned software system, managed by RAM

Labs, and licensed by NASA. SPEEDES was used in the early 90's to model global ballistic missile defense applications on parallel and distributed supercomputers. SPEEDES currently supports various large-scale distributed simulation projects, under the sponsorship of the Department of Defense, such as Wargame 2000, Joint Simulation System (JSIMS), and Extended Air Defense Test Bed (EADTB). SPEEDES is implemented in C++, and supported operating systems are IRIX, HPUX, Solaris, Linux, and Windows NT.

With the HLA gaining popularity in the defense community, SPEEDES is being modified and augmented to serve as an RTI that is compatible with the HLA specifications, and the result will be the RTI-HPC. The RTI-HPC is the first attempt to transform a pre-existing simulation engine into an RTI. There is another element of the RTI-HPC that differentiates it from the other RTI implementations, and that is its support of time management. The RTI-HPC will provide time-management across all six HLA services. The HLA does not require the Declaration Management, Data Distribution Management, or Ownership Management services to be time-managed. However, such capabilities would be extremely useful for a federation that wanted to enforce casual ordering of events, for the purpose of repeatability or other reasons. In the remainder of this section, we will discuss how SPEEDES performs Data Distribution Management.

The SPEEDES Data Distribution Management mechanism follows a grid-based approach. Interest regions are mapped to grid cells, which are represented by entities called Hierarchical Grid (HiGrids) where region-overlap computations are performed. The HiGrids are distributed among the participating nodes. When an overlap is detected, HiGrids tell the publisher which subscribers are interested in the publisher's attributes.

The publisher then sends the subscriber its proxy, which represents the state of an entity. The proxy is updated whenever the publisher's attributes change, thus ensuring that the subscriber can assess the publisher's current attributed values.

The SPEEDES method of performing data distribution management was developed prior to the advent of the HLA specification, so some changes will need to be made in the transition to the HPC-RTI. The concept of a proxy is not fully compliant with the HLA specification. In addition, SPEEDES allows subscriptions and publications on a per-object basis, whereas the HLA only allows interests to be declared on a per-federate basis. The use of proxies, and the per-object interest expressions, are features that may need to be modified as SPEEDES is converted to the RTI-HPC. It remains to be seen how closely the RTI-HPC method of supporting the HLA Data Distribution Management service resembles the current SPEEDES approach to DDM.

2.2 IEEE 1516 vs. RTI 1.3

IEEE 1516 is the HLA standard approved by IEEE in September 2000 as a successor of the HLA 1.3 specifications ([31], [32], [33]). It simplifies the DDM implementation by removing multiple routing spaces and incorporating all the dimensions within this routing space. The regions are composed of dimension name and range pairs. Two regions overlap only if they have at one dimension in common. If two regions do have one or more dimensions in common, then the regions overlap if and only if ranges for all the dimensions that the regions have in common overlap pair-wise. Petty [34] and Morse [35] have discussed the migration of HLA 1.3 based simulation system to IEEE 1516 standards in detail.

2.3 Concept of Routing Space

DDM is based on a multi-dimensional coordinate system called a routing space. For example, a two-dimensional routing space might represent the play box in a virtual environment. A rectangular publisher region within the routing space is associated with each update message generated by a publishing federate. Receiving federates declare their interests via rectangular subscriber regions within the routing space. If the publisher region associated with a message overlaps with the subscriber region of a federate, the message is routed to that subscribing federate. By calculating the intersection of publisher and subscriber regions, the Run-Time Infrastructure in HLA establishes connectivity between sender and receiver federates for routing updates and interactions. Each overlapping subscriber and publisher federate joins a multicast group to facilitate the message transfer. For example, in Figure 2, updates using publisher region *P* are routed to federates subscribing to region S_1, but not to federates subscribing to region S_2.

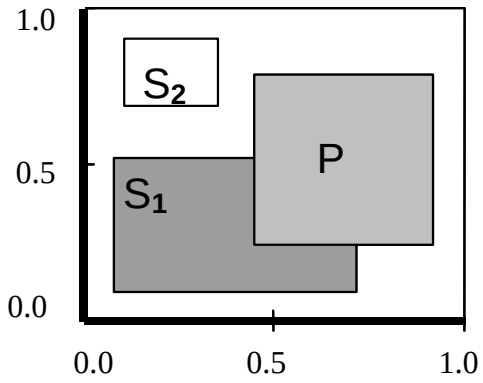


Figure 2. Two-dimensional routing space with subscriber regions: S_1 and S_2, and a publisher region *P*

The DDM provides flexible mechanism for publishing and subscribing interests through multidimensional routing space. The basic structure of routing space in the IEEE 1516 standard is as following:

Routing space: There is a single routing space and all dimensions are included in this routing space.

Regions: A region is a single rectangular subspace within the coordinate space. Regions may be defined on any subset of the available dimensions of the coordinate system.

Region set: Regions are grouped into region sets, which consist of one or more regions. The regions in a region set need not all have the same subset of the dimensions of the coordinate system.

Dimension: Dimensions correspond to simulation data and they are used to define regions.

The interest matching process begins by specifying subscription and update regions. An object is said to be interested by a federate if and only if at least one of the object's attributes is subscribed by the federate (through declaration management) and at least one update region associating with the object overlaps the subscriber region of the federate.

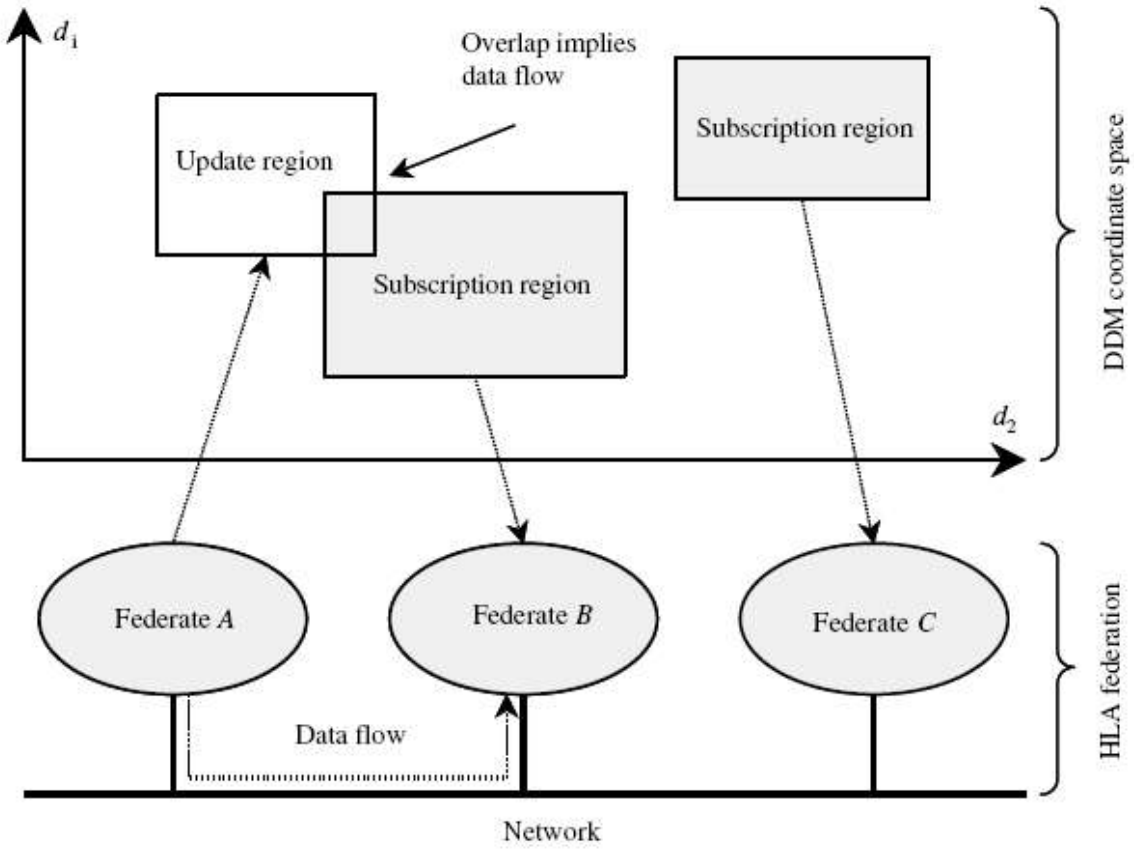


Figure 3. Concept of region overlap in two-dimensional routing space

In the Figure 3, a notional federation has three federates; each has declared one region. The update region declared by federate A overlaps the subscriber region declared by federate B, so updates to the data items associated with the update region are delivered by the HLA RTI from federate A to federate B. No data is delivered to federate C.

For instance, a routing space representing the position of the units on the battlefield could be called "Map", with two dimensions "X" and "Y". A region in the routing space Map would be the ranges for the dimensions X and Y: [(x_{min}, x_{max}), (y_{min}, y_{max})]. A region set would be a set of one or several of such regions. A publisher region could be for instance a region with one extent representing the position of a unit. A

subscriber region could be for instance a region with several extents, each of them representing the range of some sensors.

In Figure 4, we have illustrated the publisher and subscriber regions for a squadron of airplanes. The spy plane in region 1 tracks the squadron of planes in region 3.

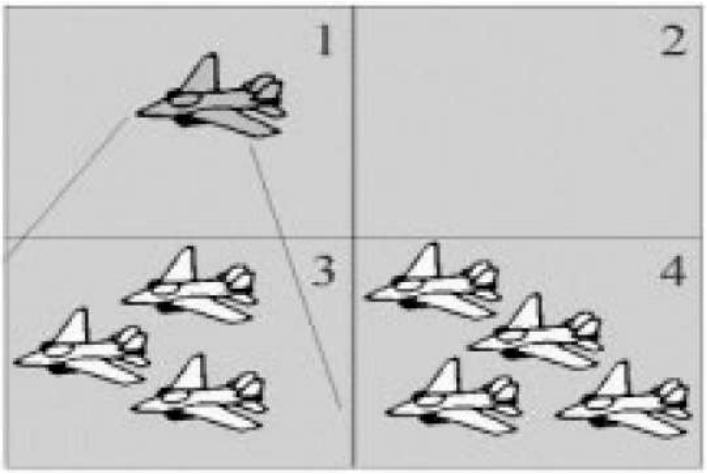


Figure 4. Illustration of publisher and subscriber region in an airplane squadron

2.4 Related Work

The earliest work on DDM research appears in Van Hook *et al.* [4]. The HLA specification, key elements in its architecture, and implementation are described in Dahmann *et al.* ([36], [37], [38], [39]) and Van Hook [5]. An overview and tutorial of DDM and related research work appears in ([40], [41], [42]).

Boukerche and Roy [43] described taxonomy of DDM schemes and basic concepts. Petty [34] presented a comparison of the DoD 1.3 and IEEE 1516 HLA specifications. Since 1995, different DDM algorithms have been proposed such as the fixed-grid [44], dynamic-grid ([45], [46], [47]), region-matching [4], agent-based [49],

and hybrid-method [50]. A sort-based algorithm running in $O(n^2)$ time is presented in ([51], [52]). A partition-based DDM technique is proposed in Kumova [53]. In Liu *et al.* [54], DDM for multidimensional routing space is explored, while a DDM scheme for distributed virtual environments is presented in Minson and Theodoropoulos [55]**.** Performance-evaluation study of different DDM strategies appears in Boukerche and Dzermajko ([56], [57], [58]), Gupta and Guha [59]. Scalability-related issues for the implementation of DDM are addressed in [103].

In their doctoral dissertations, Morse [60] and Petty [61] made numerous algorithmic contributions to the DDM problem. The problem of dynamic multicast grouping is addressed in [60], while an interval-tree based DDM algorithm is presented in [61]. A connection-graph based cost-function approach is proposed by Morse in [62]. In this method, the latency of data communication is taken into account. A computational analysis of various DDM strategies appears in [63]. Petty and Morse have discussed the computational complexity of the high level architecture data distribution management matching and connecting processes in ([64], [65], [66]).

Federated Simulations Development Kit (FDK) is an implementation of HLA architecture developed at Georgia Institute of Technology [3]. It has been used by researchers in academia, industry, and government laboratories as an effective software package for evaluating their research contributions to distributed simulation technology. FDK has been used as the platform for HLA-based distributed simulation research in ([1], [65], [70], [71], [72]). Scalability issues of FDK have been researched by Fujimoto *et al.* in [2] and Perumalla *et al.* in [103]. Synchronization issues in DDM and their remedies

have been proposed in [71]. In Chapter 7, we discuss the integration of the P-Pruning DDM algorithm with FDK.

An interesting merger of distributed simulation with Web services is presented in Pullen *et al.* [22]. This paper describes an approach for extending Web Services to distributed simulation environments and providing scalable interoperability across wide variety of networked platforms. Advanced memory management schemes such as hierarchical data-caching and pre-fetching that can be applicable to resource constraint conditions related to DDM appear in ([74], [75]).

A review of I/O efficient external memory data-structures appears in Arge et al. ([76], [77]). Memory-efficient routines and implementation details appear in [78], while Meyer et al. [79] provide a good source of algorithms for memory hierarchies. Directions on I/O efficient algorithms and dynamic memory allocation in simulation appear in (Nielsen [80], Vengroff and Vitter [81], [82]).

Distributed implementations of HLA been implemented on cluster computers has been reported in ([83], [84], [85], [86], [87], [88], [90], [92], [96], [97]).

CHAPTER THREE: OVERVIEW OF DDM ALGORITHMS

In this Chapter, we provide a brief algorithmic overview of three DDM algorithms and highlight their implementation issues.

The DDM problem is stated as follows:

Let,

$F_i \in F$, F = Set of federates,

$P_i \in P$, P = Set of publisher regions, and

$S_i \in S$, S = Set of subscriber regions.

Then, there exists a matching or clustering of subsets of publisher regions P and subscriber region S.

Let $MCG_i \in MCG$, where MCG = Set of multicast group, such that

$$\forall MCG_i \in MCG,$$
$$P_i \in MCG_i \text{ such that } |P_i| > 0,$$
$$S_i \in MCG_i \text{ such that } |S_i| > 0.$$

The DDM problem is to find all possible MCG_i at any time t.

Now, we discuss three types of DDM Schemes implemented in the current HLA.

3.1 Region-Matching Algorithm

In the region-matching DDM approach, a multicast group is defined for each publisher region. Updates are simply sent to the multicast group associated with the publisher region. A federate subscribes to the multicast group, if one or more of its subscriber

regions overlap with the publisher region. When a subscriber region changes, the new subscriber region must be matched against all other publisher regions in order to determine those that overlap with the new subscriber region. The federate must then subscribe to the multicast groups with overlapping publisher regions. Similarly, when a publisher region changes, the new publisher region must be matched against all subscriber regions to determine the new composition of the multicast group that include this publisher region. This requires examining all subscriber/publisher regions in use by the federation. Thus, it does not scale well as the number of regions becomes large.

The region-matching algorithm implementation has two sub-procedures: Create Overlap_List and Create Multicast group. This algorithm needs to scan all the publisher and subscriber regions at least once. Hence, its time complexity is quadratic.

Create Overlap_List Sub-procedure

Initialization

Overlap: Flag indicating overlap between publisher and subscriber region.

Pub_overlap_counter: Counter for subscriber regions overlapping with each publisher region.

BEGIN Procedure

For all publisher region P_i do

Set Overlap flag to FALSE;

Begin

For all subscriber region S_j do

Begin

//Check all conditions for overlap between P_i and S_j

If P_i and S_j overlap do

Begin

Set Overlap flag to TRUE;

Increment counter Pub_overlap_counter for P_i;

End

End

End

END Procedure

Create Multicast Group Sub-procedure

Initialization

MCG: Multicast group for DDM

BEGIN Procedure

For all publisher region P_i having Pub_overlap_counter > 0 do

Begin

Assign Multicast Group MCG_i to P_i;

Add all subscriber regions S_j overlapping with P_i to multicast group MCG_i;

End

END Procedure

3.2 Fixed-Grid DDM

In the fixed-grid DDM algorithm, the routing space is partitioned into non-overlapping grid cells, and a multicast group is defined for each cell. A federate subscribes to the

group associated with each cell that partially or fully overlaps with its subscriber regions. The result associates a region with several multicast groups in a fixed and pre-determined manner. A publish operation is realized by sending an update message to the multicast groups corresponding to the cells that partially or fully overlap with the associated publisher region. The fixed-grid approach eliminates the need to explicitly match publisher and subscriber regions. It is less accurate than the region-matching method, because the mapping of regions onto grids may not be exact. The actual area covered by cell may be larger than the region itself. While grid partitioning eliminates the matching overhead, large number of multicast groups is needed if a fine grid structure is defined; a coarse grid leads to imprecise filtering, negating the benefits of DDM.

The fixed grid DDM algorithm implementation consists of three sub-procedures: Grid Initialization, Federate-to-Grid mapping, and Multicast Group creation.

Grid Initialization Sub-Procedure

BEGIN Procedure

Divide the routing space into grid cells G_i of given dimension in routing space;

Each cell is uniquely identified by a cell ID;

Each grid cell maintains counters for number of publisher regions overlapping and their federate ID;

Each grid cell maintains counters for number of subscriber regions overlapping and their federate ID;

END Procedure

Federate-to-Grid Mapping Sub-Procedure

Initialization

For each grid cell G_i, the following variables are maintained:

Pub_Fed_ID: Array for storing the federate ID of each publisher region overlapping with G_i,

Pub_Region_Counter: Counter for number of publisher regions overlapping with G_i,

Sub_Fed_ID: Array for storing the federate ID of each subscriber region overlapping with G_i,

Sub_Region_Counter: Counter for number of subscriber regions overlapping with G_i.

BEGIN Procedure

For all federates F_i do

Begin

// For the publisher region P_i

For all grid cells G_i covered by publisher region P_i do

Begin

Add publisher region P_i information to grid cell G_i;

Increment the Pub_Region_Counter for grid cell G_i;

End

// For the subscriber region S_i

For all grid cells G_i covered by subscriber region S_i do

Begin

Add subscriber region S_i information to grid cell G_i;

Increment the Sub_Region_Counter for G_i;

End

End

END Procedure

Create Multicast Group Sub-procedure

Initialization

MCG: Multicast group for DDM.

Each grid cell G_i is assigned a multicast group MCG_i.

BEGIN Procedure

For all grid cell G_i do

Begin

Add all publisher regions in grid cell G_i to MCG_i;

Add all subscriber regions in grid cell G_i to MCG_i;

End

END Procedure

3.3 Dynamic-Grid DDM

The fixed-grid method offers no mechanism to prevent publishers from sending data on a multicast group that no subscribers have joined. The dynamic-grid method addresses this drawback of the fixed-grid scheme. Like fixed-grid approach, the routing space has grid overlay that defines the cells. This scheme dynamically allocates multicast groups, based on the current publisher and subscriber regions in the system and triggers hosts to join those groups, as in the region-based method. Only those cells, in which there is at least

one publishing and one subscribing federate, are assigned to a multicast group. Thus, a multicast group is allocated to each cell that is part of the intersection of a publisher region and a subscriber region. Publisher federates join and transmit on a multicast group, if there is at least one subscriber region interested in its data. Similarly, subscriber federates join and listen on multicast group, only if there is at least one publisher federate transmitting on that group.

This technique prevents the publishing federates from transmitting data on a multicast group with no subscribers and reduces the number of multicast groups that a federate needs to join.

The dynamic-grid DDM algorithm implementation consists of three sub-procedures: Grid Initialization, Federate-to-Grid mapping, and Multicast Group creation. The Grid Initialization and Federate-to-Grid mapping sub-procedures are similar to the fixed grid algorithm. Hence, we are listing only the Multicast Group creation sub-procedure.

Create Multicast Group Sub-procedure

Initialization

MCG: Multicast group for Dynamic Grid DDM.

Each Grid Cell G_i is assigned a multicast group MCG_i.

BEGIN Procedure

For all grid cell G_i

Begin

If grid cell G_i has at least one publisher region P_i

AND at least one subscriber region S_j

Begin

Add all publisher regions in grid cell G_i to MCG_i;

Add all subscriber regions in grid cell G_i to MCG_i;

End

End

END Procedure

CHAPTER FOUR: THE P-PRUNING ALGORITHM FOR DDM

In this Chapter, we present the P-Pruning DDM algorithm with its three sub-procedures. We also provide an illustration of the steps involved in P-Pruning algorithm using a small example. Finally, we analyze the computational complexity of P-Pruning algorithm using average-case analysis and the effect of federate distribution within routing space on its performance.

4.1 The P-Pruning Algorithm: An Overview

The P-Pruning DDM algorithm computes the multicast groups in three steps: List Computation, MCG Population, and MCG Pruning. Each publisher and subscriber region is described by four-coordinate system in the routing space (P_{x1}, P_{x2}, P_{y1}, P_{y2}) and (S_{x1}, S_{x2}, S_{y1}, S_{y2}), respectively. Each federate F_i has one publisher region P_i and one subscriber region S_i, where i denotes the federate ID (Fed_ID).

The entire algorithm is based on an array, ListX, whose size is equal to R, *i.e.*, the length of the routing space X-axis. The elements in ListX array correspond to the coordinates in X-axis of the routing space. The List Computation sub-procedure scans all the publisher and subscriber regions once, and stores the information about their coordinates at each point of the axis. A multicast group is assigned to an element of the ListX array, if there is a publisher region P_i whose $(P_i)_{X_1}$ coordinate coincides with this element of ListX.

The MCG Population sub-procedure creates the DDM multicast group based on information stored in ListX array, but it considers only the overlap information on X-dimension of the routing space. A multicast group is created at a point on ListX only if

there exists at least one publisher region and there is at least one subscriber region overlapping with this publisher region on X-axis. Thus, this sub-procedure creates a set of multicast groups that may include some multicast groups that are having publisher and subscriber region as members, but these member regions may not actually overlap on the Y-axis of the routing space. Overall, this sub-procedure computes the entire information faster by avoiding the simultaneous checking of X-axis and Y-axis overlap.

The errors in creation of multicast group *MCG* are now corrected by the final MCG Pruning sub-procedure. The pruning sub-procedure verifies that the regions in multicast group *MCG* actually overlap on Y-axis, and it eliminates any non-overlapping subscriber from the specific multicast group. It also verifies that every multicast group has at least one subscriber region after this step. At the end of this process, it deletes any multicast group having no subscriber region.

