

A Comparative Study of Data Distribution Management Algorithms

Pankaj Gupta

School of Electrical Engineering & Computer Science
University of Central Florida
Orlando, FL 32816, U.S.A.
pgupta@cs.ucf.edu

Ratan K. Guha

School of Electrical Engineering & Computer Science
University of Central Florida
Orlando, FL 32816, U.S.A.
guha@cs.ucf.edu

In large-scale distributed defense modeling and simulation, Data Distribution Management (DDM) controls and limits the data exchanged reducing the processing requirements of federates. In this paper, we present a comparative study of a recently proposed DDM algorithm, called *P-Pruning algorithm*, with three other known techniques: region-matching, fixed-grid, and dynamic-grid DDM algorithms. By populating the multicast group, first only on the basis of X-axis information of routing space, and pruning the non-overlapping subscriber regions within multicast groups in successive steps, the P-Pruning algorithm avoids the computational overheads of other algorithms. From the simulation study, we found that the P-Pruning algorithm is faster than the other three DDM algorithms. The performance evaluation results also 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. We also present the design and implementation of a memory-efficient, scalable enhancement to the P-Pruning algorithm.

Keywords: Data Distribution, High Level Architecture, Distributed Simulation, Multicast, Scalability.

1. 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 purpose 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 taking 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 $O(n^2)$, where n is the number of objects or federates. DDM 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 Contributions and Scope

The main contribution of this paper is the comparison of a recently proposed algorithm, called P-Pruning algorithm, for the DDM problem with three techniques: region-matching, fixed-grid, and dynamic-grid DDM algorithms. We also present the design and implementation of a memory-efficient, scalable enhancement to the P-Pruning algorithm. In the simulation experiments, the P-Pruning algorithm performed better than these DDM methods in terms of computation time, memory utilization at runtime, and number of multicast groups required. We have used IEEE 1516 specifications for representing the federates, and their publisher and subscriber regions.

Table 1. List of symbols

P	Set of publisher regions
S	Set of subscriber regions
F	Set of federates
MCG	Multicast Group
$MCG_i:P_{jk}$	A set of subscriber regions: Multicast group i of a publisher region P_j of federate k and a set of subscriber regions
G	Grid cells
R	Length of routing space on X-axis & Y-axis
n	Number of federates
$FedID$	Federate Identifier
$Region Projection$	Array for storing information on publisher and subscriber regions
Pub	Entry in $Region Projection$ array for publisher regions
S_{X1}	Entry in $Region Projection$ array for subscriber regions
S_{X2}	Entry in $Region Projection$ array for subscriber regions

In this sub-section, we will outline the assumptions used to develop the P-Pruning DDM algorithm. The symbols used in this paper are shown in Table 1. The P-Pruning algorithm is based on: (i) projecting the coordinates of the publisher and subscriber regions of federates on the X-axis of the multidimensional routing space and (ii) correcting the overlap information by successive pruning along the remaining axis. However, for illustration and simplicity of presentation, throughout the paper, we consider a two-dimensional square routing space $R \times R$, where R is the length of routing space on both X and Y axis. The results can be easily extended to a multidimensional routing space. All the publisher and subscriber regions are rectangular regions within the square routing space. We have used a two end-point system to represent a single region in the routing space. A single region represents a contiguous set of data in this mapping. This paper focuses on improvement of DDM services in HLA and hence, other methods of inter-federate communications are not considered. The P-Pruning algorithm considers federates having rectangular publisher and subscriber regions within the routing space. In addition, these regions and the routing space have integer coordinates.

Detailed pseudo-code of the P-Pruning algorithm with average-case computational complexity analysis appears in Gupta and Guha ([1], [2]). The P-Pruning algorithm creates multicast groups based on the region overlap information within the routing space. Each multicast group has one publisher region and at least one subscriber region as members; all regions are owned by their respective federates. This implies that the federate having publisher region in a multicast group sends messages to the federates having subscriber region in the multicast group. Due to these underlying assumptions, the P-Pruning algorithm may be viewed as restricted DDM algorithm subject to the conditions described here.