Since the two main sub-procedures in the algorithm perform the function of multicast group *Population* and *Pruning*, the algorithm is called *P-Pruning* algorithm throughout the remainder of this report. The P-Pruning algorithm is efficient because, unlike other algorithms, it focuses on creation of multicast groups right from the beginning. Also, it consumes less CPU and memory resources by avoiding the simultaneous checking of X and Y axis overlap.

4.2 The P-Pruning Algorithm

We now present the three sub-procedures in the P-Pruning DDM algorithm.

4.2.1 List Computation Sub-Procedure

Initialization:

ListX: List of all the coordinates for all publishers regions *P* and subscriber regions *S* in ascending order.

Each publisher and subscriber region is described by four-coordinate system in the routing space.

$\left((P_i)_{X_1}, (P_i)_{X_2}, (P_i)_{Y_1}, (P_i)_{Y_2}\right)$ are coordinates of publisher region P_i.

$\left((S_i)_{X_1}, (S_i)_{X_2}, (S_i)_{Y_1}, (S_i)_{Y_2}\right)$ are coordinates of subscriber region S_i.

Each federate F_i has one publisher region P_i and one subscriber region S_i, where *i* denotes the federate ID (Fed_ID).

ListX = Array of size *R*, where *R* is the maximum coordinate of the routing space.

For each element in ListX, following variables are maintained:

List_ID; //Identifier for each element in ListX

Pub_Region_Counter = 0; //Counter for publisher regions having $(P_i)_{X_1}$ = List_ID

X1_Sub_Region_Counter = 0; //Counter for subscriber region having $(S_i)_{X_1}$ = List_ID

X2_Sub_Region_Counter = 0; //Counter for subscriber regions having $(S_i)_{X_2}$ = List_ID

BEGIN Procedure

For each federate F_i do

Begin

//Processing the publisher region of federate F_i

List_ID = $(P_i)_{X_1}$;

Copy the publisher region counter of ListX[List_ID] into Pub_Region_Counter;

ListX[List_ID].Pub_Region[Pub_Region_Counter].Fed_ID = F_i;

Copy coordinates of publisher region P_i at

ListX[List_ID].Pub_Region[Pub_Region_Counter] ;

Increment Pub_Region_Counter;

Update the publisher region counter of ListX[List_ID];

//Processing the X1 subscriber region of federate F_i

List_ID = $(S_i)_{X_1}$;

Copy the X1 subscriber region counter of ListX[List_ID] into

X1_Sub_Region_Counter;

ListX[List_ID].X1_Sub_Region[X1_Sub_Region_Counter].Fed_ID = F_i;

Copy coordinates of subscriber region S_i at

ListX[List_ID].Sub_Region[X1_Sub_Region_Counter];

Increment X1_Sub_Region_Counter;

Update the X1 subscriber region counter of ListX[List_ID];

// Processing the X2 subscriber region of federate F_i

List_ID = $(S_i)_{X_2}$;

Copy the X2 subscriber region counter of ListX[List_ID] into

X2_Sub_Region_Counter;

ListX[List_ID].X2_Sub_Region[X2_Sub_Region_Counter].Fed_ID = F_i;

Copy coordinates of subscriber region S_i at

ListX[List_ID].Sub_Region[X2_Sub_Region_Counter];

Increment X2_Sub_Region_Counter;

Update the X2 sub region counter of ListX[List_ID];

End

END Procedure

4.2.2 MCG Population Sub-Procedure

Initialization:

MCG_Counter = Counter for recording the number of Multicast groups *MCG* so far.

$(P_i)_{X_1}$ and $(P_i)_{X_2}$ are coordinates on X-axis for publisher region P_i.

$(S_j)_{X_1}$ and $(S_j)_{X_2}$ are coordinates on X-axis for subscriber region S_j.

BEGIN Procedure:

For each List element List_ID of ListX with Pub_Region_Counter > 0 do

Begin

For all publisher region P_i whose $(P_i)_{X_1}$ coordinate coincides with List_ID do

Begin

Note $(P_i)_{X_1}$ and $(P_i)_{X_2}$ coordinates of publisher region P_i;

Add Fed_ID of P_i to publisher region of MCG[MCG_Counter];

Increment the publisher region counter of MCG[MCG_Counter];

//Complete Overlap Condition

For all points j from 0 to $(P_i)_{X_1}$ on X-axis do

Begin

For all X1 subscriber region S_j at point j on X-axis do

Begin

Note $(S_j)_{X_1}$ and $(S_j)_{X_2}$ coordinates of subscriber region S_j;

If subscriber region S_j completely overlaps publisher region P_i do

Begin

Add Fed_ID of S_j to subscriber region of MCG[MCG_Counter];

Increment the subscriber region counter of MCG[MCG_Counter];

End

End

End

// $(P_i)_{X_1}$ to $(P_i)_{X_2}$ range Overlap Condition

For all points *j* from $(P_i)_{X_1}$ to $(P_i)_{X_2}$ on X-axis do

Begin

//Checking for $(S_j)_{X_1}$ overlap in range $(P_i)_{X_1}$ to $(P_i)_{X_2}$

For all X1 subscriber region S_j at point *j* on X-axis do

Begin

Note $(S_j)_{X_1}$ and $(S_j)_{X_2}$ coordinates of subscriber region S_j;

If subscriber region S_j overlaps publisher region P_i do

Begin

Add Fed_ID of S_j to subscriber region of

MCG[MCG_Counter];

Increment the subscriber region counter of

MCG[MCG_Counter];

End

End

//Checking if $(S_j)_{X_2}$ of any subscriber region fall in range

// $(P_i)_{X_1}$ to $(P_i)_{X_2}$

For all X2 subscriber region S_j at point *j* on X-axis do

Begin

Note $(S_j)_{X_1}$ and $(S_j)_{X_2}$ coordinates of subscriber region S_j;

If subscriber region S_j overlaps publisher region P_i do

Begin

Add Fed_ID of S_j to subscriber regions of MCG[MCG_Counter];

Increment the subscriber region counter of MCG[MCG_Counter];

End

End

End

If current multicast group (MCG[MCG_Counter]) has no subscriber region

Begin

Re-initialize publisher region and subscriber region count of MCG[MCG_Counter] to 0;

End

If current multicast group (MCG[MCG_Counter]) has subscriber region

Begin

Increment the multicast group counter; MCG_Counter;

End

End

End

END Procedure

4.2.3 MCG Pruning Sub-Procedure

Initialization:

$\left((P_i)_{X_1}, (P_i)_{X_2}, (P_i)_{Y_1}, (P_i)_{Y_2}\right)$ are coordinates of publisher region P_i.

$\left((S_j)_{X_1}, (S_j)_{X_2}, (S_j)_{Y_1}, (S_j)_{Y_2}\right)$ are coordinates of subscriber region S_j.

Y_Overlap: Flag indicating overlap condition.

DELETE_MCG: Flag indicating deletion of current multicast group.

BEGIN Procedure:

For all multicast group MCG_i do

Begin

Set Y_Overlap to FALSE;

Set DELETE_MCG to FALSE;

Note $(P_i)_{Y_1}$ and $(P_i)_{Y_2}$ coordinates of publisher region P_i;

For all subscriber region S_j of multicast group MCG_i do

Begin

Note $(S_j)_{Y_1}$ and $(S_j)_{Y_2}$ coordinates of subscriber region S_j;

//Complete Overlap Condition

If S_j completely overlaps P_i on Y-axis

Set Y_Overlap = TRUE;

//Partial Overlap condition on Y-axis

If P_i overlaps S_j on Y-axis

Set Y_Overlap = TRUE;

If Y_Overlap flag is FALSE

Begin

//Prune the subscriber region S_j

Decrement the subscriber region count of MCG_i;

Erase the Fed_ID of current subscriber region S_j ;

Reset Y_Overlap to FALSE;

End

End

If MCG_i has no subscriber region

Begin

Decrement publisher region count of current MCG_i;

Set DELETE_MCG flag to TRUE;

End

If MCG_i has no publisher region

Begin

Set DELETE_MCG flag to TRUE;

Delete current multicast group MCG_i;

End

If DELETE_MCG = TRUE

Begin

Delete current multicast group MCG_i ;

Decrement the total count for multicast groups;

End

End

END Procedure

4.3 Illustration of the P-Pruning Algorithm

Now, we will illustrate step-by-step execution of the P-Pruning algorithm using the example shown in Figure 5. The example consists of a two-dimensional routing space with size 10 x 10 units. As stated earlier, the set of federate $F = \{F_1, F_2, F_3\}$ such that federate F_1 has two publisher regions (P_{11}, P_{12}) and three subscriber regions (S_{11}, S_{12}, S_{13}). Federate F_2 has three publisher regions (P_{21}, P_{22}, P_{23}) and two subscriber regions (S_{21}, S_{22}). Federate F_3 has one publisher region P_{31} and one subscriber region S_{31}. This example demonstrates the region-matching calculation by P-Pruning algorithm for three federates. In practice, a distributed simulation can involve hundreds of federates.

In the following discussion, $(P_i)_{X_1}$ and $(P_i)_{X_2}$ are coordinates on X-axis for any publisher region P_i. $(S_j)_{X_1}$ and $(S_j)_{X_2}$ are coordinates on X-axis for any subscriber region S_j. We now walkthrough each step of the P-Pruning algorithm as it computes the multicast group.

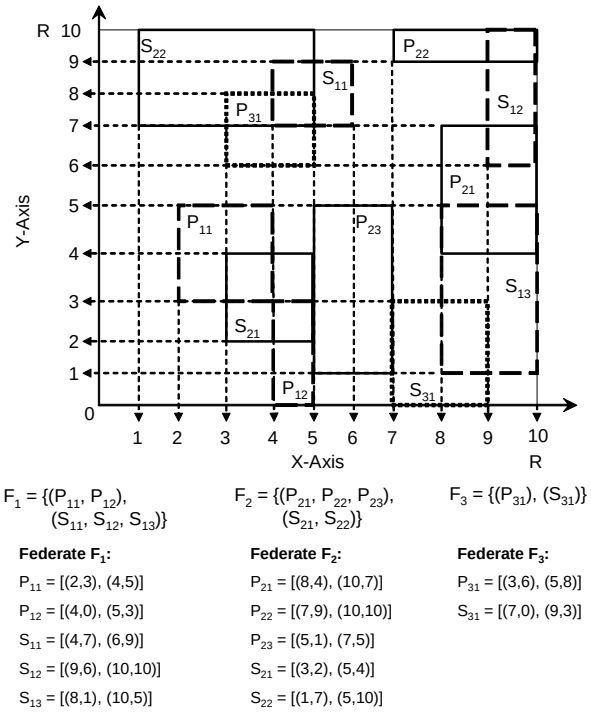


Figure 5. Routing space layout for illustration of P-Pruning DDM algorithm

List Computation Step: The ListX[1,2,3,...,10] array stores three important details for each point on X-axis: publisher region counter, X1 subscriber region counter, and X2 subscriber region counter. For a given point *x* in ListX: publisher region counter records the number of publisher regions, whose $(P_i)_{X_1}$ coordinate coincides with point *x*; X1 subscriber region counter records the number of subscriber regions, whose $(S_j)_{X_1}$ coordinate coincides with point *x*; and X2 subscriber region counter records the number of subscriber regions, whose $(S_j)_{X_2}$ coordinate coincides with point *x*.

In this step, the publisher and subscriber regions of each federate, F_1, F_2 and F_3, are examined. After this, the ListX element corresponding to the X_1 and X_2 coordinates of each publisher and subscriber region is updated. The state of ListX array at the end of this

step is shown in Table 1. The table shows (region counter, federate ID) pair for each entry. Entry '*' in this table indicates that there is no publisher or subscriber region, whose $(P_i)_{X_1}$ or $(S_j)_{X_1}$ or $(S_j)_{X_2}$ coordinate coincides with this point of ListX array. Row *Pub* indicates region counter = 1 for the points in ListX whose value coincides with $(P_i)_{X_1}$ coordinate of a publisher region, and it indicates 0 otherwise. The region counter for each point on ListX (in row *Pub*) also records the number of publisher regions, whose $(P_i)_{X_1}$ coordinate coincides with this point. So, it can be more than one. The federate ID corresponds to the identifier for the federate owning the publisher region P_i. Row *S_X1* indicates region counter = 1 for the points in ListX whose value coincides with $(S_j)_{X_1}$ coordinate of a subscriber region, and it indicates 0 otherwise. The region counter for each point on ListX (in row *S_X1*) also records the number of publisher regions, whose $(S_j)_{X_1}$ coordinate coincides with this point. So, it can be more than one. The federate ID corresponds to the identifier of the federate owning the subscriber region S_j. Row *S_X2* indicates region counter = 1 for the points in ListX whose value coincides with $(S_j)_{X_2}$ coordinate of a subscriber region, and it indicates 0 otherwise. The region counter for each point on ListX (in row *S_X2*) also records the number of publisher regions, whose $(S_j)_{X_2}$ coordinate coincides with this point. So, it can be more than one. The federate ID corresponds to the identifier for the federate owning the subscriber region S_j.

Table 1. Status of ListX array after List Computation step. Each entry has (region counter, federate ID) pair.

ListX	1	2	3	4	5	6	7	8	9	10
Pub	*	1, 1	1, 3	1, 1	1, 2	*	1, 2	1, 2	*	*
S_X1	1,2	*	1, 2	1, 1	*	1, 1	1, 3	1, 1	1, 1	*
S_X2	*	*	*	*	2, 2	1, 1	*	*	1, 3	2, 1

MCG Population Step: This step checks the overlapping status of publisher and subscriber regions based on the X-axis information. It also creates the multicast group *MCG* for the DDM.

This step scans the ListX entries, from the Table 1 shown above, that have at least one publisher region. A multicast group is assigned to each such entry in the beginning. All the subscriber regions that are not owned by the federate of the publisher region and overlapping with this publisher region are added to the multicast group. We describe the formation of two multicast groups MCG_1 and MCG_2 in detail.

ListX[2] has a publisher region entry in its *Pub* position, which corresponds to the publisher region P_{11}. So, the first multicast group MCG_1 is created with P_{11} as its member. This implies that federate F_1 (which owns P_{11}) is added to MCG_1 as the publishing federate. Since, P_{11} = [(2, 3), (4, 5)], the ListX array is scanned from the range of $(P_{11})_{X1}$ to $(P_{11})_{X2}$ (*i.e.*, 2 to 4) to check any $(S_j)_{X_1}$ coordinate of an overlapping subscriber region. The subscriber region, S_{21} = [(3, 2), (5, 4)], has S_X1 entry at ListX[3] in Table 1, which implies that S_{21} overlaps with P_{11}. Hence, S_{21} is added to multicast group MCG_1. In addition to this, the ListX array is also scanned from 0 to $(P_{11})_{X1}$ (*i.e.*, 0 to 2) to check $(S_j)_{X_1}$ coordinate of an overlapping subscriber region. The subscriber region

S_{22} = [(1, 7), (5, 10)] has S_X1 entry at ListX[1] in Table 1, which implies that S_{22} also overlaps with publisher region P_{11}. Hence, S_{22} is also added to multicast group MCG_1. Now, the multicast group MCG_1:P_{11} has two subscriber members: {S_{21}, S_{22}}. This implies that federate F_2 (which owns regions S_{21} and S_{22}) is added to MCG_1 as the subscribing federate. Finally, ListX array is scanned in range $(P_{11})_{X1}$ to $(P_{11})_{X2}$ (*i.e.*, 2 to 4) to check for any $(S_j)_{X_2}$ coordinate of an overlapping subscriber region. In this case, there are no such overlapping subscriber regions.

After this step, the next entry in ListX array having a publisher region entry in its *Pub* position is ListX[3], which corresponds to the publisher region P_{31}. So, the second multicast group MCG_2 is created with P_{31} as its member. This implies that federate F_3 (which owns P_{31}) is added to MCG_2 as the publishing federate. Since, P_{31} = [(3, 6), (5, 8)], the ListX array is scanned from the range of $(P_{31})_{X1}$ to $(P_{31})_{X2}$ (*i.e.*, 3 to 5) to check any $(S_j)_{X_1}$ coordinate of an overlapping subscriber region. The subscriber region, S_{21} = [(3, 2), (5, 4)], has S_X1 entry at ListX[3] in Table 1, which implies that S_{21} overlaps with P_{31}. Hence, S_{21} is added to multicast group MCG_2. The subscriber region, S_{11} = [(4, 7), (6, 9)], has S_X1 entry at ListX[4] in Table 1, which implies that S_{11} overlaps with P_{31}. Hence, S_{11} is also added to multicast group MCG_2. In addition to this, the ListX array is also scanned from 0 to $(P_{31})_{X1}$ (*i.e.*, 0 to 3) to check $(S_j)_{X_1}$ coordinate of an overlapping subscriber region. The subscriber region S_{22} = [(1, 7), (5, 10)] has S_X1 entry at ListX[1] in Table 1, which implies that S_{22} overlaps with publisher region P_{11}. Hence, S_{22} is also added to multicast group MCG_2. Now, the multicast group MCG_2:P_{31} has three subscriber members: {S_{11}, S_{21}, S_{22}}. This implies that federates F_1 (which owns region S_{11}) and F_2

(which owns regions S_{21} and S_{22}) is added to MCG_2 as the subscribing federate. Finally, ListX array is scanned in range $(P_{31})_{X1}$ to $(P_{31})_{X2}$ (*i.e.*, 3 to 5) to check for any $(S_j)_{X_2}$ coordinate of an overlapping subscriber region. In this case, there are no such overlapping subscriber regions.

Using the steps described above the P-Pruning algorithm creates four more multicast groups for this example. The final list of multicast groups is shown below.

MCG_1: $P_{11} = \{S_{21}, S_{22}\}$,

MCG_2: $P_{31} = \{S_{11}, S_{21}, S_{22}\}$,

MCG_3: $P_{12} = \{S_{21}, S_{21}, S_{22}\}$,

MCG_4: $P_{23} = \{S_{11}\}$,

MCG_5: $P_{22} = \{S_{12}, S_{13}, S_{31}\}$, and

MCG_6: $P_{21} = \{S_{12}, S_{13,} S_{31}\}$.

MCG Pruning Step: This step examines the overlap information of every region within all multicast groups on Y-axis and prunes any regions that do not overlap.

First, multicast group MCG_1: $P_{11} = \{S_{21}, S_{22}\}$ is examined. Since, publisher region P_{11} = [(2, 3), (4, 5)], we scan only the range from $(P_{11})_{Y1}$ to $(P_{11})_{Y2}$ (*i.e.*, 3 to 5). The subscriber region S_{21} = [(3, 2), (5, 4)] overlaps with P_{11} on Y-axis. However, S_{22} = [(1, 7), (5, 10)] with $(S_{22})_{Y1} = 7$ and $(S_{22})_{Y2} = 10$ does not overlap with P_{11} on Y-axis. Hence, it is pruned from MCG_1. The final composition this group is MCG_1: $P_{11} = \{S_{21}\}$.

The second multicast group MCG_2: $P_{31} = \{S_{11}, S_{21}, S_{22}\}$ has publisher region P_{31} = [(3, 6), (5, 8)]. The subscriber region S_{21} = [(3, 2), (5, 4)] does not overlap with P_{31} on Y-axis and hence, it is pruned from MCG_2. Subscriber regions S_{11} and S_{22} are retained in MCG_2

after verifying that they overlap with P_{31} on Y-axis. The final composition of MCG_2: P_{31} = {S_{11}, S_{22}}.

Similarly, the remaining multicast groups are also examined and pruned to have correct overlapping publisher and subscriber regions. For the multicast group MCG_4, the subscriber region S_{11} is pruned as it does not overlap with P_{23} on Y-axis. Since, there is no subscriber region in this multicast group, MCG_4 is deleted from the list of multicast groups.

The list of multicast groups created by the P-Pruning at the end of all three steps is as follows:

MCG_1: P_{11} = {S_{21}},

MCG_2: P_{31} = {S_{11}, S_{22}},

MCG_3: P_{12} = {S_{21}},

MCG_5: P_{22} = {S_{12}}, and

MCG_6: P_{21} = {S_{12}, S_{13}}.

Thus, using this example, we have demonstrated the formation of multicast groups in P-Pruning algorithm for two federates. The multicast groups created using P-Pruning algorithm can then be used by RTI for communication amongst federates.

4.4 Algorithm Analysis

The P-Pruning algorithm focuses on the computation of multicast groups right from the beginning. For the List Computation sub-procedure, complexity is O(n), where n is the number of federates in the distributed simulation. The MCG Computation sub-procedure runs for between O(n) and O(n^2) depending on the density of the regions within the routing space. For the MCG Pruning sub-procedure, complexity is O(n) times. The total

number of multicast groups in this algorithm is limited by O(n), which is significantly lesser than the number of multicast groups in fixed-grid and dynamic-grid algorithms. We prove this property in Section 4.6. In the next section, we present the computational complexity analysis of the P-Pruning algorithm in detail.

The P-Pruning algorithm is faster than region-matching, fixed-grid, and dynamic-grid DDM algorithms, as it avoid the quadratic computation step involved in these algorithms. By populating the multicast group, first only on the basis of X-axis information, and pruning the multicast group of unwanted subscriber regions in another step, it avoids the computational overheads of other algorithms.

4.5 Average-Case Computation Complexity Analysis of P-Pruning DDM Algorithm

We now present the complexity analysis of the P-Pruning algorithm. In particular, we prove its correctness and efficiency analytically through average-case analysis. We are using the average-case analysis as it is better representative of publisher and subscriber region distribution within the routing space.

Assumptions:

Let, F, P, and S be the set of federates, publisher regions and subscriber regions in the distributed simulation system, respectively. In our complexity analysis and simulation experiments, each federate F_i has one publisher region P_i and one subscriber region S_i, where i denotes the federate ID.

$n = |F|$ = Number of federates, and

$|P| = |S|$ = Number of publisher and subscriber regions.

R = Length of the x-dimension of routing space. For simplicity, we assume that the routing space is a square two-dimensional coordinate system.

MCG = Set of multicast groups and |*MCG*| is the number of multicast groups.