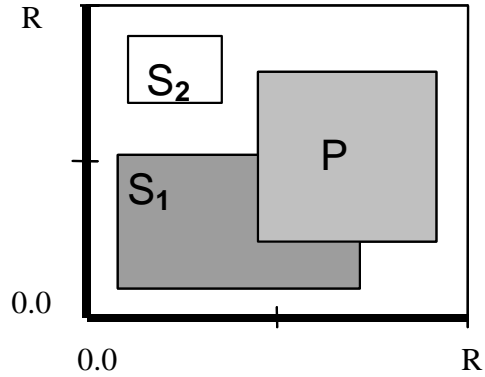


Figure 1. Two-dimensional routing space with subscriber regions: S_1 and S_2 , and a publisher region P .

1.2 Organization

The remainder of this paper is organized as follows. Section 2 provides a background on the importance of data distribution techniques, HLA-compliant implementations, and a review of related work in DDM research. Section 3 presents the P-Pruning algorithm with its three procedures and a detailed illustration of the step-by-step execution. In addition, Section 3 also overviews the three DDM algorithms: region-matching, fixed-grid, and dynamic-grid with an illustration using the same example. The performance-evaluation of the four DDM algorithms with implementation details and simulation results are discussed in Section 4. A scalable, memory-efficient enhancement to the P-Pruning algorithm is proposed in Section 5. Finally, Section 6 presents the concluding remarks with directions on future research work.

2. Motivation and Background

In this section, we discuss the importance of DDM and explain the IEEE 1516 specifications. We also provide an overview of related research work.

2.1 Importance of DDM

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 [3], load-balancing schemes [4], Web services [5], and parallel computing [6]. 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.

2.2 DDM in HLA Implementations

HLA-compliant implementation is available from DoD, universities, and several companies in industry. Table 2 lists some of these RTI implementations with their year of first release. DDM forms an important component in any HLA implementation. The region-matching DDM is most commonly used method in these HLA implementations (*e.g.*, RTI 1.3, DMSO RTI). Due to proprietary restrictions, the exact type of DDM techniques implemented in the commercial HLA versions is not publicly known. However, presenting a comparative study of several algorithms may benefit the developers in upgrading these software.

Table 2. List of HLA implementations

HLA	Developer	Release Year
RTI 1.3	MIT Lincoln Labs	1996
DMSO RTI	DoD	1996
GMU RTI	George Mason University	1997
pRTI	Pitch AB	2000
FDK	Georgia Tech.	2000
MÄK RTI	MÄK Technologies	2002
HPC RTI	RAM Labs	2002
RTI-NG	SAIC & VTC	2002
XRTI	Open Source	2003

2.3 Routing Space, Region, and Dimension in IEEE 1516

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 1, updates using publisher region P are routed to federates subscribing to region S_1 , but not to federates subscribing to region S_2 .

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. The regions are composed of dimension name and range pairs.

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.

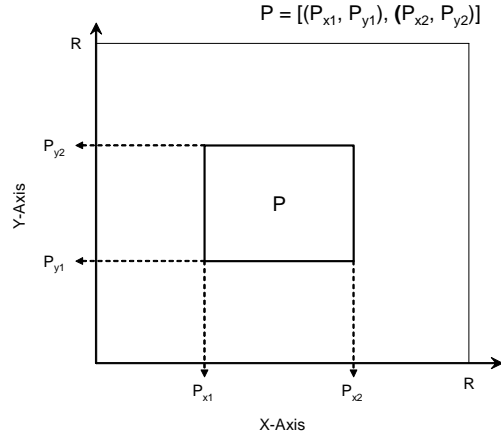
Dimension: Dimensions correspond to simulation data and they are used to define regions.

Throughout this paper, unless otherwise stated, there are two dimensions in routing space represented by the X and Y axis. Petty [7] and Morse [8] have discussed the migration of HLA 1.3 based simulation system to IEEE 1516 standards in detail.

Given a set of federates, F , each federate F_i has publisher and subscriber regions within the routing space. The DDM problem is to find a set of multicast groups MCG , whose each element MCG_i is a multicast group. Each multicast group element MCG_i is composed of several federates at any time t . Within each multicast group element, MCG_i , there is one federate F_p with a publisher region and one or more federate(s) F_s with subscriber regions such that the publisher region of federate F_p overlaps with the subscriber regions of other federates F_s in MCG_i at time t .

2.4 Related Work

The earliest work on DDM research appears in Van Hook *et al.* [9]. The HLA specification, key elements in its architecture, and implementation are described in Dahmann *et al.* [10] and Van Hook [11]. An overview with tutorial about DDM and related research work appears in ([12], [13]).



A publisher region P in routing space of dimension $R \times R$

Figure 2. Routing space layout for a single publisher region P

Boukerche and Roy [14] described taxonomy of DDM schemes and basic concepts. Petty [7] presented a comparison of the HLA 1.3 and IEEE 1516 HLA specifications. Since 1995, different DDM algorithms have been proposed such as the fixed-grid [15], dynamic-grid [14], region-matching [7], agent-based [16], and hybrid-method [17]. A sort-based algorithm running in $O(n^2)$ time is presented in [18], where n is the number of federates. Performance-evaluation study of different DDM strategies appears in ([1], [2], [19]). Scalability-related issues for the implementation of DDM are addressed in [20].

In their doctoral dissertations, Morse [21] and Petty [22] made numerous algorithmic contributions to the DDM problem. The problem of dynamic multicast grouping is addressed in [21], while an interval-tree based DDM algorithm is presented in [22]. A connection-graph based cost-function approach is proposed in [21]. In this method, the latency of data communication is taken into account.

Federated Simulations Development Kit (FDK) is an implementation of HLA architecture developed at Georgia Institute of Technology ([23], [24]). 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], [15], [19], [25], [26]). Scalability issues of FDK have been researched by Riley *et al.* in [27]. Synchronization issues in DDM have been explored and their remedies have been proposed in [28]. In Section 6, 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.* [5]. 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 [29].

3. The Region-Matching, Grid-Based, and P-Pruning DDM Algorithms

In this section, we provide a brief overview of three DDM algorithms used in current HLA implementations and also present the P-Pruning algorithm. We illustrate each algorithm with a common example involving three federates. A detailed presentation of the P-Pruning algorithm with pseudo-codes and average-case computational complexity analysis appears in Gupta and Guha ([1], [2]).

The symbols and notations used to describe the algorithms and their illustrations are listed in Table 1 of Section 1.1. Each publisher region P_i is represented by a two end-point system $[(P_{i_{x1}}, P_{i_{y1}}), (P_{i_{x2}}, P_{i_{y2}})]$ in the two-dimensional routing space (with dimensions $R \times R$) as shown in Figure 2. Similarly, each subscriber region S_i is represented by coordinates: $[(S_{i_{x1}}, S_{i_{y1}}), (S_{i_{x2}}, S_{i_{y2}})]$.