Each publisher region is identified by four coordinates (P_{x1}, P_{x2}, P_{y1}, P_{y2}) as shown in Figure 6. Similarly, each subscriber region is identified by four coordinates (S_{x1}, S_{x2}, S_{y1}, S_{y2}).

For all publisher regions:

$$P_{x1} < P_{x2} \text{ and } P_{y1} < P_{y2.}$$

For all Subscriber regions:

$$S_{x1} < S_{x2} \text{ and } S_{y1} < S_{y2.}$$

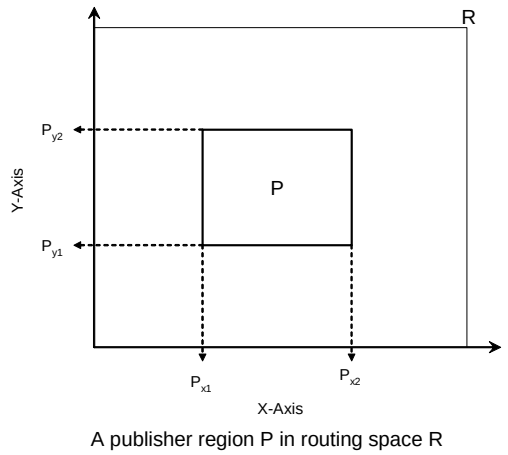


Figure 6. Routing space layout for a single publisher region *P*

In order to have distinct variable names and clear understanding, we will be denoting probability of random variable *x* as *Pr*[*x*]. First, we analyze the different overlap cases between publisher/subscriber regions. After this, we find the expected number of subscriber regions that can overlap with a given publisher region P_i on X-axis of the routing space.

For a publisher region, $P_{i,}$ P_{x1} and P_{x2} can fall in the range 0-R with equal probability.

Hence, for any coordinate x on X-axis,

$$\text{Probability } Pr[P_{x1} = x\ ] = \frac{1}{R}\text{, and}$$

$$\text{Probability } Pr[P_{x2} = x\ ] = \frac{1}{R}.$$

Similarly, we can see that for a subscriber region, $S_{j,}$ S_{x1} and S_{x2} will have probability

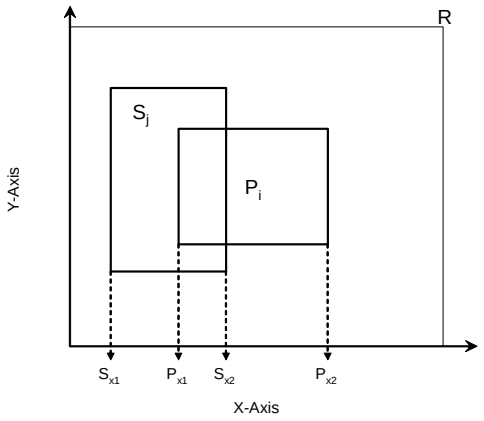
$$Pr[S_{x1} = x\ ] = \frac{1}{R}, \qquad \text{and} \qquad Pr[S_{x2} = x\ ] = \frac{1}{R}.$$

We need to find all subscriber regions that overlap with publisher region P_i and add them to multicast group MCG_i.

4.5.1 Analysis for Overlap Cases

Given a publisher region P_i and a subscriber region S_j, there are four distinct cases for overlap of P_i and S_j. We only consider X-axis overlap information for the MCG Population sub-procedure.

Case (a): In this case, as shown in **Figure 7**, subscriber region S_j overlaps publisher region P_i such that S_{x1} is less than P_{x1} and S_{x2} lies between P_{x1} and P_{x2}.



Case (a): Subscriber region S_j overlaps publisher region P_i

Figure 7. Region overlap analysis for case (a)

Using the formula,

$$Pr[a < x < b] = \sum_{x=a}^{b} Pr(x),$$

the probability that S_{x1} is less than P_{x1} is given as

$$\begin{aligned} Pr[S_{x1} < P_{x1}] &= Pr[0 < S_{x1} < P_{x1}] \\ &= \sum_{x=0}^{P_{x1}} Pr[S_{x1} = x] \\ &= \sum_{x=0}^{P_{x1}} \frac{1}{R} \\ &= \frac{P_{x1}}{R}. \end{aligned}$$

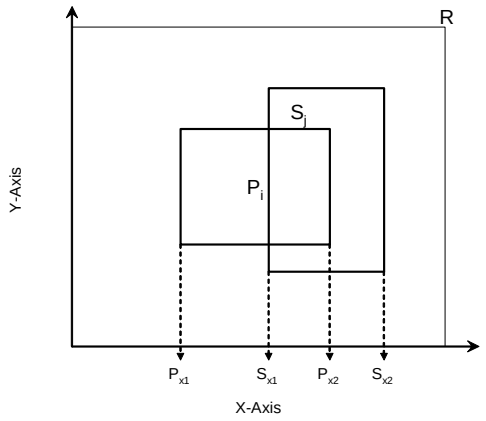
Also, the probability that S_{x2} lies between P_{x1} and P_{x2} is

$$\begin{aligned} Pr[P_{x1} < S_{x2} < P_{x2}] &= \sum_{x=P_{x1}}^{P_{x2}} Pr[S_{x2} = x] \\ &= \sum_{x=P_{x1}}^{P_{x2}} \frac{1}{R} \\ &= \frac{P_{x2} - P_{x1}}{R}. \end{aligned}$$

Therefore, the probability that a subscriber region S_j overlaps publisher region P_i in case (a) is

$$Pr[(S_{x1} < P_{x1}) \ AND \ (P_{x1} < S_{x2} < P_{x2})]$$
$$= Pr[(S_{x1} < P_{x1})] \text{ x } Pr[(P_{x1} < S_{x2} < P_{x2})]$$
$$(\because \text{Both are independent events})$$
$$= \left(\frac{P_{x1}}{R} \right) \text{x} \left(\frac{P_{x2} - P_{x1}}{R} \right).$$

Case (b): In this case, as shown in Figure 8, subscriber region S_j overlaps publisher region P_i such that S_{x2} is greater than P_{x2} and S_{x1} lies between P_{x1} and P_{x2}.



Case (b): Subscriber region S_j overlaps publisher region P_i

Figure 8. Region overlap analysis for case (b)

The probability that S_{x1} lies between P_{x1} and P_{x2} is given as

$$Pr[P_{x1} < S_{x1} < P_{x2}] = \sum_{x=P_{x1}}^{P_{x2}} Pr[S_{x1} = x]$$
$$= \sum_{x=P_{x1}}^{P_{x2}} \frac{1}{R}$$
$$= \frac{P_{x2} - P_{x1}}{R}.$$

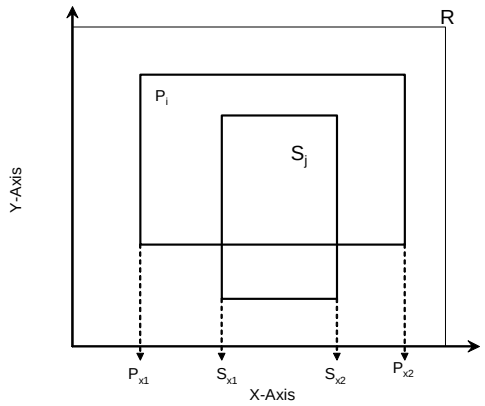
Also, the probability that S_{x2} is greater than P_{x1} is

$$Pr[P_{x2} < S_{x2}] = Pr[P_{x2} < S_{x2} < R]$$
$$= \sum_{x=P_{x2}}^{R} Pr[S_{x2} = x]$$
$$= \sum_{x=P_{x2}}^{R} \frac{1}{R}$$
$$= \frac{R - P_{x2}}{R}.$$

Thus, the combined probability for case (b) is

$$Pr[(P_{x1} < S_{x1} < P_{x2}) \ AND \ (P_{x2} < S_{x2} < R)]$$
$$= Pr[(P_{x1} < S_{x1} < P_{x2})] \text{ x } Pr[(P_{x2} < S_{x2} < R)]$$
$$(\because \text{Both are independent events})$$
$$= \left(\frac{P_{x2} - P_{x1}}{R}\right) \text{x} \left(\frac{R - P_{x2}}{R}\right).$$

Case (c): In this case, as shown in **Figure 9**, publisher region P_i completely overlaps subscriber region S_j such that S_{x1} is greater than P_{x1} and S_{x2} is less than P_{x2}.



Case (c): Publisher region P_i completely overlaps subscriber region S_j

Figure 9. Region overlap analysis for case (c)

The probability that S_{x1} is greater than P_{x1} and less than S_{x2} is given as

$$\begin{aligned} Pr[P_{x1} < S_{x1} < S_{x2}] &= \sum_{x=P_{x1}}^{S_{x2}} Pr[S_{x1} = x] \\ &= \sum_{x=P_{x1}}^{S_{x2}} \frac{1}{R} \\ &= \frac{S_{x2} - P_{x1}}{R}. \end{aligned}$$

Also, the probability that S_{x2} is greater than S_{x1} and less than P_{x2} is

$$\begin{aligned} Pr[S_{x1} < S_{x2} < P_{x2}] &= \sum_{x=S_{x1}}^{P_{x2}} Pr[S_{x2} = x] \\ &= \sum_{x=S_{x1}}^{P_{x2}} \frac{1}{R} \\ &= \frac{P_{x2} - S_{x1}}{R}. \end{aligned}$$

Thus, the combined probability for case (c) is

$$Pr[(P_{x1} < S_{x1} < S_{x2}) \ AND \ (S_{x1} < S_{x2} < P_{x2})]$$
$$= Pr[(P_{x1} < S_{x1} < S_{x2})] \ \text{x} \ Pr[(S_{x1} < S_{x2} < P_{x2})]$$
$$(\because \text{Both are independent events})$$
$$= \left(\frac{S_{x2} - P_{x1}}{R}\right) \text{x} \left(\frac{P_{x2} - S_{x1}}{R}\right).$$

Case (d): In this case, as shown in Figure 10, subscriber region S_j completely overlaps publisher region P_i such that S_{x1} is less than P_{x1} and S_{x2} is greater than P_{x2}.

The probability that S_{x1} is less than P_{x1} is given as

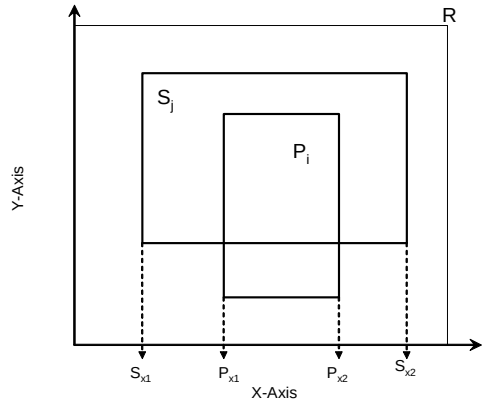
$$Pr[0 < S_{x1} < P_{x1}] = \sum_{x=0}^{P_{x1}} Pr[S_{x1} = x]$$
$$= \sum_{x=0}^{P_{x1}} \frac{1}{R}$$
$$= \frac{P_{x1}}{R}.$$

Also, the probability that S_{x2} is greater than P_{x2} is

$$Pr[P_{x2} < S_{x2} < R] = \sum_{x=P_{x2}}^{R} Pr[S_{x2} = x]$$
$$= \sum_{x=P_{x2}}^{R} \frac{1}{R}$$
$$= \frac{R - P_{x2}}{R}.$$

Thus, the combined probability for case (d) is

$$Pr[(0 < S_{x1} < P_{x1}) \ AND \ (P_{x2} < S_{x2} < R)]$$
$$= Pr[(0 < S_{x1} < P_{x1})] \text{ x } Pr[(P_{x2} < S_{x2} < R)]$$
$$(\because \text{Both are independent events})$$
$$= \left(\frac{P_{x1}}{R}\right) \text{x} \left(\frac{R - P_{x2}}{R}\right).$$



Case (d): Subscriber region S_j completely overlaps publisher region P_i

Figure 10. Region overlap analysis for case (d)

In all, we have *n* subscriber regions spread over *R* length, where *R* is the length of *X*-dimension of routing space. Hence, assuming a uniform distribution of regions over the routing space, the expected number of subscriber regions at any point on X-axis of the routing space is $\frac{R}{n}$. Each of these subscriber region has coordinate S_{x1} coinciding with a point *x* on X-axis. Therefore, the probability that any point on X-axis has a subscriber region S_j, whose S_{x1} coordinate coincides with this point is $\frac{1}{R/n}$.

The expected value of a random variable x is given as $E(x) = \sum x.Pr(x)$, where $Pr(x)$ is the probability of x. So, we can calculate the expected number of subscriber regions for each case, and then the total expected number of subscriber region overlapping with publisher region P_i for case (a) is given as:

$$
\begin{aligned}
E(case(a)) &= \sum x.Pr(x) \\
&= \sum \left[\begin{array}{r} (\text{Number of possibilities of case(a)}) \times \\ (\text{Probability of case(a)}) \end{array} \right] \\
&= \sum \left[\frac{1}{R/n} \times \left(\frac{P_{x1}}{R} \right) \text{x} \left(\frac{P_{x2} - P_{x1}}{R} \right) \right].
\end{aligned}
$$

Since, the entire X-dimension of routing space is used for summation, the limits of above sum is from 0 to R. Therefore, the above equation is

$$
\begin{aligned}
&= \sum_{i=0}^{R} \left[\frac{1}{R/n} \times \left(\frac{P_{x1}}{R} \right) \text{x} \left(\frac{P_{x2} - P_{x1}}{R} \right) \right] \\
&= R \times \frac{n}{R} \times \left(\frac{P_{x1}}{R} \right) \text{x} \left(\frac{P_{x2} - P_{x1}}{R} \right) \\
&= n \times \left(\frac{P_{x1}}{R} \right) \text{x} \left(\frac{P_{x2} - P_{x1}}{R} \right).
\end{aligned}
$$

For simplicity, we consider $n = R$. Thus, the expected number of subscriber regions in case (a) is given as:

$$E[case(a)] = \frac{P_{x1} \times (P_{x2} - P_{x1})}{R}.$$

Similarly, we can derive the average-case values for remaining cases. They are as follows:

$$E[case(b)] = \frac{(P_{x2} - P_{x1}) \times (R - P_{x2})}{R},$$

$$E[case(c)] = \frac{(S_{x2} - P_{x1}) \times (P_{x2} - S_{x1})}{R}, \text{ and}$$

$$E[case(d)] = \frac{P_{x1} \times (R - P_{x2})}{R}.$$

The overall average-case (expected value) for the number of subscriber regions overlapping with publisher region P_i is given as

$$\frac{1}{4}\begin{bmatrix} E[case(a)] + E[case(b)] + \\ E[case(c)] + E[case(d)] \end{bmatrix}$$

$$= \frac{1}{4R}\begin{pmatrix} [P_{x1} \times (P_{x2} - P_{x1})] + \\ [(P_{x2} - P_{x1}) \times (R - P_{x2})] + \\ [(S_{x2} - P_{x1}) \times (P_{x2} - S_{x1})] + \\ [P_{x1} \times (R - P_{x2})] \end{pmatrix}.$$

Since, all the terms in above equation are constants, the average number of subscriber regions overlapping with any publisher region P_i is $O(1)$, *i.e.*, constant.

Observation 1:

The MCG Population sub-procedure in Section 4.2.2 runs for n times and calculates the subscribe regions overlapping with the publisher regions at all coordinates. This step takes constant time on average. Hence, the average case complexity of MCG Population sub-procedure is $O(n)$.

Observation 2:

List Computation sub-procedure in Section 4.2.1 runs in O(n) time in both average and worst-case scenarios. This is because the publisher and subscriber regions of all federates are scanned only once.

Observation 3:

MCG Pruning sub-procedure in Section 4.2.3 prunes the unnecessary subscriber regions from multicast groups. It checks only the Y-axis coordinates of pre-existing subscriber regions. Since, there are $O(1)$ subscriber regions and $O(n)$ multicast groups, this procedure also takes $O(n)$ on average.

From observations 1, 2, and 3, we conclude that the P-Pruning algorithm takes $O(n)$ in average case and $O(n^2)$ in worst-case scenario.

4.5.2 Federate Distribution Analysis

We now analyze two more cases related to distribution of federates within the routing space. In the preceding section, we considered the case when number of federates is equal to the size of routing space. Here, we consider the remaining two possibilities.

Case (i): $n < R$

In this condition, the number of federates is significantly fewer than the length of the routing space. Hence, communication channels are relatively free and this facilitates message transfer among federates. Also, we have a sparse distribution of publisher and subscriber regions. Thus, the expected number of subscriber regions in overlap case (a) is given as

$$E[case\ (a)] = n \times \left(\frac{P_{x1}}{R} \right) \mathrm{x} \left(\frac{P_{x2} - P_{x1}}{R} \right).$$

The overall average-case (expected value) for the number of subscriber regions overlapping with publisher region P_i is given as

$$\frac{1}{4}\begin{bmatrix} E[case(a)] + E[case(b)] + \\ E[case(c)] + E[case(d)] \end{bmatrix}$$

$$= \frac{n}{4R^2}\begin{pmatrix} [P_{x1} \times (P_{x2} - P_{x1})] + \\ [(P_{x2} - P_{x1}) \times (R - P_{x2})] + \\ [(S_{x2} - P_{x1}) \times (P_{x2} - S_{x1})] + \\ [P_{x1} \times (R - P_{x2})] \end{pmatrix}$$

$$= \frac{n}{R^2} \times [\text{constant}] \cdot$$

Therefore, the average-case complexity for MCG population sub-procedure in Section 4.2.2 is $O(\frac{n^2}{R^2})$. Since, $n < R$, it implies that the DDM computations in the P-Pruning algorithm are faster in this case.

Case (ii): $n > R$

In this condition, the number of federates is significantly greater than the length of the routing space. Hence, communication channels are overloaded and this impedes message transfer among the federates. Also, we have dense distribution of publisher and subscriber regions. Thus, the expected number of subscriber regions in overlap case (a) is given as

$$E[case\ (a)] = n \times \left(\frac{P_{x1}}{R}\right) \text{x} \left(\frac{P_{x2} - P_{x1}}{R}\right).$$

The overall average-case (expected value) for the number of subscriber regions overlapping with publisher region P_i is given as $\frac{n}{R^2} \times [\text{constant}]$.

Therefore, the average-case complexity for MCG Population sub-procedure in Section 4.2.2 is $O(\frac{n^2}{R^2})$. Since, $n > R$, this implies that the DDM computations in the P-Pruning algorithm will take $O(n^2)$ time.

4.6 Size of Multicast Group Analysis

We now prove that P-Pruning algorithm is efficient in terms of space required to represent the multicast groups.

Lemma: The P-Pruning DDM algorithm requires less memory space to store the multicast groups.

Proof: The fixed-grid algorithm creates a multicast group for each grid cell. If the dimension of each grid cell is $a \times a$, then the number of multicast group is $\left(\frac{R \times R}{a \times a}\right)$.The dynamic-grid algorithm requires fewer memory space than $\left(\frac{R \times R}{a \times a}\right)$ because it assigns multicast groups to only those cells that have at least one overlapping subscriber and one publisher region. However, dynamic-grid algorithm does not save significant memory. If the size of grid cell is increased, then it requires less memory, but at the cost of accuracy. The region-matching algorithm requires $n = |F|$ multicast groups, where n is the number of federates in the distributed simulation. The P-Pruning DDM algorithm first assigns n memory space for the multicast groups, and then prunes the unwanted multicast groups which have publisher and subscriber regions that do not overlap on Y-axis. Hence, the total space required to store the multicast groups in the P-Pruning DDM algorithm is fewer than the other three algorithms.

4.7 Extending the P-Pruning DDM to multidimensional routing space

A region represents interests. Region = {(X_L, Y_L, Z_L), (X_H, Y_H, Z_H)}. Thus, a publisher region P_{ii} is represented as {(X_{iiL}, Y_{iiL}, Z_{iiL}), (X_{iiH}, Y_{iiH}, Z_{iiH})}.

The three-dimensional P-Pruning algorithm assumes that all the 3-D coordinates are present.

Input: Federate F = {F_1, F_2, F_3, …, F_n}. Every federate F_i has publisher regions {P_{i1}, P_{i2}, …, P_{ix}}. Every F_i has subscriber regions {S_{i1}, S_{i2}, …, S_{ix}}.

Output: Multicast group MCG = {MCG_1, MCG_2, …, MCG_n} such that MCG_i: P_j = {S_1, S_2, …, S_k}.

First, we project all regions on the ListX array and compute the multicast groups MCG using the MCG Population procedure. Then, we apply the MCG pruning process successively first on the Y-axis and then on the Z-axis. Here, we only show the MCG Pruning procedure on Z-axis here.

MCG Pruning on Z-axis Procedure

BEGIN Procedure

For all multicast groups MCG_i do

Begin

P_i is the publisher region in multicast group MCG_i;

For all subscriber regions S_j in MCG_i do

Begin

If subscriber region S_j does not overlap publisher region P_i on Z-axis of routing space

Begin

Delete subscriber region S_i from MCG_i;

Decrement the subscriber region count of MCG_i;

End

End

If MCG_i has no subscriber region

Begin

Delete current multicast group MCG_i;

Decrement the multicast group counter in MCG;

End

End

END Procedure

4.8 Dynamic P-Pruning Algorithm

We now describe the extension of P-Pruning algorithm to dynamic situations where federates can join and leave multicast groups. In this approach, the multicast groups

produced by P-Pruning algorithm are corrected whenever a federate joins or resigns from the routing space.

4.8.1 Federate Join and Resign Procedure at Run-Time

Federate Join Procedure

Input: Federate F_i (with federate ID Fed_ID) joining the federation. Federate F_x has publisher regions $\{P_{i1}, P_{i2}, \ldots, P_{ix}\}$ and subscriber regions $\{S_{i1}, S_{i2}, \ldots, S_{ix}\}$.

Output: Updated multicast group MCG.

BEGIN Procedure

For all publisher regions P_i of federate F_i do

Begin

Create Multicast group MCG_{x+1}: P_{ix}

Find all subscriber regions overlapping with publisher region P_{ix} and add them to MCG_{x+1}

End

For all subscriber regions S_{ix} of federate F_i do

Begin

Scan all publisher regions P_{ix} in multicast groups to check if S_{ix} and P_{ix} overlap

If (Overlap Scan = TRUE) Then add S_{ix} to MCG:P_{ix}

End

END Procedure

Federate Resign Procedure

Input: Federate F_i (with federate ID Fed_ID) resigns from the federation. Federate F_x has publisher regions $\{P_{i1}, P_{i2}, \ldots, P_{ix}\}$ and subscriber regions $\{S_{i1}, S_{i2}, \ldots, S_{ix}\}$.

Output: Updated multicast group MCG.

BEGIN Procedure

For all multicast groups MCG_i do

Begin

If Fed_ID of publisher region in MCG_i = Fed_ID of F_i

Begin

Delete multicast group MCG_i

End

Else

Begin

//Check if any subscriber regions in MCG_i are owned by federate F_i

If Fed_ID of any subscriber region in MCG_i = Fed_ID of F_i

Begin

Delete subscriber region S_i from MCG_i

End

End

End

END Procedure

CHAPTER FIVE: PERFORMANCE EVALUATION OF DDM ALGORITHMS

In this Chapter, we describe the performance evaluation of the P-Pruning DDM algorithm against three other algorithms: region-matching, fixed-grid, and dynamic-grid DDM algorithms. The simulations were implemented in C++ on Windows XP running on a Pentium IV 3 GHz PC. We used object-oriented class structures to represent federates, their publisher and subscriber regions, the grid cells and the multicast groups.

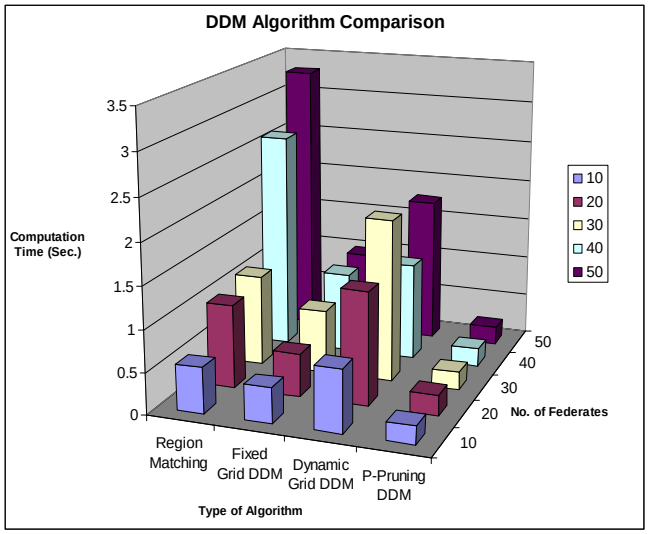


Figure 11. Performance evaluation of P-Pruning algorithm for routing space 50 x50 and grid size 2 x 2

5.1 Implementation Details

We compared the four DDM algorithms using three performance criteria: computation time, run-time memory usage, and number of multicast groups required. In our simulation experiments, we generated federates with publisher and subscriber regions,

whose coordinates were randomly distributed within the routing space. Each federate F_i has one publisher region P_i and one subscriber region S_i. In the graphs from Figure 11 through Figure 19, we have shown the results for three set of distributed simulation environment: 50 x 50 routing space with 2 x 2 grid cells, 50 x 50 routing space with 5 x 5 grid cells, and 100 x 100 routing space with 5 x 5 grid cells. The grid cell dimensions are applicable only for the fixed-grid and dynamic-grid algorithms. The number of federates in this simulation environment was increased from 10 to 50 for 50 x 50 routing space and from 20 to 40 for the 100 x 100 routing space. In all the graph charts, the data corresponding to DDM algorithm is referred as *P-Pruning DDM*. In Chapter 6, we discuss the impact of system resources such as memory constraints on data distribution strategies. To ensure consistency in the simulation results, all DDM algorithms access the same federates at run-time.

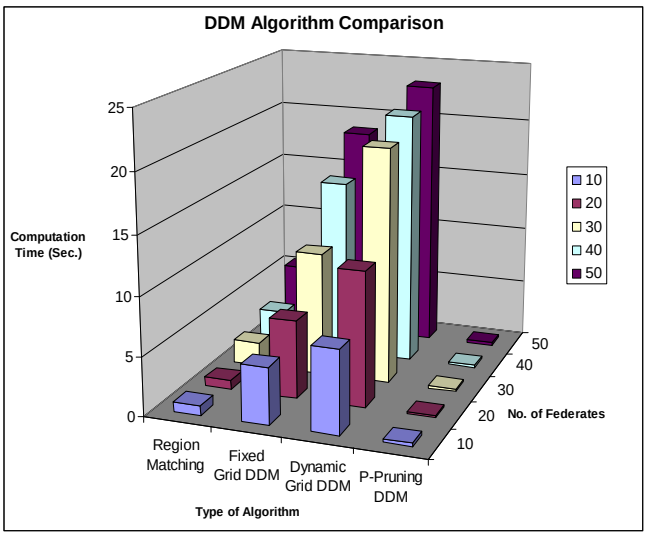


Figure 12. Performance evaluation of P-Pruning algorithm for routing space 50 x50 and grid size 5 x 5

The graphs in Figures 11, 12, and 13 show the comparison of computation time required for different DDM algorithms. The routing space is 50 x 50 in Figure 11 and 12, while it is 100 x 100 for Figure 13. The grid size is 2 x 2 for Figure 11, and 5 x 5 for Figures 12 and 13. The results show that the P-Pruning DDM algorithm computes the multicast groups for DDM faster than any of the three algorithms.

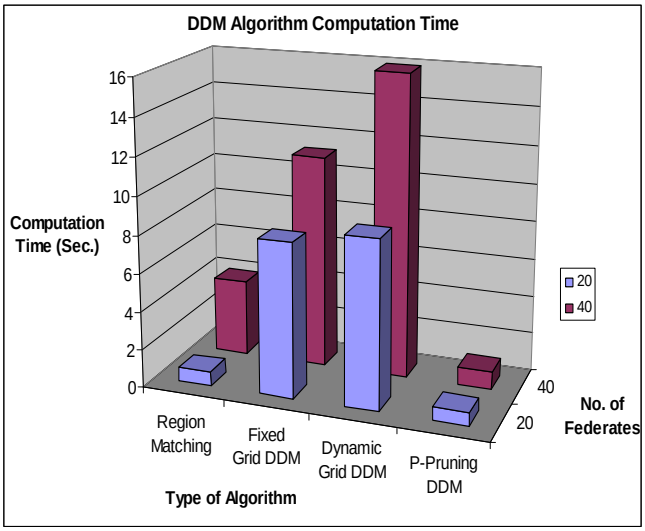


Figure 13. Performance evaluation of the P-Pruning algorithm for routing space 100 x 100 and grid size 5 x 5

We compared the memory usage at run-time for the four algorithms and the results are shown in Figures 14, 15, and 16. The routing space is 50 x 50 in Figure 14 and 15, while it is 100 x 100 for Figure 16. The grid size is 2 x 2 for Figure 14, and 5 x 5 for Figures 15 and 16. The graphs show that the P-Pruning DDM algorithm used system memory more efficiently as compared to the other three algorithms. From our simulation experience, we learned that memory management is very critical in distributed

simulations. Chapter 6 discusses the memory constraint management issues in more detail.

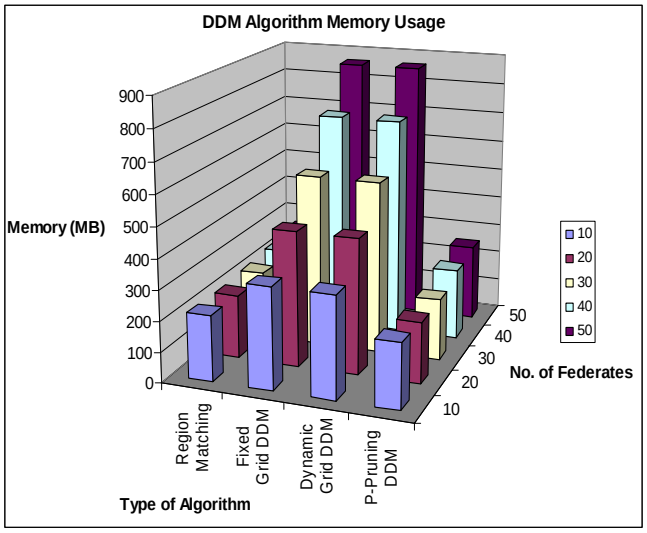


Figure 14. Comparison of memory usage by DDM algorithms for routing space 50 x 50 and grid size 2 x 2

All DDM algorithms provide the multicast groups as an output. The size of the multicast groups is an important metric for evaluating the performance of DDM algorithms. The graphs in Figures 17, 18, and 19 show the comparison of DDM algorithms in terms of size of multicast groups required to provide the region overlap information. The routing space is 50 x 50 in Figure 17 and 18, while it is 100 x 100 for Figure 19. The grid size is 2 x 2 for Figure 17, and 5 x 5 for Figures 18 and 19. The results show that the P-Pruning DDM algorithm requires significantly fewer multicast groups as compared to fixed-grid and dynamic-grid algorithm.

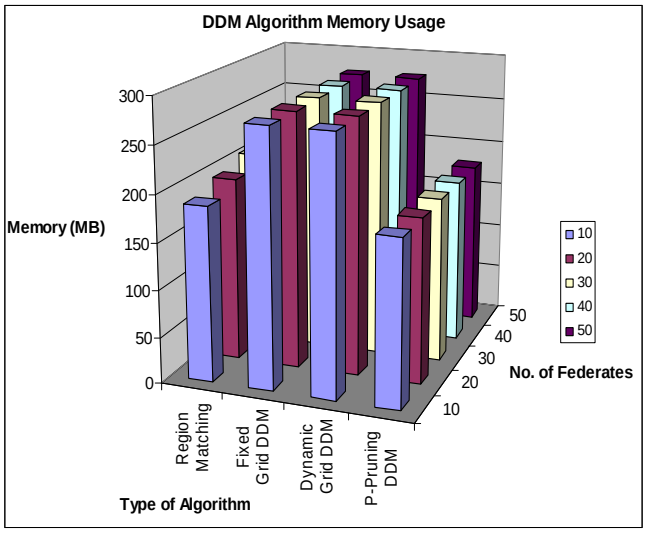


Figure 15. Comparison of memory usage by DDM algorithms for routing space 50 x 50 and grid size 5 x 5

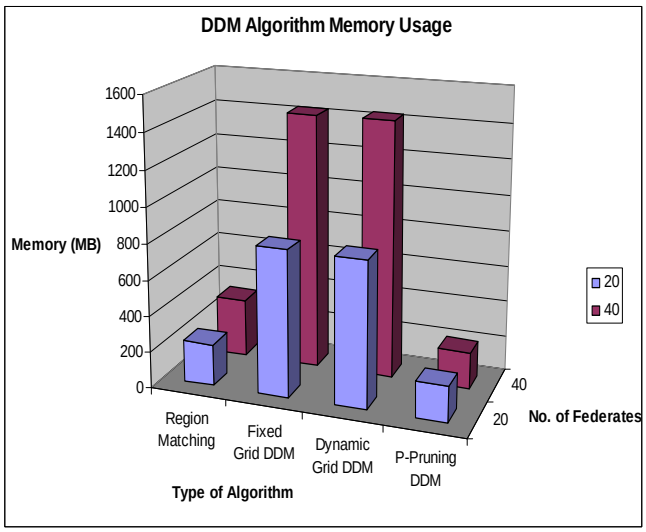


Figure 16. Comparison of memory usage by DDM algorithm for routing Space 100 x 100 and grid size 5 x 5

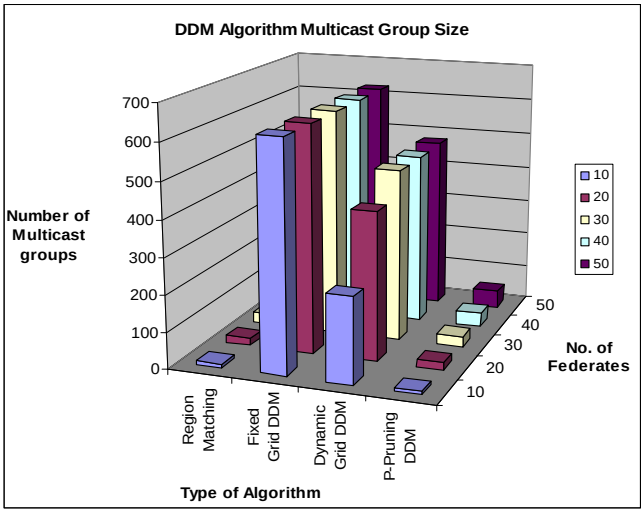


Figure 17. Comparison of multicast group size in DDM algorithms for routing space 50 x 50 and grid size 2 x 2

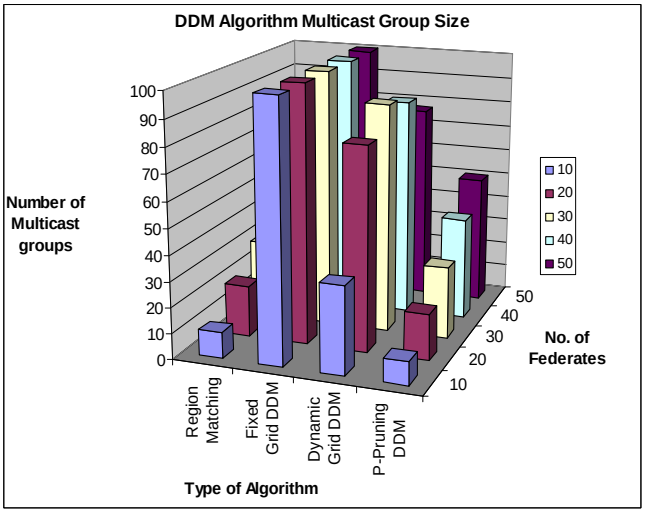


Figure 18. Comparison of multicast group size in DDM algorithms for routing space 50 x 50 and grid size 5 x 5

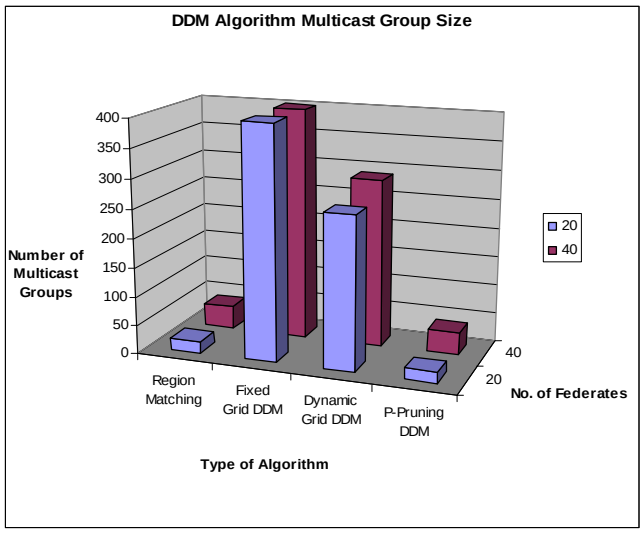


Figure 19. Comparison of multicast group size in DDM algorithms for routing space 100 x100 and grid size 5 x 5

5.2 Simulation Results Analysis

In this sub-section, we analyze the simulation results from the performance evaluation of DDM algorithms. We found that the P-Pruning DDM algorithm provides the region overlapping information efficiently with respect to three important metrics: computation time, memory usage at run-time, and size of multicast groups. In the performance-evaluation simulation, the region-matching algorithm is an exact algorithm with quadratic time complexity, while the grid-based (fixed and dynamic) algorithms are approximate heuristics. The P-Pruning DDM algorithm is an exact algorithm, and it outperforms both exact and approximate class of DDM algorithms.

We also found that the density of federates in the routing space and the variation of grid sizes affects the performance of all DDM algorithms. The performance of fixed-grid and dynamic-grid algorithm deteriorates with increase in the number of federates more severely as compared to region-matching and the P-Pruning DDM algorithm.

However, if we increase the size of grid cells, the grid algorithms become fast, but at the cost of accuracy [99]. The P-Pruning algorithm does not suffer from this constraint. Hence, it is both efficient and accurate. The computations in all DDM algorithms can be extremely memory intensive. Therefore, system memory can become a huge constraint on the performance. Also, managing the communication overhead with increase in number of federates--and their overlapping regions--is very crucial for scalable distributed simulation. We discuss these important aspects of DDM research in the next Chapter.

CHAPTER SIX: RESOURCE CONSTRAINT MANAGEMENT IN DISTRIBUTED SIMULATION

In this Chapter, we present the design and implementation of a resource-efficient enhancement to the *P-Pruning algorithm*. We also present a performance evaluation study in a memory-constraint environment.

6.1 Memory as a Resource

The memory system of current machines is composed of several levels:

Memory = L1 Cache (On-Chip) + L2 Cache (Secondary) + RAM (Main Memory) + Disk

A cache provides temporary storage that can be accessed quicker than RAM. By placing computationally intensive portions of a program in the cache, the processor can avoid the overhead involved in continuous access of RAM. L1 cache is a storage space that is located on the processor itself, while L2 cache is typically a RAM chip outside the processor (*e.g.*, the Intel Pentium 4 features a 256 or 512KB L2 advanced transfer cache). In this hierarchy, memory gets larger and slower as it gets further away from the processor. A typical access time to internal main memory (RAM) is in the order of nanoseconds, while access time to external memory (such as hard disk) is in the order of milliseconds. Thus, the access times of internal and external memory differ by a factor of million. In many large-scale distributed simulation applications, the communication between internal and external memory, and not the internal computation time, is actually bottleneck in the computation. Also, as the application size is scaled, the Input/Output (I/O) requirements can lead to serious memory crunch. Modern operating systems use sophisticated paging and data pre-fetching strategies to minimize the effect of I/O

bottleneck and ensure that the accessed data is present in the internal memory. However, these strategies are general in nature and cannot exploit the properties of a specific problem. Hence, we need to design solutions which consider a memory of limited size.

In memory-constraint approach, we view the system memory as a resource that has to be optimally allocated among the processes. The problem is how to deploy efficient data-structures and reorganize the data at run-time so that the DDM computation is not as memory intensive as encountered in practical simulations. I/O efficient data-structures are the key tools in developing a resource-efficient approach. Also, dynamic memory-management strategy that provides efficient garbage collection to reduce unnecessary memory leak at run-time is crucial. The primary motivation in the resource-constraint approach is to devise a scalable, memory-efficient solution for high-performance distributed simulation applications.

6.2 I/O Efficient Resource-Constraint Strategy for DDM

In this section, we explore the resource-constraint issues in the DDM algorithms and present a memory-efficient enhancement to the P-Pruning algorithm.

6.2.1 Resource-Constraint Issues in DDM Implementation

In Chapter 5, we compared the performance of P-Pruning algorithm with region-matching, fixed-grid and dynamic-grid DDM algorithm through simulation studies. During the simulation experiments, it was observed that the performance of DDM algorithms is adversely affected as the number of federates is increased in the simulation environment. In practice, system scalability can be seriously inhibited by limits on

bandwidth and computation. While this is not totally unexpected; for a DDM algorithm to be effective and deployable in high performance modeling and simulation applications, it must be scalable. In general, the performance of all DDM algorithms is severely affected by limitations in system resources such as communication bandwidth, memory, and CPU availability. Hence, we have considered the system memory as a resource in this research. In practical distributed simulation applications, the designers should deploy efficient data structures to achieve the dual goal of reducing computation time and memory utilization.

6.2.2 A Memory-Efficient Strategy for Data Distribution Management

The P-Pruning algorithm is not resource-efficient because it does not conserve memory. In a resource-constraint environment, the system memory is limited and special routines are needed for developing scalable solutions. We now present a memory-efficient enhancement to the P-Pruning algorithm. We consider the system memory as a resource and modify the P-Pruning algorithm for optimal utilization of this resource. In memory-efficient P-Pruning algorithm, the List Computation Sub-Procedure in Chapter 4 is modified by incorporating a resource-efficient data structure. We define a node which maintains three different types of lists: Publisher region list, X1 subscriber region list, and X2 subscriber region list. The set of node is represented as list which replaces the ListX array in the List Computation sub-procedure. The set of nodes can be viewed as disjoint set of forests, where each node stores three different trees. This representation reduces the memory allocated at run-time significantly for the DDM computation and

also improves the computation time as evident from the performance evaluation results in next sub-section.

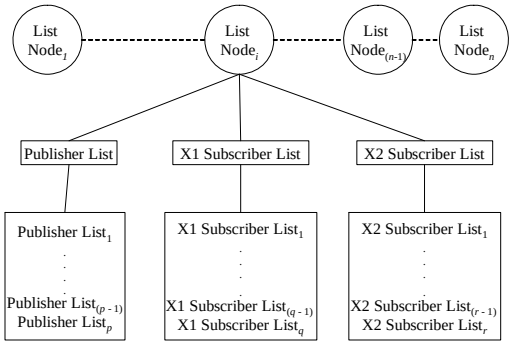


Figure 20. Representation of memory-efficient data structure

Data Structure Design: The structure of each node in the disjoint set is shown in Figure 20. There are *n* nodes in the list, and each node maintains three different lists of size *p*, *q*, and *r*. Here, *p* = number of publisher regions; *q* = number of X1 subscriber regions; and *r* = number of X2 subscriber regions. The list node is represented using the class structure shown in Figure 20. The three lists in disjoint set are populated in List Computation sub-procedure and the multicast groups are built using the disjoint set in the MCG population sub-procedure. Using the new structure, we can reduce the memory allocated at run-time and reduce the access time during computations.