3.1 Region-Matching Algorithm

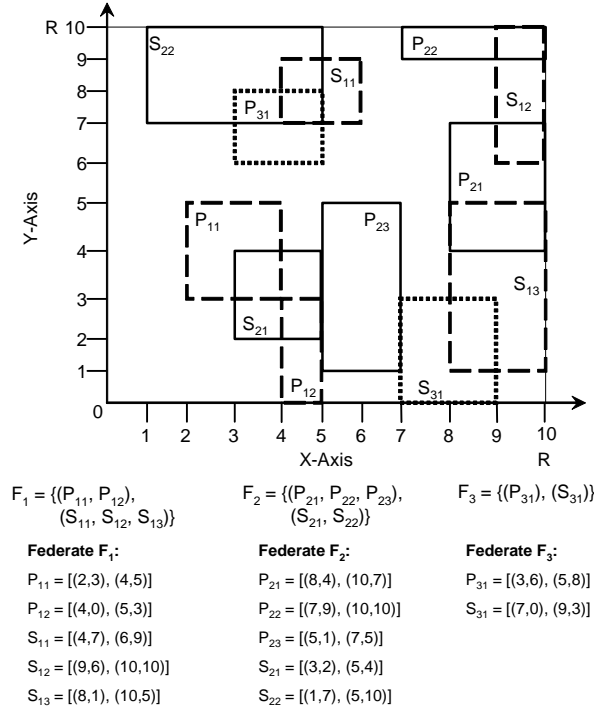


Figure 3. Routing space layout for illustration of DDM algorithms

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 identify those publisher regions 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. This algorithm needs to scan all the publisher and subscriber regions at least once. Hence, its time complexity is quadratic.

3.1.1 Illustration of Region-Matching Algorithm

We now illustrate the execution of region-matching algorithm using the example in Figure 3. This example considers a two-dimensional routing space of size 10 x 10 units. The set of federates $F = \{F_1, F_2, F_3\}$ such that federate F_i has two publisher regions (P_{i1}, P_{i2}) and three subscriber regions (S_{i1}, S_{i2}, S_{i3}). 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} . In Figure 3, we have depicted the publisher and subscriber regions of federates F_1, F_2 , and F_3 using long-dashed line, solid line, and square-dotted line, respectively as well as their coordinates in the routing space. The region-matching algorithm compares the publisher regions of each federate with all the subscriber regions of other federates. A multicast group (MCG) is allocated to each publisher region as follows:

$MCG_1: P_{11}, MCG_2: P_{12}, MCG_3: P_{21}, MCG_4: P_{22}, MCG_5: P_{23}$, and $MCG_6: P_{31}$.

Each publisher region is matched against the subscriber regions of other federates in the simulation system and the overlapping subscriber regions are added to each multicast groups. For example, in MCG_1 , the coordinates of publisher region P_{11} are compared with the subscriber regions S_{21}, S_{22} , and S_{31} . Since, S_{21} overlaps with P_{11} , it is added to multicast group MCG_1 . The same steps are repeated for each multicast groups. The final list of multicast groups is given as:

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

- $MCG_2: P_{12} = \{S_{21}\}$,
- $MCG_3: P_{21} = \{S_{12}, S_{13}\}$,
- $MCG_4: P_{22} = \{S_{12}\}$, and
- $MCG_6: P_{31} = \{S_{11}, S_{22}\}$.

The multicast group MCG_5 is deleted as no subscriber region overlaps with its publisher region P_{23} . Since, the publisher and subscriber regions are owned by federates, the actual messages are sent from the publishing federate to the subscribing federate. The region-matching can be computationally expensive, and it does not scale as number of federates is increased.

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. For example, in Figure 3, only 4 multicast groups are needed for a 5 x 5 grid cell, while 25 multicast groups are required for a 2 x 2 grid cell.

3.2.1 Illustration of Fixed-Grid DDM

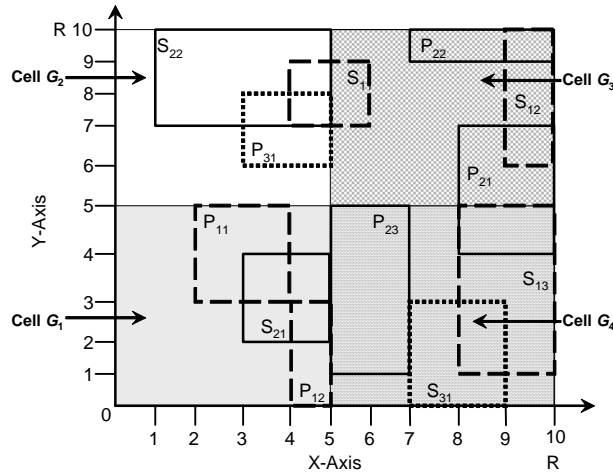


Figure 4. Grid partitioning for routing space into 5 x 5 cells

Now, we illustrate the execution of fixed-grid DDM method for the same example in Figure 3. For this illustration, we have divided the routing space into grid cells of 5 x 5 units as shown in Figure 4. There are four grid cells G_1 , G_2 , G_3 , and G_4 indicated by different shades. A multicast group is assigned to each grid cell. Thus, four multicast groups are created as follows:

$$MCG_1: G_1, MCG_2: G_2, MCG_3: G_3, \text{ and } MCG_4: G_4.$$

The federates become publishing or subscribing members of each multicast group depending on the overlap information of their respective publisher and subscriber region with each grid cell. Table 3 shows the region-cell overlap information and the status of each publisher and subscriber region at the end of this algorithm.

Table 3. State of multicast groups in fixed-grid algorithm

Grid Cell	Multicast Group	Publisher		Subscriber	
		Region	Federate	Region	Federate
G_1	MCG_1	P_{11}, P_{12}	F_1	S_{21}	F_2
G_2	MCG_2	P_{31}	F_3	S_{11}, S_{22}	F_1, F_2
G_3	MCG_3	P_{22}, P_{21}	F_2	S_{11}, S_{12}	F_1
G_4	MCG_4	P_{21}, P_{23}	F_2	S_{13}, S_{31}	F_1, F_3

Thus, in this method, there is no explicit matching of publisher and subscriber regions. The publishing federates send messages to subscribing federate as indicated by the Table 3.

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.

3.3.1 Illustration of Dynamic-Grid DDM

To illustrate the execution of dynamic-grid method, we use the same example in Figure 3 with one exception that subscriber regions S_{11} and S_{22} do not exist. The multicast groups are created in the same manner as explained in Section 3.2.1 for the fixed-grid method. However, this step does not assign a multicast group to the grid cells that do not have an overlapping subscriber region. Since, S_{11} and S_{22} are not present in this example, grid cell G_2 will not have a subscriber region, and hence it is not allocated a multicast group. The final list of multicast region with the status of each federate is shown in Table 4.

Table 4. State of multicast groups in dynamic-grid algorithm

Grid Cell	Multicast Group	Publisher		Subscriber	
		Region	Federate	Region	Federate
G_1	MCG_1	P_{11}, P_{12}	F_1	S_{21}	F_2
G_3	MCG_2	P_{22}, P_{21}	F_2	S_{11}, S_{12}	F_1
G_4	MCG_3	P_{21}, P_{23}	F_2	S_{13}, S_{31}	F_1, F_3

The results differ from the fixed-grid method as there are only three multicast groups created: MCG_1 : G_1 , MCG_2 : G_3 , and MCG_3 : G_4 . This eliminates unnecessary transmission of messages to federates having no subscribers.

3.4 The P-Pruning DDM Algorithm

In this sub-section, we describe the P-Pruning algorithm for DDM and also illustrate it using the example in Figure 3. The P-Pruning algorithm computes the multicast groups in three steps: List Computation, MCG Population, and MCG Pruning. The central idea in this algorithm is to compute the matching of publisher and subscriber regions first on the basis of the overlap on X-axis only and then correct the overlap information by checking the Y-axis.

The algorithm can be understood by following the step-by-step execution of the example in Figure 3 as described in Section 3.4.1. The entire algorithm is based on an array data structure, called the *Region Projection array*, whose size is equal to the length (R) of the routing space X-axis. The *Region Projection* array stores the information about coordinates of the publisher and subscriber regions in the routing system. The elements (or points) in the *Region Projection* array correspond to the coordinates in X-axis of the routing space which have integer values.

List Computation: The List Computation procedure scans the publisher and subscriber regions of all the federates once, and stores the information about their coordinates at each element of the *Region Projection* array. At the end of the List

Computation procedure, each element of *Region Projection* array has three entries: publisher region (*Pub*), X1 subscriber region (*S_X1*), and X2 subscriber region (*S_X2*). Each element x of the Region Projection array stores information about the publisher region (denoted as PUB) and subscriber region end points (denoted as *S_X1* and *S_X2*) of a federate. Each of the region entry is composed of two distinct data: region counter and federate ID (*FedID*) information for each publisher and subscriber region.