6.3 Performance Evaluation of Resource-Constraint P-Pruning Algorithm

In this section, we describe the performance evaluation of the P-Pruning DDM and *Memory-Constraint* P-Pruning algorithm. The study was aimed at modeling high-

performance distributed simulation scenario and implemented in C++ under Windows XP running on a Pentium IV 2.8 GHz PC with 512 MB RAM and 2500 MB virtual memory. We used object-oriented class structures. as shown in Figure 21, to represent federates, their publisher and subscriber regions, the grid cells and the multicast groups.

```
class List_Node
{ public:
        vector<Region> Pub_Region;
        vector<Region> X1_Sub_Region;
        vector<Region> X2_Sub_Region;
        List_Node();
        ~List_Node();
};
```

Figure 21. Class structure to represent the disjoint set of forest

6.4 Simulation Implementation and Analysis

In the simulation experiments, we generated federates with publisher and subscriber regions, whose coordinates were randomly distributed within the routing space. Each federate F_i has one publisher region P_i and one subscriber region S_i. The graph in Figure 22 shows the comparison of memory utilized at run-time by the *Memory-Constraint* P-Pruning and P-Pruning algorithms for distributed simulation having routing space of 4,000 x 4,000 and number of federates ranging from 100 to 4,000. Figure 23 shows the comparison of computation time required for the *Memory-Constraint* P-Pruning and conventional P-Pruning implementation for the similar range of routing space and

number of federates in simulation environment. It is evident from these graphs that the *Memory-Constraint* version uses constant memory as compared to the P-Pruning algorithm. It also requires less computation time.

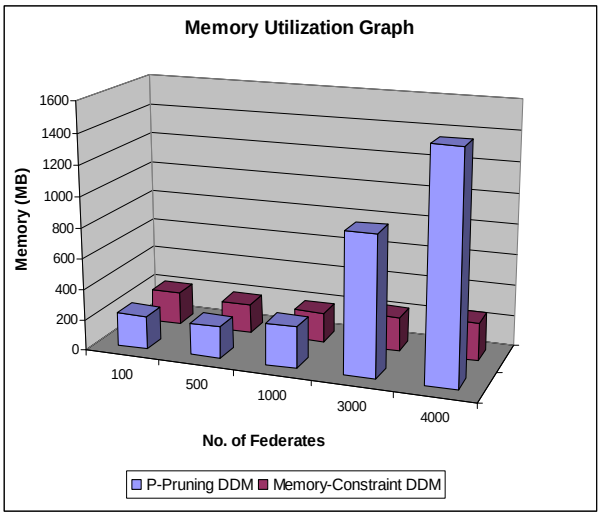


Figure 22. Comparison of memory utilization by the Memory-Constraint and P-Pruning DDM algorithms

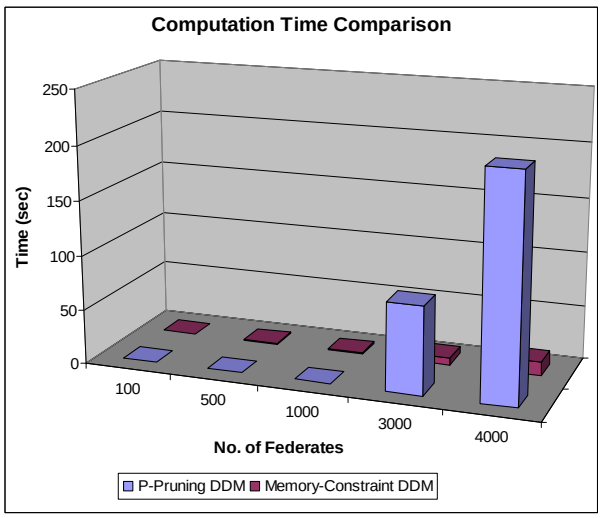


Figure 23. Comparison of computation time for routing space from 100 x 100 to 4000 x 4000

The graph in Figure 24 shows the memory utilization for the routing space upto 20000 x 20000 and the number of federates ranging from 100 to 20,000. This result demonstrates the scalable nature of *Memory-Constraint* P-Pruning algorithm. The P-Pruning algorithm could simulate only upto 4,000 federates due to inefficient memory utilization at run-time.

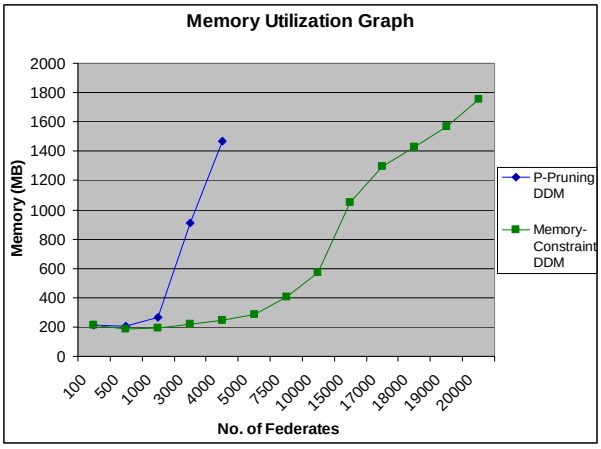


Figure 24. Memory utilization for distributed simulation with 20,000 federates

From the performance evaluation study of two versions of the P-Pruning DDM algorithm, it is clear that the *Memory-Constraint* P-Pruning DDM algorithm provides the region overlapping information efficiently with respect to important metrics: computation time and memory usage at run-time. The list of disjoint forests minimizes I/O requirements and optimizes memory access at run-time.

6.5 Summary of Memory-Efficient Approach

In this Chapter, we presented the design and performance evaluation of a resource-efficient enhancement to the P-Pruning algorithm for DDM. By deploying efficient data structures, the resource-constraint P-Pruning DDM algorithm scales well in high performance distributed-simulation environment. It shows better performance in terms of computation time and memory usage at run-time in simulation environment.

CHAPTER SEVEN: INTEGRATION OF THE P-PRUNING DDM ALGORITHM IN FDK

In this Chapter, we describe the integration of the P-Pruning algorithm with FDK. FDK is an implementation of HLA architecture developed at Georgia Institute of Technology. It has been widely used by researchers in academia, industry, and government laboratories as an effective software package for evaluating their research contributions to distributed simulation technology. We first provide an overview of the FDK architecture with emphasis on the DDM component. After this, we discuss the issues with FDK DDM module and areas of improvement. Then, we describe the integration of the P-Pruning algorithm with FDK and provide results of our experiences. We also describe the enhancements made to FDK from its existing HLA 1.3 specification to the IEEE 1516 standard for DDM implementation. Finally, we provide the concluding remarks.

7.1 An Overview of FDK Architecture

Federated Simulations Development Kit (FDK) is an open source implementation of HLA-based RTI software system developed at Georgia Institute of Technology. It has been used by researchers in academia, industry, and government laboratories as an effective software package for evaluating their research contributions to distributed simulation technology.

FDK contains composable modules for building run-time infrastructures (RTI) using which different simulations can be integrated together. RTI-Kit, a principal component of FDK, is a collection of libraries. It supports development of Run-Time

Infrastructures for parallel and distributed simulation systems, especially federated simulation systems running on high performance computing platforms. Figure 25 and Figure 26 show an architectural overview of FDK and its interconnection with the Federate and the underlying network.

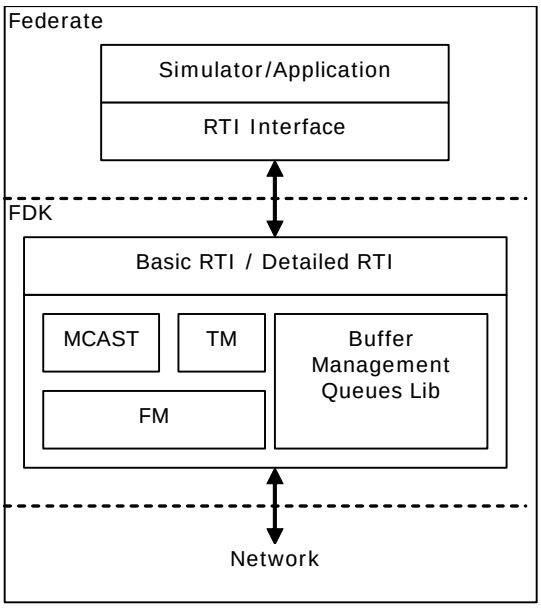
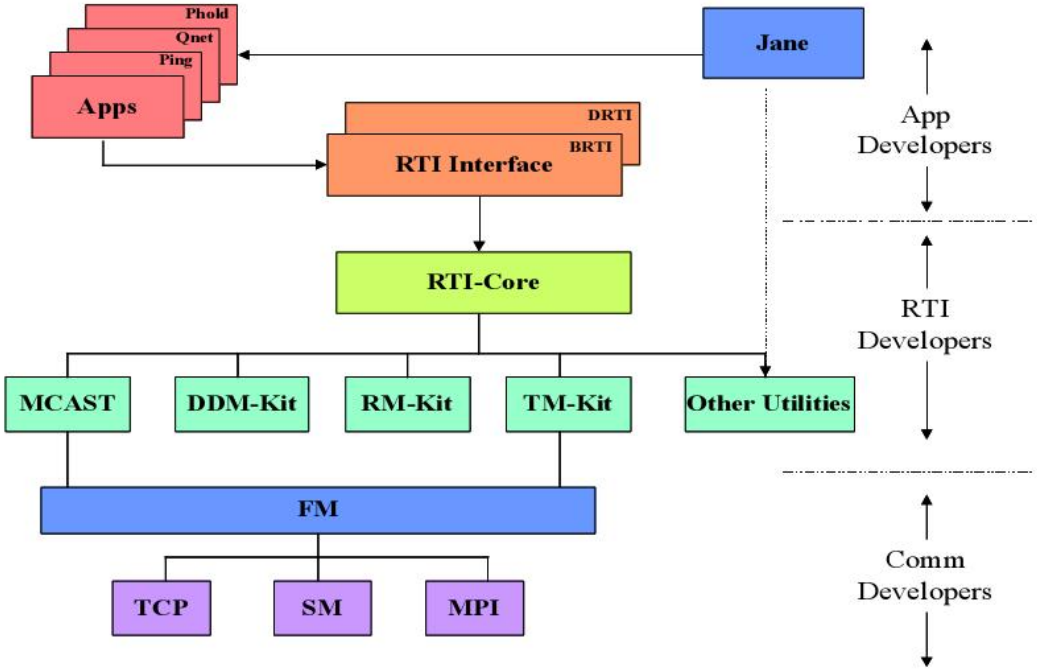


Figure 25. Architectural Overview of FDK

FDK enables its users to develop the functions of RTI, and more importantly, provides developers with the flexibility to expand the functionality. FDK is designed so that RTI developers can pick and choose from the set of FDK modules that are most appropriate for developing their particular RTI implementation. Each library can be used separately, or together with other RTI-Kit libraries, depending on the functionality required by the user. Because each library is designed as a stand-alone component, RTI implementations that are constructed using RTI-Kit are highly modular, with clear, well-

defined (and documented) interfaces. These libraries can be embedded into existing RTI to add new functionality. RTI developers can benefit from incorporating these ready-made modules, and avoid having to develop them on their own. Thus, FDK is a modular and reusable set of libraries designed to facilitate the development of RTI for developing



or integrating parallel and distributed simulation systems.

Figure 26. FDK Architecture (source: FDK user Manual)

The RTI-Kit consists of the following modules all of which implemented in C/C++:

- Buffer Management and Queues Library.
- Time Management Kit (TM).
- Multicast Kit (MCAST).

- Fast Messages (FM).

Buffer Management is a common module for the MCAST-Kit (Multicast), TM-Kit (Time Management) and FM-Lib (Fast Messages) modules for the management of buffers and queues. The Fast Messages module, FM-Lib, is a low-level messaging layer. Fast Messages is designed to enable convenient and high performance layering of other APIs and protocols on top of it. MCAST-Kit handles the management of multicast groups including communication in multicast groups. The Time Management (*TM*) module, TM-Kit, provides basic primitives for synchronizing events in HLA distributed simulations. It also enables message delivery in both Receive Order (RO) and Time Stamp Order (TSO).

In addition to the RTI-Kit, FDK contains two HLA Interface Specification complaint RTI implementations:

- Baby RTI (BRTI), C implementation.
- Debbie's RTI (DRTI), C++ implementation.

While these two implementations are not complete realization of the HLA Interface Specification, sufficient HLA services are already in place for simple simulations and benchmarking. The MCAST and FM modules are of particular interest in this paper. The MCAST module is responsible for the management of multicast groups and group communications while the FM module provides the low-level primitives for communications on the underlying network.

7.1.1 HLA Functional Components Implemented in FDK

The HLA Interface Specification defines sets of services to support realization of distributed simulations. The Runtime Infrastructure (RTI) in HLA is software that implements those services.

The DRTI software implements services in five of the categories defined in the HLA Interface Specification:

FEDERATION MANAGEMENT: These services initialize the execution of the federation.

The joinFederationExecution service is used to initialize the RTI and to define the object classes and interaction classes that are valid for the federation. No other RTI function should be invoked prior to making this call. Constants indicating the number of federates (RTIKIT_numnodes) and the ID of this federate (RTIKIT_nodeid) are undefined until this procedure is called. Note that all federates joining the federation must supply identical fed files. Otherwise, the handles assigned to go with the names will not match, and the federation will not work correctly.

DECLARATION MANAGEMENT: These services define object and interaction classes and set up communications between federates using a newsgroup-like publish/subscribe paradigm.

The declaration management services are used to specify those object and interaction classes for which a federate intends to send messages, and specifies those classes for which a federate desires to receive messages. Some means is required to specify which

federates are to be notified when an interaction is sent, an object instance is created, or an attribute of an object instance is updated. A publish/subscription mechanism not unlike Internet newsgroups is used for this purpose. Specifically, the HLA uses something called "class-based filtering." This means a federate can subscribe to receive all updates to all instances of objects of a certain class. For example, to get a message whenever any tank moves, a federate can subscribe to the Tank class. Note there is no mechanism to only get updates for a specific object instance, only for all objects of some class. DDM mechanisms are defined in the HLA for this type of data filtering.

OBJECT MANAGEMENT: These services allow federates to declare object instances, update attributes, send interactions, receive updates to attributes, and receive interactions produced by other federates.

The object management services are used to transmit messages between federates. A message will be sent whenever a federate creates (registers) an instance of an object class, updates the attributes of an object instance, or sends an interaction. Callbacks are used to receive messages. When the RTI is ready to deliver a message to the federate, it calls one of the methods of the FederateAmbassador class. FederateAmbassador is an abstract class. This means the federate developer has to derive a class from FederateAmbassador and provide definitions for each of the methods. When a federate creates an instance of an object (of some class specified in the FedFile), it must notify the RTI of this fact by registering this instance. When an instance is registered, the RTI returns a handle for the object instance that is used in future references to it, e.g., to update attributes of the object. The type ObjectHandle denotes a reference to an object instance.

TIME MANAGEMENT**:** These services control the advancement of simulation time within each federate, and prevent federates from receiving messages in their past (i.e., time stamp less than the federate's current simulation time).

The time management services ensure that messages sent with a time stamp are delivered to each federate in time-stamp order, and that no federate receives a time-stamped message in its past, i.e., a message with time stamp less than the federate's current simulation time. In the current implementation, updates and sends that include a time stamp are delivered in time-stamp order. All other messages are delivered in receive order. Time management is implemented by services where the federate requests that its simulation time be advanced, and the RTI responds by issuing a GRANT when it can guarantee the time advance will not later result in a message in the federate's past. Specifically, there are two services to request simulation time advances. The grant is implemented via a callback to the federate.

Use of these services always results in the following scenario:

1) Federate requests a time advance and then calls tick,

2) RTI delivers zero or more messages to the federate via the reflectAttributeValues and/or receiveInteraction callbacks in FederateAmbassador,

3) RTI notifies the federate its simulation time has been advanced via a timeAdvanceGrant callback. No additional reflectAttributeValues and/or receiveInteraction callbacks *with time stamps* will be made until the next time an advance in simulation time is requested. (If messages without time stamps are received,

they will be delivered the next time the federate calls tick, regardless of whether an advance in simulation time has been requested.)

Support services provide mapping between string representations of names and integer handles used in the other services, and provide miscellaneous utilities which do not neatly fit into other categories.

There are two other sets of services defined in the HLA. The Ownership Management services allow one federate to transfer ownership of object instance attributes to another federate (at which point the second federate would be responsible for updating those attributes). These are not implemented in the current version of FDK DRTI. The Data Distribution Management services allow you to attach "regions" of interest to publications and subscriptions (to allow minimization of unnecessary network traffic). An initial version of a library for realizing the DDM services, called DDM-Kit, is included. All of the data types used in the RTI interface are declared within a class named RTI which acts as a namespace (requiring the scope qualifier RTI:: before each type name).

7.2 Implementation of DDM Services in FDK

FDK uses RTI 1.3 standard for implementing DDM services. Hence, regions are composed of extents, which in turn are rectangular portions of *k*-dimensional routing space. In that way, a geometric shape can be approximated with a collection of extents. A

publisher region is associated with each publish message generated by a federate. Federates express their interests via subscription regions. If the publisher region associated with a message overlaps with a subscriber region, the message is routed to that subscribing federate. In a distributed simulation application, the DDM services map the name space, description, and interest expressions to the communication services provided by the underlying network. The multicast services are used to realize communications among federates using MCAST libraries. MCAST provides standard group communication services (join, leave, and send messages to groups). Thus, the central problem addressed by DDM-Kit software is mapping description and interest expressions represented as *(region, class attributes)* pair to groups. Interest expressions must be mapped to groups to which the federate must join. Description expressions associated with a message are mapped to one or more groups to which the message must be sent. For any particular description, expression, DDM-Kit in FDK can determine a set of multicast groups. Creating and managing multicast groups is done outside DDM-Kit. However, DDM-Kit specifies how many groups are needed during DDM_init, and uses DDM_modify_groups callback to inform an RTI process when to join or leave groups. Multicast groups are referred to uniquely across all RTI processes by an integer value taken from [0, total groups required – 1] range.

Users of DDM-Kit have to map their attributes to integers by invoking DDM_get_attribute_handle. This integer representation is used to pass attributes back to users when DDM_filter_and_promote is invoked. In addition, special integer value, not assigned to any of the attributes is passed in DDM_init. It is used for DDM_filter_and_promote to designate an attribute as being filtered out.

7.3 DDM Functions in FDK

Each RTI process must call the following procedures when it begins to execute in order to ensure proper initialization of the library. These procedures must be called before RTIKIT_Init() is called.

void DDM_UsingDDM(void)

This procedure sets a flag in RTI-Kit to indicate the DDM-Kit library will be used.

void DDM_init(long p_fed_id, long p_n_feds, long p_n_dimensions,
const char *p_FedFileName, long
p__DDM_filtered_out_attr_index_value,
DDM_modify_groups_proc p_DDM_modify_groups, long
*r_DDM_n_groups, long *r_DDM_size_tag)

Data Types and Support Services

The following types and support services are provided by DDM-Kit:

DDM_expression_handle: a handle that serves as a pointer to a description or interest expression

attribute_handle: a handle for an attribute

class_handle: a handle for a class

The next two services are used to obtain attribute and class handles.

int DDM_get_class_handle(char *p_class_name, class_handle *r_class)

DDM_get_class_handle returns a class handle to be used in subsequent DDM-Kit service calls. p_class_name is a hierarchical name of the class which uniquely identifies it.

r_class is the returned handle. The value returned by this function can be one of the following:

- DDM_success indicates the operation completed successfully.
- DDM_no_class_error indicates no class with this name exists.

int DDM_get_attribute_handle(char *p_attr_name, class_handle

p_class, long p_attr_index, attribute_handle *r_attr)

DDM_get_attribute_handle returns an attribute handle to be used in subsequent DDM-Kit service calls. p_class is the class handle and p_attr_name is an attribute name. p_attr_index is an integer value assigned to attribute by the user. The user must map each attribute to a unique integer value. This integer representation is used to pass attributes back to users when DDM_filter_and_promote is invoked. r_attr is the returned handle. The value returned by this function will be one of the following:

- DDM_success indicates the operation completed successfully.
- DDM_no_class_error indicates no class with this handle exists.
- DDM_no_attribute_error indicates no attribute with this name and class exists.

Finally, each region is represented as follows. A region is a sequence of extents. An extent is a sequence of ranges, one for each dimension. A range is a half closed interval [lower_bound, upper_bound). Dimensions are numbered from 0 to the N-1 in an N-dimensional routing space.

Ranges are specified in order of increasing dimensions to define an extent. An extent can be shared among multiple regions.

7.4 Issues with DDM Implementation in FDK

FDK implements HLA 1.3 specifications. Hence, it uses the system of multiple routing spaces, where regions can span across different routing spaces. Also, the number of extents per region and number of regions per federate is limited to 5 and 10, respectively. The maximum number of federates is limited to 256. Moreover, DDM services are not available as a standard functionality. To overcome these drawbacks, we have developed the P-Pruning DDM algorithm and integrated it with FDK.

7.5 Integration with HLA Architecture in FDK

FDK is an implementation of HLA architecture developed at Georgia Institute of Technology. It has been widely deployed as the platform for HLA-based distributed simulation research. Hence, we have integrated the P-Pruning DDM algorithm with FDK software as a library and compare the performance of P-Pruning DDM algorithm with other DDM algorithms using the FDK simulated environment for some applications with certain characteristics. This unified FDK-DDM architecture will improve current HLA implementations and advance the current state-of-the-art distributed simulation methodologies.

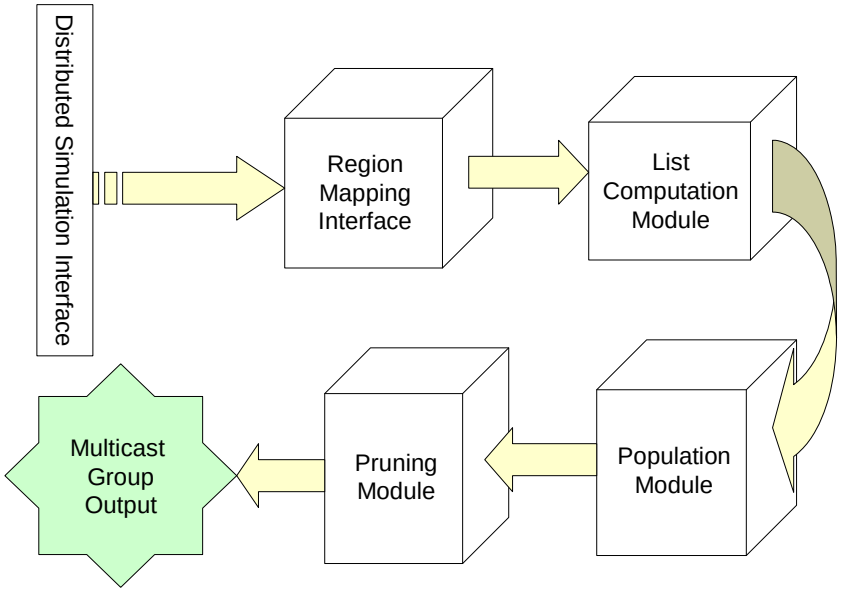


Figure 27. A modular overview of P-Pruning algorithm

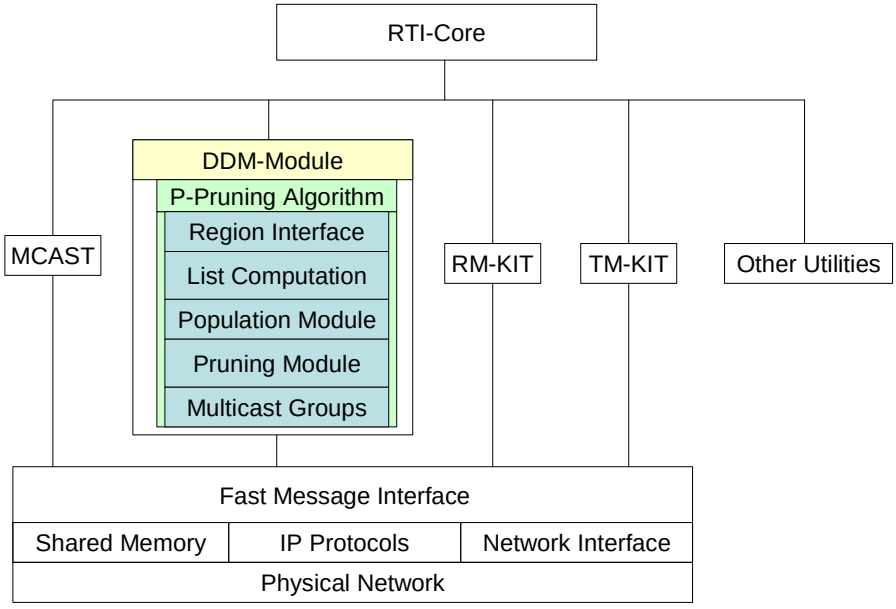


Figure 28. Architectural layout for integration of P-Pruning DDM algorithm with FDK

The integrated architecture of FDK with P-Pruning DDM algorithm is shown in Figure 27. Figure 28 presents the DDM module used within FDK. We now describe the setup procedures involved in the integration process.

Setup Instructions:

To Compile:

nmake /f PDDM_Makefile.win

This creates pddm_fdk.exe, the federate executable, in current folder.

To Run on Single Machine:

Edit MyFed.net to relect the IP of the machine and decide the number of nodes.

Set the COMM_NODE_ID = 0, = 1 and so on.

Run the executable from different DOS windows to simulate the federates.

Setup:

PDDMFed.fed is the new Federation info file. It is based on the MyFed.fed file in minsim simulation. The federation name has been changed to PDDMFed from MyFed in the original simulation.

Structure of MyFed.NET File

```
(Communication
 (Device fm-tcp                  ;; Use FM/MCAST over TCP
   (Mode sync fixed)              ;; the mode supported by fm-tcp
 )
 (Memory Static 16 4096)          ;; use 128 16K buffers
 (Topology
```

```
   (Network 4

     (Host 132.170.109.231  33333)
     (Host 132.170.109.231)
     (Host 132.170.109.231)
     (Host 132.170.109.231)

;;   (Host hostname_of_first_node port)
;;   (Host hostname_of_next_node)
;;   (Host hostname_of_next_node)
;;   (Host hostname_of_next_node)
   )
 )
)
```

Communication Across Network

```
(Communication
 (Device fm-tcp                  ;; Use FM/MCAST over TCP
   (Mode sync fixed)              ;; the mode supported by fm-tcp
 )
 (Memory Static 16 4096)           ;; use 128 16K buffers
 (Topology
  (Network 2

     (Host 132.170.109.231  33333)
     (Host 132.170.109.246)

;;   (Host hostname_of_first_node port)
;;   (Host hostname_of_next_node)
;;   (Host hostname_of_next_node)
```

```
;;    (Host hostname_of_next_node)
    )
  )
)
```

7.6 Design and Development of the Communicator Module

We now describe the integration of P-Pruning algorithm with FDK. First, we present the design of a communicator module and then a minimal simulation in FDK.

The integration of P-Pruning algorithm with FDK consists of following steps:

- Design of P-Pruning to FDK communicator module
- Integration of FDK software and the communicator module

We now describe the design of the communicator module for integrating the P-Pruning algorithm with FDK. The communicator module provides connection between P-Pruning DDM and the FDK federate. It enables conversion of the federate and region representation formats from the IEEE 1516 to the HLA 1.3 as required in FDK. The template of communicator module is useful in any implementation of simulation involving P-Pruning algorithm. Thus, it acts as an API for further simulation experiments.

In the current version of FDK, the number of processors used during the execution does not change and all the processors are available during initialization and entire execution. Each processor s assigned unique number. According to the FDK user manual, FDK has implemented only 25% of the total DDM services from the RTI1.3 Interface specification. Hence, the integration of P-Pruning algorithm with FDK can provide help

with implementation of DDM services in FDK. The information about regions created by federates are stored in two classes: RegionInfo and RegionSets. A RegionInfo class is created upon calling CreateRegion, and contains an array of extents. Since this information also may relate to routing, ClassInfo retains information about multicast groups. During join, the parsing of the fed file results in the creation of multicast groups based on the value of ClassSet.CreateMCast.

IEEE 1516 is the HLA standard approved by IEEE in September 2000 as a successor of the HLA 1.3 specifications. It simplifies the DDM implementation and the basic components in this specification are as follows:

Routing space: There is a single routing space and all dimensions are included in this routing space.

Regions: A region is a single rectangular subspace within the coordinate space. Regions may be defined on any subset of the available dimensions of the coordinate system.

Region set: Regions are grouped into region sets, which consist of one or more regions. The regions in a region set need not all have the same subset of the dimensions of the coordinate system.

Dimension: Dimensions correspond to simulation data and they are used to define regions.

7.6.1 Simulation Results of P-Pruning Algorithm Integrated in FDK

In Figure 29, we have shown the interface setting for a simulation using FDK and its integration with P-Pruning. A simulation that is developed in FDK needs to deliver the information on federates to P-Pruning module. This information includes details about the publisher and subscriber region in each federates in the IEEE 1516 specifications. The P-Pruning algorithm then computes the multicast groups and sends this to the distributed simulation which is used by the federation for DDM.

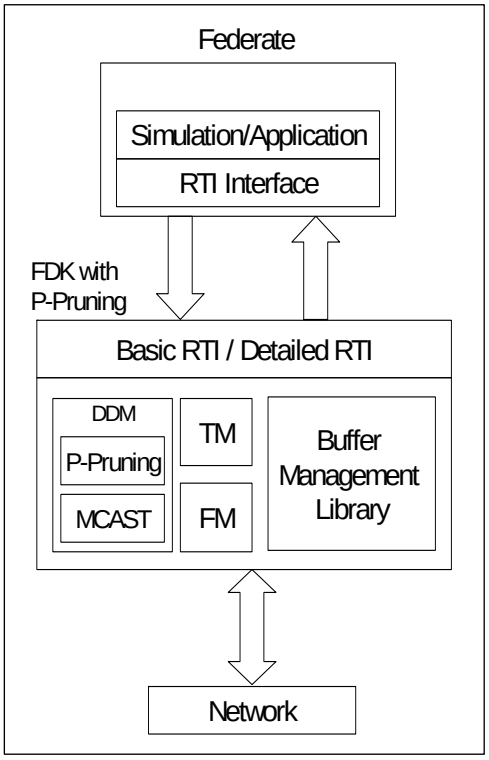


Figure 29. Integrated architecture of FDK with P-Pruning algorithm

The FDK uses following data structures to define regions. As it can be seen, region, referred to as region_handle, is an array of extents *p_extents_value* with *p_n_extents*

number of elements. Extent, in turn, referred to as extent is an array of ranges, one for each dimension, whereas range is a half closed interval [lower_bound, upper_bound).

In the FDK, at any point during an execution there is a set of description and interest expressions, each of which may either be registered (i.e. active) or not. Registered expressions are used to determine data distribution connectivity. Two procedures are used to tag an expression as being active or not. These are registering and unregistering an expression. They can only be invoked after obtaining a handle for a description or interest expression. Besides registering expressions, it is also possible to modify description and interest expressions. For these purpose, two procedures are defined:

```
typedef struct range_Struct range;
typedef struct range_Struct *extent;
struct range_Struct {
long lower_bound; /* lower bound for a range */
long upper_bound; /* upper bound for a range */
};

typedef struct region_handle_Struct region_handle;
struct region_handle_Struct {
extent *p_extents_value; /* sequence of extents */
int p_n_extents; /* number of extents for this region */
};
```

DDM_modify_expression and DDM_modify_region. The first procedure allows changing an expression's value by atomically changing all arguments of an expression's (region, class attribute) pair. The second procedure is used to change a region's argument only.

The DDM_register_expression(DDM_expression_handle p_exp) procedure registers an expression with the handle p_exp. The returned parameter may have one of the following values: DDM_success indicates the operation completed successfully and DDM_no_expression_error indicates no interest or description expression with this handle could be found.

int DDM_unregister_expression(DDM_expression_handle p_exp)

This procedure deactivates an expression with the handle p_exp. The return value is the same as for DDM_register_expression.

int DDM_modify_expression(DDM_expression_handle p_exp,

region_handle p_region, attribute_handle *p_attrs, int p_n_attrs)

Expression p_exp is modified by modifying all parts of an expression value, that is, its region p_region, attributes p_attrs with p_n_attrs.

```
class pRegion
{
public:
   int Xrange1, Xrange2;
   int Yrange1, Yrange2;
   int Region_Type;

   int ID;
   pRegion ();
};
```

The return value is the same as DDM_register_expression.

int DDM_modify_region(DDM_expression_handle p_exp, region_handle p_region).

Modifying the region part of an expression value, that is, its region p_region, modifies expression p_exp. The return value is the same as for DDM_register_expression.

The P-Pruning algorithm can be invoked as library for a distributed simulation application. Based on the federate information provided by the RTI, it generates the multicast groups. In the implementation of the P-Pruning algorithm, we used the following data-structure to represent a region:

In our integration environment, we created a simulation using 4 nodes under Windows XP based on the *minsim* simulation in FDK version 4.2b3. The DDM services were invoked in the simulation to utilize the P-Pruning region-matching routines. Finally, the multicast groups information generated by the P-Pruning was delivered to the simulation for inter-federate communication. Using this experimental setup, we have demonstrated the P-Pruning algorithm can be used a library for implementing DDM services in FDK.

7.7 Distributed FDK Implementation

The performance of DDM algorithms is strongly dependant on the availability of system resources such as memory, CPU time, and communication bandwidth. In the performance-evaluation simulations, we compared the memory usage of four DDM algorithms and found that system memory becomes a bottleneck as the number of federates in increased. Hence, resource-efficient DDM algorithms that can deliver results in constrained conditions are critical in distributed simulation. The constraints become

more relevant as we incorporate real-world large-scale distributed simulations applications in the experiments. Advanced memory management schemes such as hierarchical data-caching and pre-fetching routines can help in solving the memory constraint issues. In the near future, we plan to adapt the P-Pruning DDM algorithm in a resource-constraint environment.

In memory-constraint approach, we view the system memory as a resource that has to be optimally allocated among the processes. The problem is how to deploy efficient data-structures and reorganize the data at run-time so that the DDM computation is not as memory intensive as encountered in practical simulations. I/O efficient data-structures are the key tools in developing a resource-efficient approach. Also, dynamic memory-management strategy that provides efficient garbage collection to reduce unnecessary memory leak at run-time is crucial. The primary motivation in the resource-constraint approach is to devise a scalable, memory-efficient solution for high-performance distributed simulation applications.

7.8 Summary

In this Chapter, we presented the integration of P-Pruning DDM algorithm with FDK. FDK has been widely deployed as the platform for HLA-based distributed simulation research. We believe that integration of P-Pruning DDM algorithm with FDK software can improve the current HLA implementations, and help advance the modeling and simulation technology.

CHAPTER EIGHT: DIRECTIONS FOR FURTHER RESEARCH

8.1 Scalable DDM approach in Distributed Environment

During the performance evaluation experiments, it was observed that the performance of DDM algorithm is adversely affected as the number of federates is increased in the simulation environment. In large-scale distributed simulation application, system scalability can be seriously inhibited by limits on resources such communication bandwidth, memory, and CPU availability [98].

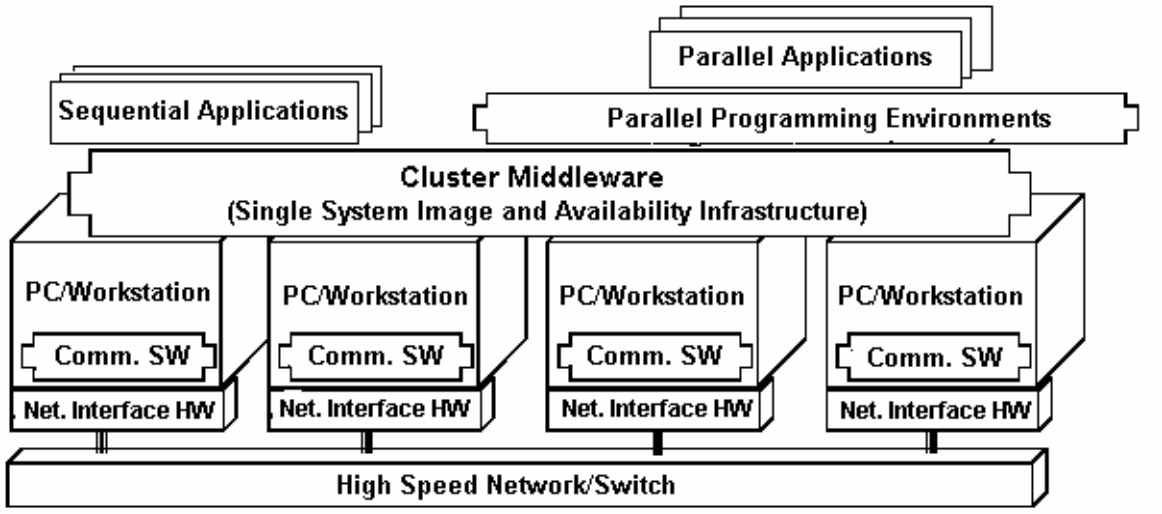


Figure 30. Cluster computer architecture

A distributed DDM algorithm implementation on cluster computers (Figure 30) can scale well as the number of simulation entities increases ([86], [113], [117], [118]). An interesting extension to our work could be a parallel/distributed DDM algorithm that incorporates resources such as communication bandwidth in the resource-constraint

analysis, and address the scalability issues. Figure 31 shows a visual image of the 48-node dual-processor Ariel cluster running on SUN Solaris operating system in the School of Computer Science at University of Central Florida.

Figure 31. 48-node dual-processor Ariel Solaris cluster at UCF

Data distribution management services provided by the RTI should scale in terms of (1) *computational complexity* for handling requests, (2) *message traffic* and/or bandwidth for distributing information, and (3) *memory requirements* for storing attribute information, maintaining tables, etc. The parameters that normally affect scalability are: (a) the *number of federates* (or hosts) in the federation, (b) the *number of simulated entities* per federate, (c) the average complexity concerning the *interests of each entity* (i.e., an entity may have a number of different kinds of sensors), (d) the *interaction rates between objects* after they discover each other, (e) the *locality of objects*, and indirectly (f) the *scenario*.

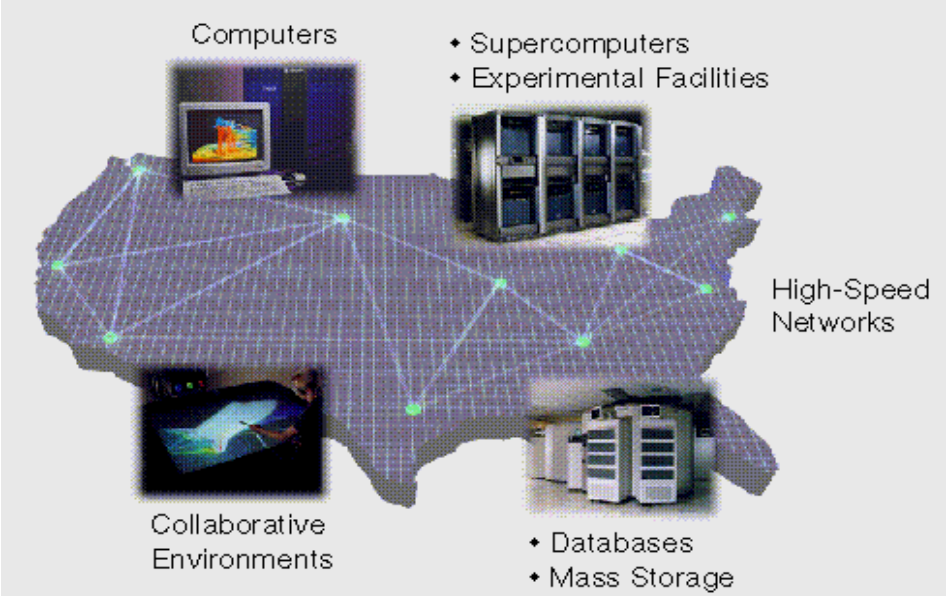


Figure 32. Layout of a distributed computing application sharing data resources

In large-scale distributed simulation application, system scalability can be seriously inhibited by limits on resources such communication bandwidth, memory, and CPU availability. While this is not totally unexpected; for a DDM algorithm to be effective and deployable in high performance modeling and simulation applications, it must be scalable. Our work can be extended to a distributed DDM algorithm implementation on cluster computers, incorporate resources such as communication bandwidth in the resource-constraint analysis, and investigate the scalability issues. Figure 32 shows the layout of a distributed computing application as often encountered in DDM scenarios sharing resources across geographically distributed locations. The distributed implementation of P-Pruning algorithm will provide a scalable and resource-efficient DDM approach. The P-Pruning DDM algorithm on integration with FDK

software will improve the current HLA implementations, and advance the modeling and simulation technology.

8.2 Using Real-World DDM Applications for Test

We can apply the DDM techniques developed in this research using data derived from real-world applications. Now, we present three scenarios for illustrating the use of DDM in real-world applications ([104], [106], [107], [108], [111], [112]). The following applications are considered:

i) Ground-Based Radar (GBR) tracking tanks with limited operating range.

ii) JSTARS (airborne radar) flying and tracking tanks with limited operating range.

iii) AWACS flying and tracking airborne aircrafts.

Application (i) represents static condition, i.e., neither the publisher or subscriber regions, once set, are modified. As illustrated in Figure 33, ground-based radars GBR1 through GBR2 cover an area and their subscriber regions are shown. Tanks T1 through T6 have limited operating range reflected by their respective publisher regions. The subscriber region for radar GBR1 is shown overlaps publisher regions of tanks T1 and T2.

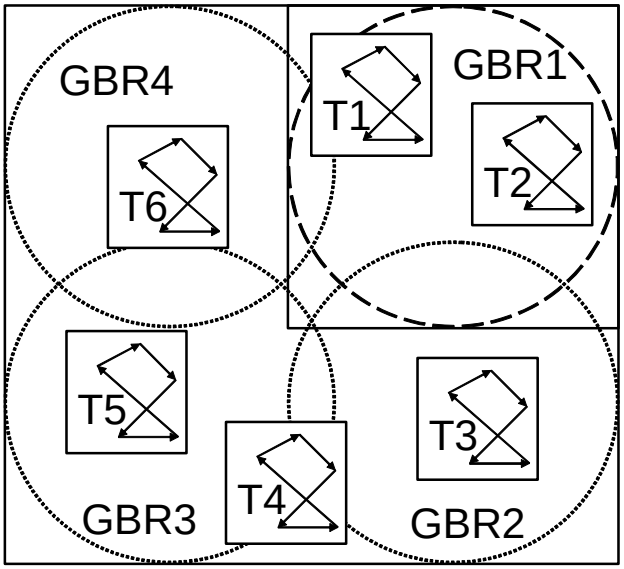


Figure 33. Application having ground-based radars tracking tank with limited range

Application (ii) represents semi-dynamic condition, which have either the publisher or subscriber regions fixed. In this case, the publication region of JSTARS is not fixed. As illustrated in Figure 34, a dynamic JSTARS (Joint Surveillance Target Attack Radar System) is circling over the entire area and its subscriber region is shown. Tanks T1 through T6 have limited operating range reflected by their respective publisher regions. At this particular instance, the subscriber region for JSTARS overlaps publisher regions of tanks T1, T2, and T3.

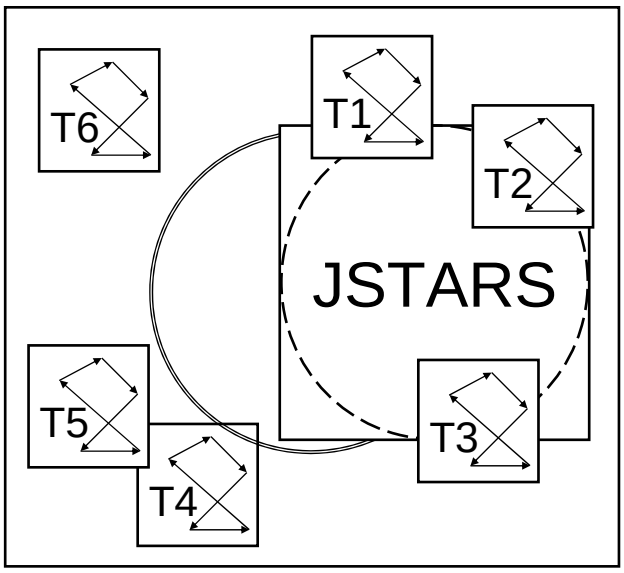


Figure 34. Application having JSTARS flying and tracking tanks with limited range

Application (iii) represents the dynamic DDM test-case, where both subscriber and publisher regions are dynamically modified. In this case, the publication region of JSTARS is fixed. As illustrated in Figure 35, a dynamic AWACS (Airborne Warning and Control System) is circling over the entire area and its subscriber region is shown. Airborne aircrafts A1 through A6 are circle over the area and move in and out of AWACS range. At this particular instance, the subscriber region for AWACS overlaps publisher regions of aircrafts A1, A2, and A3. Table 2 shows the classification of three applications based on the nature of subscriber region.