For every element (or point) x of *Region Projection* array, the publisher region (PUB) entry contains either an empty set (i.e., no federate with publisher region having its $(P_i)_{x_1}$ at coordinate x) indicated by “*” or contains a set of two elements: one containing number of publisher regions P_i , whose $(P_i)_{x_1}$ coordinate coincides with element x and other containing the federate ID owning these publisher regions.

Similarly, for element (or point) x of the Region Projection array, the *S_X1* entry contains either an empty set (i.e., no federate with subscriber region having its $(S_j)_{x_1}$ at coordinate x) indicated by “*” or contains a set of two elements: one containing number of subscriber regions whose $(S_j)_{x_1}$ (i.e, lower) coordinate coincides with element x and other containing the federate ID owning these subscriber regions.

Also, for element (or point) x of the Region Projection array, the *S_X2* entry contains either an empty set (i.e., no federate with subscriber region having its $(S_j)_{x_2}$ at coordinate x) indicated by “*” or contains a set of two elements: one containing number of subscriber regions whose $(S_j)_{x_2}$ (i.e. higher) coordinate coincides with element x and other containing the federate ID owning these subscriber regions.

Thus, the X1 subscriber region entry (*S_X1*) corresponds to all subscriber regions S_j , whose $(S_j)_{x_1}$ coordinate coincides with element x and X2 subscriber region entry (*S_X2*) corresponds to the all the subscriber regions S_j , whose $(S_j)_{x_2}$ coordinate coincides with element x . In our implementation system, we have used the identifier (called Federate ID) of the federate owning each region to uniquely identify each publisher region. In addition, each element of region projection array stores the identifier (called counter) for its list of publisher region. Thus, a combination of federate ID and counter can uniquely identify each publisher and subscriber region in our representation system. This idea becomes clearer in our example in section 3.4.1.

MCG Population: The MCG Population procedure scans each element of the *Region Projection* array and checks the coordinates of publisher region, X1 subscriber region, and X2 subscriber region for overlap condition. This procedure creates the DDM multicast groups based on information stored in *Region Projection* array, but it considers only the overlap information on X-dimension of the routing space. A multicast group is assigned to an element of the *Region Projection* array, if there is a publisher region P_i whose $(P_i)_{x_1}$ coordinate coincides with this element of *Region Projection*. Also, a multicast group is created at an element on *Region Projection* only if there exists at least one publisher region P_i whose $(P_i)_{x_1}$ coordinate coincides with this element and there is at least one subscriber region overlapping with this publisher region on X-axis. As stated in Section 1.1, regions in a multicast group are owned by their respective federates. When a region is included in a multicast group, it implies that the federate owning this particular region is also actually a member of this multicast group. Thus, this 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 procedure computes the entire information faster by avoiding the simultaneous checking of X-axis and Y-axis overlap.

MCG Pruning: The errors in creation of multicast group *MCG* are now corrected by the final MCG Pruning procedure. The pruning 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.

Region Projection Array Data structure: We have used a two-dimensional data structure to represent the *Region Projection* array information. Here, we describe this data structure and explain its function. In actual implementation, this array is stored using instances of classes, where each class instance represents a point on the X-axis of the routing space. Thus, the *Region Projection* array is a collection of points on the X-axis, where the information about the publisher and subscriber region that have X1 and X2 coordinates incident with these points is stored. In our representation system, each publisher and subscriber region is uniquely identified by its federate ID. Hence, the *Region Projection* array stores this Federate ID for each entry.