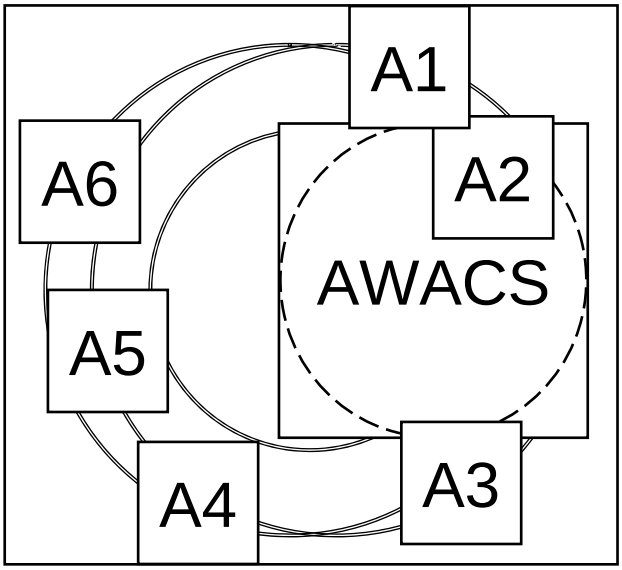


Figure 35. Application having AWACS flying and tracking airborne aircrafts

Table 2. Classification of real-world scenarios based on subscriber region update

Sensor / Target	Sensor Subscription	Target Update
GBR/Tank	Static	Static
JSTAR/Tank	Dynamic	Static
AWACS/Aircraft	Dynamic	Dynamic

CHAPTER NINE: CONCLUSIONS

DDM is necessary for large-scale distributed simulation applications as information exchange and delivery become more complex. In this thesis, we presented the design, analysis, and performance evaluation of an efficient algorithm for DDM. The P-Pruning DDM algorithm shows better performance in terms of computation time, memory usage at run-time, and size of multicast groups as compared to other algorithms in a simulated environment. We have also presented the design and performance evaluation of a resource-efficient enhancement to the P-Pruning algorithm for DDM. By deploying efficient data structures, the resource-constraint P-Pruning DDM algorithm scales well in high performance distributed-simulation environment. It shows better performance in terms of computation time and memory usage at run-time in simulation environment. We have also extended our DDM research work by incorporating resource constraints and develop resource-efficient approaches in constrained environments. A distributed implementation of DDM on cluster computers will provide scalable solution to this problem. The P-Pruning DDM algorithm when integrated with FDK software will improve the current HLA implementations, and advance the modeling and simulation technology.

APPENDIX A: LIST OF RTI SOFTWARE

Table 3. List of commercial RTI software and their details

RTI Name	Vendor	HLA Spec.	DDM Method	Language	Platform	Compiler	Latest Release Date	Notes
pRTI	Pitch AB	IEEE 1516, 1.3	Unknown	Java, C++	Java2, Win32	JDK 1.4, MSVC++	Aug. 05	www.pitch.se
ERTI	Mitsubishi Space Software (MSS)	1.3	Unknown	C++	SunOS 8, Windows, Linux	MSVC++, gcc, Sun Visual C++	Apr. 02	*
MAK RTI	MAK Technologies	1.3	Unknown	C++	Win32, Linux, IRIX	MSVC++, gcc, MIPS Pro C++	Jan. 03	www.mak.com
RTI-NG Pro	VTC & SAIC	1.3	Unknown	C++	Solaris, Linux	C++ Forte 6, gcc	May 05	www.virtc.com
RTI NG	SAIC	1.3	Unknown	C++	Solaris 2.6	Sun C++ 4.2	May 04	*
Matrex RTI NG	VTC	1.3	Unknown	C++	Linux	gcc	May 05	*
HPC-RTI	RAM Labs	1.3	Grid	*	*	*	*	www.ramlabs.com

* denotes that data is not available.

Table 4. List of academic RTI software

NAME	Institute	Version	DDM Method	Platform	Website
DRTI	Georgia Institute of Technology	1.3	*	C++ on Win32 and Linux	http://www.cc.gatech.edu/computing/pads
Light-Weight RTI	George Mason University	1.0	*	*	http://netlab.gmu.edu/rti
RTI 1.3	MIT Lincoln Labs	1.0	Grid	*	http://dss.ll.mit.edu

* denotes that data is not available.

APPENDIX B: RESULTS OF SIMULATION EXPERIMENTS

Table 5. Computation time results (in Seconds) for routing space 50 x 50 and grid size 2 x 2

Routing Space Dimensions	No. of Federates	Region Matching	Fixed Grid DDM	Dynamic Grid DDM	P-Pruning DDM
50	10	0.813	4.828	7.078	0.265
50	20	0.766	6.672	11.547	0.218
50	30	2.032	10.593	20.172	0.219
50	40	2.953	15.282	21.654	0.219
50	50	5.36	18.516	23.125	0.218

Table 6. Computation time results (in Seconds) for routing space 50 x 50 and grid size 5 x 5

Routing Space Dimensions	No. of Federates	Region Matching	Fixed Grid DDM	Dynamic Grid DDM	P-Pruning DDM
50	10	0.547	0.421	0.75	0.219
50	20	1	0.516	1.359	0.25
50	30	1.094	0.766	1.953	0.218
50	40	2.625	0.969	1.172	0.219
50	50	3.297	0.984	1.75	0.218

Table 7. Computation time results (in Seconds) for routing space 100 x 100 and grid size 5 x 5

Routing Space Dimensions	No. of Federates	Region Matching	Fixed Grid DDM	Dynamic Grid DDM	P-Pruning DDM
100	20	0.734	8.078	8.781	0.688
100	40	4	11.203	15.922	0.922

Table 8. Memory usage (in MB) results for routing space 50 x 50 and grid size 2 x 2

Routing Space Dimensions	No. of Federates	Region Matching	Fixed Grid DDM	Dynamic Grid DDM	P-Pruning DDM
50	10	215.641	334.125	334.289	213.961
50	20	209.555	443.297	443.289	201.719
50	30	219.68	565.797	565.629	201.852
50	40	232.945	719.879	720.105	232.848
50	50	250.141	861.102	861.113	250.09

Table 9. Memory usage (in MB) results for routing space 50 x 50 and grid size 5 x 5

Routing Space Dimensions	No. of Federates	Region Matching	Fixed Grid DDM	Dynamic Grid DDM	P-Pruning DDM
50	10	186.867	274.773	274.773	177.168
50	20	197.117	274.969	274.969	177.363
50	30	207.281	275.164	275.164	177.559
50	40	217.449	275.359	275.359	177.754
50	50	227.621	275.563	275.563	177.957

Table 10. Memory usage (in MB) results for routing space 100 x 100 and grid size 5 x 5

Routing Space Dimensions	No. of Federates	Region Matching	Fixed Grid DDM	Dynamic Grid DDM	P-Pruning DDM
100	20	232.668	813.805	813.324	200.711
100	40	328.059	1430.27	1429.92	201.512

Table 11. Multicast group size results for routing space 50 x 50 and grid size 2 x 2

Routing Space Dimensions	No. of Federates	Region Matching	Fixed Grid DDM	Dynamic Grid DDM	P-Pruning DDM
50	10	10	625	235	8
50	20	20	625	408	20
50	30	30	625	476	30
50	40	40	625	476	40
50	50	50	625	478	50

Table 12. Multicast group size results for routing space 50 x 50 and grid size 5 x 5

Routing Space Dimensions	No. of Federates	Region Matching	Fixed Grid DDM	Dynamic Grid DDM	P-Pruning DDM
50	10	10	100	34	9
50	20	20	100	79	18
50	30	30	100	89	28
50	40	40	100	85	40
50	50	50	100	77	50

Table 13. Multicast group size results for routing space 100 x 100 and grid size 5 x 5

Routing Space Dimensions	No. of Federates	Region Matching	Fixed Grid DDM	Dynamic Grid DDM	P-Pruning DDM
100	20	20	400	263	19
100	40	40	400	290	40

Table 14. Memory-constraint P-Pruning algorithm scalability result for routing space 20000 x 20000

Routing Space Dimensions	No. of Federates	P-Pruning DDM	Memory Constraint DDM
100	100	215.035	214.293
500	500	208.227	189.242
1000	1000	268.352	191.926
3000	3000	909.316	222.535
4000	4000	1469.2	248.504
5000	5000		283.059
7500	7500		402.465
10000	10000		570.445
15000	15000		1049.01
17000	17000		1294.45
18000	18000		1429.02
19000	19000		1570.9
20000	20000		1752

Table 15. Comparison of computation time between Memory-Constraint and normal P-Pruning routing space 4000 x 4000

Routing Space Dimensions	No. of Federates	P-Pruning DDM	Memory Constraint DDM
100	100	0.281	0.266
500	500	0.328	0.906
1000	1000	0.5	1.265
3000	3000	80.672	6.672
4000	4000	204.782	12.656

Table 16. Comparison of memory usage at run-time by Memory-Constraint and normal P-Pruning routing space 4000 x 4000

Routing Space Dimensions	No. of Federates	P-Pruning DDM	Memory Constraint DDM
100	100	215.035	214.293
500	500	208.227	189.242
1000	1000	268.352	191.926
3000	3000	909.316	222.535
4000	4000	1469.2	248.504

APPENDIX C: FDK P-PRUNING INTEGRATION OUTPUT

RESULTS on Two Nodes

Node 0:

C:\FDK\New_minsim>node0.bat

C:\FDK\New_minsim>set COMM_NODE_ID=0

C:\FDK\New_minsim>minsim
Debug set to DEBUG_DRTI
Initializing Instance tables
Registering Handlers
RTI Ambassador Initialized
double: 8
rti13::Double: 8
TM_Time: 8
float: 4
Creating federation execution... created. (sort of)
Current simulation time: 0
Joining federation execution...
myID initialized
Begin Join
Pass 1 complete
Pass 2 complete
Transports: 0
Spaces: 1
Dimensions: 1
Classes: 2
Attributes: 1
Interactions: 2
Parameters: 1

Allocating a heap of max 16 elements.

Info: commkit: loaded fm-tcp: sockets-based TCP using the FM library

Info: commkit: execution is synchronous (single-threaded)

Info: commkit: net topolgy is fixed

Number of FM Nodes: 2

0: Initializing RTI-KIT(Version: 4.0)...Node 0 initialized logging services.

Join: Creating Object Class Multicast Groups

Join: Creating Interaction Class Multicast Groups

Join: Entering Barrier

Join: Leaving Barrier

Join: Entering Final Barrier

Out of the call now joined.

Publishing and subscribing...Creating HVPS..done.

Enabling Time Constraint...enabled.

Entering initial barrier...done.

Publishing an object class and its attributes...getObjectClassHandle for Name: o
bjectRoot.MyObjectClass

done

Subscribing to that object class...done

Tick now...done

Create a AHVPS...done

Sending update message for attribute values of MyObjectClass...done

Sending delete message for MyObjectClass object...Received update attributes message.

done

Sending a message for time 3.500000 ...done.

Requesting time... 1.000000 ...done.

Granted to time: 1.000000

Requesting time... 2.000000 ...done.
Granted to time: 2.000000
Requesting time... 3.000000 ...done.
Granted to time: 3.000000
Requesting time... 4.000000 ...done.
Granted to time: 4.000000
Requesting time... 5.000000 ...done.
Granted to time: 5.000000
Requesting time... 6.000000 ...done.
Granted to time: 6.000000
Entering final barrier...done
Requesting time... 7.000000 ...done.
Exited with tick_count 1000
Resigning from Federation...service_this_socket: myid 0 readn() error; proc 1 so
ck 1860 bytes_read = -1

Node 1

C:\FDK\New_minsim>node1.bat

C:\FDK\New_minsim>set COMM_NODE_ID=1

C:\FDK\New_minsim>minsim
Debug set to DEBUG_DRTI
Initializing Instance tables
Registering Handlers
RTI Ambassador Initialized
double: 8
rti13::Double: 8
TM_Time: 8
float: 4

Creating federation execution... created. (sort of)
Current simulation time: 0
Joining federation execution...
myID initialized
Begin Join
Pass 1 complete
Pass 2 complete
 Transports: 0
 Spaces: 1
 Dimensions: 1
 Classes: 2
 Attributes: 1
 Interactions: 2
 Parameters: 1

Allocating a heap of max 16 elements.
Info: commkit: loaded fm-tcp: sockets-based TCP using the FM library
Info: commkit: execution is synchronous (single-threaded)
Info: commkit: net topolgy is fixed
Number of FM Nodes: 2
1: Initializing RTI-KIT(Version: 4.0)...Node 1 initialized logging services.
Join: Entering Barrier
Join: Leaving Barrier
Join: Joining Groups
Join: Entering Final Barrier

Out of the call now joined.
Publishing and subscribing...Creating HVPS..done.
Enabling Time Constraint...enabled.
Entering initial barrier...done.

Publishing an object class and its attributes...getObjectClassHandle for Name: o
bjectRoot.MyObjectClass

done

Subscribing to that object class...done

Tick now...done

Create a AHVPS...done

Sending update message for attribute values of MyObjectClass...done

Sending delete message for MyObjectClass object...Received update attributes message.

done

Requesting time... 1.000000 ...done.

Granted to time: 1.000000

Requesting time... 2.000000 ...done.

Granted to time: 2.000000

Requesting time... 3.000000 ...done.

Granted to time: 3.000000

Requesting time... 4.000000 ...done.

Granted to time: 4.000000

Requesting time... 5.000000 ...done.

Granted to time: 5.000000

Requesting time... 6.000000 ...done.

Granted to time: 6.000000

Entering final barrier...done

Setting not constrained

Thrown Exception: TimeRegulationWasNotEnabled because: Time Regulation was not enabled.

Destroying Federation...done.

APPENDIX D: LIST OF PUBLICATIONS

Parallel & Distributed Simulation:

1. **Pankaj Gupta** and Ratan K. Guha, "Integration of the P-Pruning Data Distribution Management Technique with FDK", 2007 (In preparation).

2. **Pankaj Gupta** and Ratan K. Guha, "Design, Analysis, and Performance Evaluation of the P-Pruning DDM Algorithm", Modeling and Simulation in Engineering Journal, 2007 (In Preparation).

3. **Pankaj Gupta** and Ratan K. Guha, "Design, Analysis, and Performance Evaluation of an Efficient Algorithm for Data Distribution Management in High Level Architecture", Journal of Defense Modeling and Simulation, 2007.

4. **Pankaj Gupta** and Ratan K. Guha, "Integration of the P-Pruning Data Distribution Management Technique with FDK", 2006 Summer Computer Simulation Conference, Calgary, Canada, July 31-August 2, 2006.

5. **Pankaj Gupta** and Ratan K. Guha, "Data Distribution Management for High Performance Distributed Simulation in Resource-Constraint Environment", 2006 High Performance Computing & Simulation (HPC & S), 20th European Conference on Modelling And Simulation, Bonn, Germany, May 28-31, 2006.

6. **Pankaj Gupta** and Ratan K. Guha, "Design and Implementation of an Efficient Algorithm for Data Distribution Management in High Level Architecture", 4th Symposium on Design, Analysis, and Simulation of Distributed Systems, 2006 Spring Simulation Multiconference, Huntsville, Alabama, April 2-6, 2006.

7. **Pankaj Gupta** and Ratan K. Guha, "A Heuristic for Efficient Data Distribution Management in Distributed Simulation", Enabling Technologies for Simulation Science IX, (eds. Dawn A. Trevisani, Alex F. Sisti), Proc. of the SPIE, vol. 5805, Defense & Security Symposium, March 2005, Orlando, FL, pp. 362-370.

Modeling & Simulation:

8. **Pankaj Gupta** and G. Prasad, "A Scalable, Portable, Object-Oriented Framework for Parallel Multi-Sensor Data-Fusion Applications in HPC Systems", *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications VIII, SPIE Defense and Security Symposium*, 13-15 April 2004, Orlando, FL.

9. **Pankaj Gupta** and G. Prasad, **"**SIMPAR: a portable object-oriented simulation-science-based metamodel framework for performance modeling, prediction, and evaluation of HPC systems", *Enabling Technologies for Simulation Science VIII Conference, SPIE Defense and Security Symposium,* 13-15 April 2004, Orlando, FL.

10. G. Prasad, S. Jayaram, J. Ward, and **Pankaj Gupta**, "SimBox: a simulation-based scalable architecture for distributed command and control of spaceport and service constellations", *Enterprise Modeling Applications and Techniques Session, Enabling*

Technologies for Simulation Science VIII Conference, SPIE Defense and Security Symposium, 13-15 April 2004, Orlando, FL.

11. G. Prasad, S. Jayaram, J. Ward, and **Pankaj Gupta**, "SimBox: a scalable architecture for aggregate distributed command and control of spaceport and service constellations", *Modeling, Simulation and Calibration Of Space-Based Systems, SPIE Defense and Security Symposium*, 13-15 April 2004, Orlando, FL.

Graph Theory:

12. N. Deo and **Pankaj Gupta**, "Graph-Theoretic Structure of the World Wide Web", A chapter in *Information Technology: Principles and Applications* (eds. A. K. Ray and T. Acharya), Prentice-Hall, India, ISBN 81-203-2184-7, 2004, pp. 169-203.
13. **Pankaj Gupta** and N. Deo, "Modeling Discrete Optimization Problems with Graphs", *Proc. IASTED International Conference on Modelling, Simulation, and Optimization*, July 2-4, 2003, Banff, Canada.
14. **Pankaj Gupta** and N. Deo, "Diameter of a Random Graph and its Implications on the Web graph", Congressus Numerantium, 160 (2003), pp. 109-116, *34th Southeastern International Conference on Combinatorics, Graph Theory and Computing*, Mar. 3-7, 2003, Boca Raton, FL.
15. N. Deo and **Pankaj Gupta**, "Graph-Theoretic Analysis of the World Wide Web: New Directions and Challenges", Matemática Contemporânea, vol. 25, pp. 49-70, 2003.
16. **Pankaj Gupta** and N Deo, "Analysis of Graph Theoretic Models for the Web", *Proc. 33rd Southeastern International Conference on Combinatorics, Graph Theory and Computing*, Mar. 4-8, 2002, Boca Raton, FL. Congressus Numerantium, vol. 155-159 (2002).
17. N. Deo and **Pankaj Gupta**, "Graph-Theoretic Web Algorithms: An Overview", *Lecture Notes in Computer Science*, (eds. T. Böhme and H. Unger), vol. 2026, pp. 91-102, 2001, Germany.
18. N. Deo and **Pankaj Gupta**, "Sampling the Web With Random Walks", Congressus Numerantium, 149 (2001), pp. 143-154, 32nd *Southeastern International Conference on Combinatorics, Graph Theory and Computing*, Feb. 26 - Mar. 2, 2001, Baton Rouge, LA.
19. A. Abdalla, N. Deo, and **Pankaj Gupta**, "Random-Tree Diameter and the Diameter-Constrained MST", *Congressus Numerantium*, 144 (2000), pp. 161-182, *31st Southeastern International Conference on Combinatorics, Graph Theory and Computing*, March 13-17, 2000, Boca Raton, FL.
20. N. Deo, Pankaj Gupta, and B. Litow, "Modeling the Web: Linking Discrete and Continuous", *Proc. 2000 Summer Computer Simulation Conference*, July 16-20, 2000, Vancouver, Canada, pp. 395-400.

LIST OF REFERENCES

[1] G. Riley, M. Ammar, R. Fujimoto, A. Park, K. Perumalla, and D. Xu. "A Federated Approach to Distributed Network Simulation", *ACM Transactions on Modeling and Computer Simulation,* Vol. 14, No. 2, pp. 116-148, April 2004.

[2] R. Fujimoto, T. McLean, K. Perumalla, and I. Tacic. "Design of high-performance RTI software", In Proceedings of Distributed Simulations and Real- time Applications (DS-RT), San Francisco, CA, 2000.

[3] FDK—Federated simulations development kit. http://www.cc.gatech.edu/computing/pads/fdk.html

[4] D. J. Van Hook, S. J. Rak, and J. O. Calvin. "Approaches to RTI Implementation of HLA Data Distribution Management Services", 96-14-084, Fifteenth Workshop on Standards for the Interoperability of Distributed Simulations, September 16-20, 1996.

[5] D. J. Van Hook and J. O. Calvin. "Data Distribution Management in RTI 1.3.", In Proceedings of the 1998 Spring Simulation Interoperability Workshop. No. 98S-SIW-206, Orlando, FL, 1998.

[6] GMU RTI homepage. http://netlab.gmu.edu/rti/

[7] http://www.virtc.com/products.jsp

[8] 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 10 - 15, 2002, paper no. 02S-SIW-044.

[9] Defense Modeling and Simulation Office. 1997. HLA Compliance Checklist, Federate, Version 1.1.Available from http://hla.dmso.mil/hla/policy/cmp_cl11.html/.

[10] Defense Modeling and Simulation Office. 1997. Test Procedures For High Level Architecture Interface Specification, Version 1.1. Available from http://hla.dmso.mil/hla/policy/pro_v11.doc

[11] Defense Modeling and Simulation Office. 1997. Test Procedures For High Level Architecture Object Model Template, Version 1.1. Available from http://hla.dmso.mil/hla/policy/pro_v3-0.doc

[12] Join Interoperability Test Command Document. HLA Services Overview. http://jitc.fhu.disa.mil/jdep/briefings/hla_dis/hla_services_overview.pdf. Feb. 2003.

[13] Defense Modeling and Simulation Office. 1998. High Level Architecture Interface Specification, v1.3. http://hla.dmso.mil/hla/tech/ifspec/if1-3d9b.doc.

[14] Defense Modeling and Simulation Office. 1998. High Level Architecture Object Model Template, v1.3. http://hla.dmso.mil/hla/tech/omtspec/omt1-3d4.doc.

[15] Defense Modeling and Simulation Office. 1998. High Level Architecture Rules, v1.3" http://hla.dmso.mil/hla/tech/rules/rules1-3d2b.doc.

[16] Defense Modeling and Simulation Office. 1998. Federation Execution Planner's Workbook. http://hla.dmso.mil/hla/federation/fedep/fepw.xls.

[17] MAK High Performance RTI. http://www.mak.com/rti.php

[18] D. D. Wood. Implementation of DDM in the MAK High Performance RTI. White paper. http://www.mak.com/ddm_whitepaper.htm.

[19] D. D. Wood. Implementation of DDM in the MAK High Performance RTI. In Proceedings of the 2002 Spring Simulation Interoperability Workshop, March 10 - 15, 2002, paper no. 02S-SIW-056.

[20] R. C. Burns, R. M. Rees, and D. D. Long. "Efficient Data Distribution in a Web Server Farm", *IEEE Internet Computing*, Vol. 5, No. 4, pp. 56-65, 2001.

[21] L. Chen and H. Choi. "Approximation Algorithms for Data Distribution with Load Balancing of Web Servers", In Proceedings of the 2001 IEEE International Conference on Cluster Computing, pp. 274-281, Los Angeles, CA, 2001.

[22] J. Mark 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", In Proceedings of the International Conference on Computational Science, Krakow, Poland, 2004.

[23] L. Arguello, P. Chliaev, W. Fehse, J. Miró, A. Vankov. "HLA-based distributed simulation for International space station operations", In Proceedings of the International Conference on Space Operations (SpaceOps), June 1-5, 1998, Tokyo, Japan paper ID: 4c007, http://track.sfo.jaxa.jp/spaceops98/paper98/track4/4c007.pdf.

[24] P. Lee. "Efficient algorithms for data distribution on distributed memory parallel computers", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 8, pp. 825-839, 1997.

[25] J. Barbosa, C.N. Morais, and A.J. Padilha. “Simulation of data distribution strategies for LU factorization on heterogeneous machines”, In Proceedings of the International Parallel and Distributed Processing Symposium, 22-26 April 2003.

[26] H. Sivaraman and C.S. Raghavendra. “ADDT: automatic data distribution tool for porting programs to PVM”, In Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences, Vol. 1, pp. 557 – 564, 3-6 Jan. 1996.

[27] G. Yang, J. Ruoming, and G. Agrawal. “Impact of data distribution, level of parallelism, and communication frequency on parallel data cube construction”, In Proceedings of the International Parallel and Distributed Processing Symposium, 22-26 April 2003.

[28] M. Amri. “A new data distribution method for parallel ray tracing”, In Proceedings Seventh International Conference on Information Visualization, pp. 79–84, 16-18 July 2003.

[29] C. Lin, Y. Chung, and J. Liu. “Performance evaluation of data distributions with load-balancing for sparse arrays”, In Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks, pp. 207–212, May 10-12, 2004.

[30] N. Comino and V.L. Narasimhan. “A novel data distribution technique for host-client type parallel applications”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 2, pp. 97–110, Feb. 2002.

[31] IEEE New Standards Committee (NESCOM). 1997. IEEE Project Authorization Request (PAR) 1516. Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules.

[32] IEEE New Standards Committee (NESCOM). 1997. IEEE Project Authorization Request (PAR) 1516.1 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification.

[33] IEEE New Standards Committee (NESCOM). 1997. IEEE Project Authorization Request (PAR) 1516.2 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification.

[34] M. D. Petty. “Comparing high level architecture data distribution management specifications 1.3 and 1516”, *Simulation Practice and Theory*, Vol. 9, No. 3-5, pp. 95-119, 2002.

[35] K. L. Morse and M. D. Petty. “Data distribution management migration from DoD 1.3 to IEEE 1516”, In Proceedings of the Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications, pp. 58–65, 13-15 Aug., 13-15 August 2001, Cincinnati, OH.

[36] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly. “The DoD High Level Architecture: an Update”, In Proceedings of the 1998 Winter Simulation Conference, Washington DC, 1998.

[37] F. Kuhl, R. Weatherly, and J. Dahmann. *Creating computer simulation systems: an introduction to the High Level Architecture*, Upper Saddle River, NJ, Prentice Hall, 2000.

[38] J. S. Dahmann. “The High Level Architecture and beyond: technology challenges”, In Proceedings of the thirteenth workshop on Parallel and distributed simulation. Atlanta, Georgia, 64 – 70, 1999.

[39] J. S. Dahmann, J. O. Calvin, and R. M Weatherly. “A reusable architecture for simulations”, *Communications of the ACM*, Sep 1999. Vol. 42, No. 9. pp. 79-84.

[40] R. K. Guha and M. Bassiouni. “Simulation Methods and Applications”, *Simulation Practice and Theory*, Vol. 9, No. 3-5, pp. 91-93, 2002.

[41] H. A. Jacobsen. “Tutorial: OMG data distribution service”, In Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops, pp. 198–198, 2003.

[42] G. Pardo-Castellote. “OMG data distribution service: architectural overview”, *IEEE Military Communications Conference*, MILCOM, Vol. 1, pp. 242–247, 2003.

[43] A. Boukerche and A. Roy. “In search of data distribution management in large scale distributed simulations”, In Proceedings of the 2000 Summer Computer Simulation Conference, Vancouver, Canada, 2000.

[44] G. Tan, R. Ayani, Y. S. Zhang, and F. Moradi. “Grid-based data management in distributed simulation”, In Proceedings of the 33rd Annual Simulation Symposium, pp. 7-13, 16-20 April, 2000.

[45] A. Boukerche, A. Roy, and N. Thomas. “Dynamic Grid-Based Multicast Group Assignment in Data Distribution Management”, In Proceedings of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT’ 00), pp. 47-54, 2000.

[46] A. Boukerche and A. Roy. “Dynamic Grid-Based Approach to Data Distribution Management”, *Journal of Parallel and Distributed Computing*, Vol. 62, No. 3, pp. 366-392, March 2002.

[47] A. J. Roy. “Dynamic grid-based data distribution management in large scale distributed simulations”, M.S. Thesis, Dept. of Computer Science, University of North Texas, 2000.

[48] A. Boukerche, N. J. McGraw, C. Dzermajko, and K. Lu. “Grid-Filtered Region-Based Data Distribution Management in Large-Scale Distributed Simulation Systems”, In Proceedings of the 38th annual Symposium on Simulation, pp. 259 – 266, 2005.

[49] G. Tan, X. Liang, F. Moradi, and S. Taylor. “An agent-based DDM for High Level Architecture”, In Proceedings of the 15th Workshop on Parallel and Distributed Simulation, pp. 75-82, 15-18 May, 2001.

[50] G. Tan, Y. Zhang, and R. Ayani. “A hybrid approach to data distribution management”, In Proceedings of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications, pp. 55-61, 2001.

[51] Y. Jun, C. Raczy, and G. Tan. “Evaluation of a sort-based matching algorithm for DDM”, In Proceedings of the sixteenth workshop on Parallel and distributed simulation. pp. 68–75, 2002.

[52] C. Raczy, G. Tran, and J. Yu. “A sort-based DDM matching algorithm for HLA”, *ACM Transactions on Modeling and Computer Simulation* (TOMACS), Vol. 15, No. 1, pp. 14-38, 2005.

[53] B. I. Kumova. “Dynamically Adaptive Partition-Based Data Distribution Management”, In Proceedings of the 19th Workshop on Parallel and Distributed Simulation (PADS 2005), pp. 292– 300, June 1-3, 2005, Monterey, CA, USA.

[54] 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, pp. 82–85, Monterey, CA, USA, 2005.

[55] R. Minson and G. Theodoropoulos. “An Adaptive Interest Management Scheme for Distributed Virtual Environments”, In Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, pp. 273 – 281, 2005.

[56] A. Boukerche and C. Dzermajko. “Performance evaluation of Data Distribution Management strategies”, *Concurrency and Computation: Practice and Experience*, Vol. 16, No. 15, pp. 1545 – 1573, 2004.

[57] A. Boukerche and C. Dzermajko. “Scalability and performance evaluation of an aggregation/disaggregation scheme for data distribution management in large-scale distributed interactive systems”, In Proceedings of the 37th Annual Simulation Symposium, pp. 238–245, 18-22 April, 2004.

[58] A. Boukerche and C. Dzermajko. “Dynamic grid-based vs. region-based data distribution management strategies for large-scale distributed systems”, In Proceedings of the International Parallel and Distributed Processing Symposium, pp. 7, 22-26 April, 2003.

[59] P. Gupta and R. K. Guha. “Design, Analysis, and Performance Evaluation of an Efficient Algorithm for Data Distribution Management in High Level Architecture.” Computer Science Technical Report CS-TR-05-12, School of Computer Science, University of Central Florida, Orlando, FL, December 2005.

[60] K. L. Morse. “An Adaptive, Distributed Algorithm for Interest Management”, Ph.D. Dissertation, Information & Computer Science Department, University of California, Irvine, 2000.

[61] M. D. Petty. “Computational geometry techniques for terrain reasoning and data distribution problems in distributed battlefield simulation”, Ph.D. Dissertation, Computer Science Department, University of Central Florida, Orlando, 1997.

[62] K. L. Morse and M. Zyda. “Multicast grouping for data distribution management”, *Simulation Practice and Theory*, Vol. 9, No. 3-5, pp. 121-141, 2002.

[63] C. Dzermajko. “Performance comparison of data distribution management strategies in large scale distributed simulation”, M.S. Thesis, Computer Science Department, University of North Texas, 2004.

[64] M. D. Petty and K. L. Morse. “The computational complexity of the high level architecture data distribution management matching and connecting processes”, *Simulation Modelling Practice and Theory*, Vol. 12, pp. 217–237, 2004.

[65] M. D. Petty and K. L. Morse. “Computational Complexity of HLA Data Distribution Management”, In Proceedings of the 2000 Fall Simulation Interoperability Workshop, September 17-22, 2000, paper no. 00F-SIW-143.

[66] M. D. Petty. “Data Distribution Management Specifications 1.3 and 1516 Are Equivalently Powerful”, In Proceedings of the 2001 Spring Simulation Interoperability Workshop, March 25-30, 2001, paper no. 01S-SIW-127.

[67] M. D. Petty. “Geometric and Algorithmic Results Regarding HLA Data Distribution Management Matching”, In Proceedings of the 2000 Fall Simulation Interoperability Workshop, September 17-22, 2000, paper no. 00F-SIW-072.

[68] P. Gupta and R. K. Guha. “A heuristic for efficient data distribution management in distributed simulation”, In Proceedings of the Enabling Technologies for Simulation Science IX Conference, Defense & Security Symposium, March 2005, Orlando, FL, 2005.

[69] P. Gupta and R. K. Guha. “Design and Implementation of an Efficient Algorithm for Data Distribution Management in High Level Architecture”. In Proceedings of the 4th Symposium on Design, Analysis, and Simulation of Distributed Systems, 2006 Spring Simulation Multiconference, Huntsville, Alabama, April 2-6, 2006.

[70] P. Gupta and R. K. Guha, “Data Distribution Management for High Performance Distributed Simulation in Resource-Constraint Environment”, 2006 High Performance Computing & Simulation (HPC&S) Conference, Bonn, Germany, May 28-31, 2006.

[71] Y. Zhang, G. Sun, H. Yan, and L. Zhong. “Research on a new data distribution strategy for DDM”, In Proceedings of the International Conference on Machine Learning and Cybernetics, Vol. 2, pp. 672– 675, 2-5 November, 2003.

[72] J. B. Koh, B. S. Lee, W. T. Cai, and S. J. Turner. “'Multicasting Fast Messages in RTI-Kit”, In Proceedings of the 2000 Spring Simulation Inter-operability Workshop (SIW), 2000.

[73] I. Tacic and R. M. Fujimoto. “Synchronized data distribution management in distributed simulations”, In Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation, pp. 108–115, 26-29 May, 1998.

[74] J. Z. Wang and R. K. Guha. “A novel data caching scheme for multimedia servers”, *Simulation Practice and Theory*, Vol. 9, No. 3-5, pp. 193-213, 2001.

[75] J. Z. Wang. “Data caching and data allocation for multimedia servers”, Ph.D. Dissertation, Dept. of Computer Science, University of Central Florida, Orlando, FL, 2001.

[76] L. Arge, L. Toma, and J. Vitter. “I/O-Efficient Algorithms for Problems on Grid-based Terrains”, ALENEX'00. *ACM Journal of Experimental Algorithmics*, 6, No. 1, 2001.

[77] L. Arge. “External Memory Data Structures”, In *Handbook of Massive Data Sets*, J. Abello, P.M. Pardalos, M.G.C. Resende (Eds.), Kluwer Academic Publishers, pp. 313-357, 2002.

[78] B. Blunden. *Memory Management: Algorithms and Implementation in C/C++*. Plano, TX. Wordware Publishing, 2003.

[79] U. Meyer, P. Sanders, and J. Sibeyn. *Algorithms for Memory Hierarchies*. Springer-Verlag, Berlin, 2003.

[80] N. R. Nielsen. "Dynamic memory allocation in computer simulation", *Communications of the ACM* 20, No. 11, 864-873, 1977.

[81] D. E. Vengroff and J. S. Vitter. "I/O-Efficient Algorithms and Environments. Strategic Directions in Computing Research", *ACM Computing Surveys*, Vol. 28, No. 4es, 1996.

[82] Zorn, B. and D. Grunwald. "Evaluating Models of Memory Allocation", *ACM Transactions on Modeling and Computer Simulation* 4, No.1., pp. 107-131, 1994.

[83] Z. Yuan, W. Cai, and M.Y.H. Low. "A framework for executing parallel simulation using RTI", In Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications, pp. 12-19, 23-25 October, 2003.

[84] R. K. Guha, J. Lee, and O. Kachirski. "Evaluating performance of distributed computing technologies – HLA and TSpaces on a cluster computer", In Proceedings of the 19th European Conference on Modelling and Simulation, June 1-4, 2005, Riga, Latvia.

[85] T. Lu, C. Lee, W. Hsia and M. Lin. "Supporting Large-Scale Distributed Simulation Using HLA", *ACM Transactions on Modeling and Computer Simulation*, Vol. 10, No. 3, pp. 268–294, July 2000.

[86] Use of Cluster computing in Simulation http://www.sisostds.org/webletter/siso/iss_3/art_77.htm

[87] Y. Xie, Y. M. Teo, W. Cai, and S. J. Turner. "A distributed simulation backbone for executing HLA-based simulation over the internet", In Workshop on Grid Computing & Applications, In Proceedings of the 2nd International Conference on Scientific and Engineering Computation, pp. 96–103, June 2004.

[88] 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 (DS-RT.02), pp. 7- 14, 2002.

[89] W. Cai, P. Xavier, S. J. Turner, and B. Lee. "A scalable architecture for supporting interactive games on the Internet", In Proceedings of the sixteenth workshop on Parallel and distributed simulation. Washington, D.C., pp. 60– 67, 2002.

[90] S. J. E. Taylor, R. Sudra, T. Janahan, G. Tan, and J. Ladbrook. “Towards COTS distributed simulation using grids”, In Proceedings of the 2001 Winter Simulation Conference. pp. 1372-1379, 2001.

[91] K. Rycerz, M. Bubak, M. Malawski, and P. Sloot. “A Framework for HLA-Based Interactive Simulations on the Grid”, *SIMULATION: Transactions of the Society for Modeling and Simulation International*, Vol. 81, Issue 1, January 2005, pp. 67-76.

[92] J. Luthi and S. Grossmann. “The resource sharing system: dynamic federate mapping for HLA-based distributed simulation”, In Proceedings of the fifteenth workshop on Parallel and distributed simulation, Lake Arrowhead, CA, pp. 91-98, 2001.

[93] M. A. Bassiouni, M. Chiu, M. Loper, M. Garnsey, and J. Williams. “Performance and Reliability Analysis of Relevance Filtering for Scalable Distributed Interactive Simulation”, *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 3, July 1997, pp. 293–331.

[94] Y. Xie, Y. Teo, W. Cai, and S. J. Turner. “Service Provisioning for HLA-based Distributed Simulation on the Grid”, In Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, pp. 282 – 291, 2005.

[95] D. Chen, B. Lee, W. Cai, and S. J. Turner. “Design and Development of a Cluster Gateway for Cluster-based HLA Distributed Virtual Simulation Environments”, In Proceedings of the 36th annual symposium on Simulation, pp. 193-201, March 2003.

[96] C. D. Pham. “High Performance Clusters: A Promising Environment for Parallel Discrete Event Simulation”, In Proceedings of the PDPTA'99, June 28-July 1, 1999, Las Vegas, USA.

[97] A. Santoro and R.M. Fujimoto. “Off-Loading Data Distribution Management to Network Processors in HLA-Based Distributed Simulations”, In Proceedings of the Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT' 04). pp. 12-19.

[98] R. M. Fujimoto. *Parallel and distribution simulation systems*, New York, Wiley, 2000.

[99] R. Ayani, F. Moradi, and G. Tan. “Optimizing cell-size in grid-based DDM”, In Proceedings of Fourteenth Workshop on Parallel and Distributed Simulation, pp. 93-100, 28-31 May, 2000.

[100] S.J. Deitz, B.L. Chamberlain, and L. Snyder. “Abstractions for dynamic data distribution”, In Proceedings of the Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments, pp. 42–51, 26 April 2004.

[101] S. Ferenci, K. Perumalla, and R. Fujimoto. **“**An Approach to Federating Parallel Simulators”, In Proceedings of ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation, Bologna, Italy, May 2000.

[102] T. McLean, R. Fujimoto, and B. Fitzgibbons. “Next Generation Real-Time RTI Software”, In Proceedings of Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications, 2001.

[103] K. Perumalla, A. Park, R. Fujimoto, and G. Riley. “Scalable RTI-based Parallel Simulation of Networks”, In Proceedings of ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation (PADS), San Diego, CA, 2003.

[104] D. Coffin, M. Calef, D. Macannuco, and W. Civinskas. “Experimentation with DDM schemes”, In Proceedings of the 1999 Spring Simulation Interoperability Workshop, March 14-19, 1999, paper no. 99S-SIW-053.

[105] K. L. Morse, L. Bic, and M. Dillencourt. “Characterizing Scenarios for DDM Performance and Benchmarking RTIs”, In Proceedings of the 1999 Spring Simulation Interoperability Workshop, March 14-19, 1999, paper no. 99S-SIW-054.

[106] D. Cohen and A. Kemkes. “User-Level Measurements of DDM Scenarios”, In Proceedings of the 1998 Spring Simulation Interoperability Workshop, March 9-13, 1998, paper no. 98S-SIW-072.

[107] D. Cohen and A. Kemkes. “DDM scenarios”, In Proceedings of the 1997 Fall Simulation Interoperability Workshop, Sept 8-12, 1997, paper no. 97F-SIW-057.

[108] D. Cohen and A. Kemkes. “Using DDM - an application perspective”, In Proceedings of the 1997 Spring Simulation Interoperability Workshop, paper no. 97S-SIW-014.

[109] N. Kuijpers, J. Lukkien, and B. Huijbrechts. “Applying Data Distribution Management and Ownership Management Services of the HLA Interface Specification”, In Proceedings of the 1999 Fall Simulation Interoperability Workshop, September 12-17, 1999, paper no. 99F-SIW-023.

[110] E. S. Hougland, D. J. Paterson. “Data Distribution Management Issues for HLA implementations”, In Proceedings of the 1999 Spring Simulation Interoperability Workshop, March 26-31, 2000, paper no. 00S-SIW-130.

[111] M.A. Thomas. “HLA federation design for dismounted infantry simulation”, In Proceedings of the 2000 Fall Simulation Interoperability Workshop, September 17-22, 2000, paper no. 00F-SIW-099.

[112] A. C. Janett, S. J. Adelson, D. D. Miller, and R.A. Reynolds. “The FOM for Atmosphere, Ocean, Space and Dynamic Terrain – Environment Federation”, In Proceedings of the 2000 Fall Simulation Interoperability Workshop, September 17-22, 2000, paper no. 00F-SIW-092.

[113] B. Helfinstine, D. Wilbert, M. Torpey, and W. Civinskas. “Experiences with Data Distribution Management in Large-Scale Federations”, In Proceedings of the 2001 Fall Simulation Interoperability Workshop, September 9 - 14, 2001, paper no. 01F-SIW-032.

[114] K. Snively and A. Wilson. “Scalable Reliable Data Dissemination for Distributed Simulations using Hierarchical Interconnect”, In Proceedings of the 2004 Spring Simulation Interoperability Workshop, April 18 - April 23, 2004, paper no. 04S-SIW-073.

[115] S. J. Rak, M. Salisbury, R. S. MacDonald. “HLA/RTI Data Distribution Management in the Synthetic Theater of War”, In Proceedings of the 1997 Fall Simulation Interoperability Workshop, Sept 8-12, 1997, paper no. 97F-SIW-119.

[116] Katherine L. Morse. “The Object Model Template Routing Space Table: Recording Federation-Global DDM Decisions”, In Proceedings of the 1998 Spring Simulation Interoperability Workshop, March 9-13, 1998, paper no. 98S-SIW-172.

[117] S. Brunett and T. Gottschalk. “An Architecture for Large ModSAF Simulation Using Scalable Parallel Processors”, In Proceedings of the 1998 Spring Simulation Interoperability Workshop, March 9-13, 1998, paper no. 98S-SIW-180.

[118] G. Wagenbreth, K. Yao, D. M. Davis, R. F. Lucas and T. D. Gottschalk. “Enabling 1,000,000-Entity simulation on Distributed Linux clusters”, In Proceedings of the 2005 Winter Simulation Conference, Orlando, FL, Dec. 4-7, 2005, pp. 1170-1181.

[119] yaRTI. http://perso.wanadoo.fr/dominique.canazzi/dominique.htm

[120] XRTI: Extensible Run-Time Infrastructure. http://www.npsnet.org/~npsnet/xrti/

[121] CERTI. http://certi.nongnu.org/, http://www.cert.fr/CERTI/

[122] CHRONOS: http://www.magnetargames.com/Products/Chronos/

[123] pRTI 1516. http://www.pitch.se/