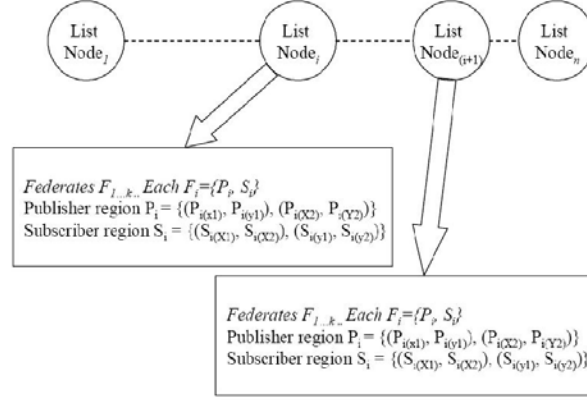


Figure 5. Representation of Memory-Inefficient Region Projection Array

Also, since every element of this array is a set of publisher regions and the two types of subscriber regions, it also stores counters for each of these region types. We are introducing the federate ID and counters terms to help understand the functioning of the algorithm and the example in Section 3.4.1.

When there are more than one publisher regions whose x_1 coordinate have same integer values, then multicast groups are created for each publisher regions. In case there is a subscriber region intersecting with these publisher regions, then this multicast is not eliminated. The *Region Projection* array stores federate ID (*FedID*) information for each publisher and subscriber region entry. This helps in identifying the federate owning a publisher or subscriber region. Thus, the components of *Region Projection* array: *Pub*, *S_X1*, and *S_X2*, are containers that store two variables: (i) Region Counter and (ii) Federate ID (*FedID*) of the federate owning the publisher or subscriber region. The region counters are of three types: publisher region counter, X1 subscriber region counter and X2 subscriber region counter. These region counters are used to store the number of regions that coincide with a given element x of the region projection array: publisher region counter records the number of publisher regions, whose $(P_i)_{x_1}$ coordinate coincides with element x ; X1 subscriber region counter records the number of subscriber regions, whose $(S_j)_{x_1}$ coordinate coincides with element x ; and X2 subscriber region counter records the number of subscriber regions, whose $(S_j)_{x_2}$ coordinate coincides with element x . These details are clearer in Section 3.4.1 using an example. Figure 5 shows the federates stored at each list node. Each node records the federate ID and also the publisher/subscriber region associated with these federates. This helps in identifying each region uniquely.

Since the two main procedures in the algorithm perform the function of multicast group *Population* and *Pruning*, the new algorithm is called *P-Pruning* algorithm. 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. 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 unwanted subscriber regions within multicast groups in another step, it avoids the computational overheads of other algorithms.

The List Computation procedure has complexity of $O(n)$, where n is the number of federates in the distributed simulation. The MCG Computation 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 procedure, complexity is $O(n)$. 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. A detailed pseudo-code of the P-Pruning algorithm with average-case computational complexity analysis appears in Gupta and Guha ([1], [2]).

We now provide explanation on how the P-Pruning algorithm can be adapted for dynamic situations. DDM is a dynamic process and efficient matching process should take advantage of the fact that part of the data in a simulation is relatively static. This is the basic premise in development of P-Pruning algorithm. A dynamic situation can be viewed as discrete event using finite time steps. In the near future, we will develop a linear-time routine for tracking federates joining and leaving the federation. Using this routine, the P-Pruning algorithm can be extended to accommodate dynamic conditions. The MCG Pruning procedure will eliminate the regions that no longer overlap due to changes in coordinates and provide accurate

results. We will also develop subroutines which take advantage of the region-distribution and facilitate joining and resigning of federates at run-time.

3.4.1 Illustration of P-Pruning Algorithm

Now, we will illustrate step-by-step execution of the P-Pruning algorithm using the example shown in Figure 3. This 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.

List Computation Step: The *Region Projection* [0, 1,2,3,...,10] array stores three important details for each point on X-axis: *Pub* entry (publisher region counter and Federate ID), *S_X1* entry (X1 subscriber region counter, and Federate ID), and *S_X2* entry (X2 subscriber region counter and Federate ID). Thus, each entry in the *Region Projection* array has two components: Region Counter and Federate ID. The Federate ID helps in identifying the federate owning a publisher or subscriber region. For a given element x in *Region Projection* array: publisher region counter records the number of publisher regions, whose $(P_i)_{x_1}$ coordinate coincides with element x ; X1 subscriber region counter records the number of subscriber regions, whose $(S_j)_{x_1}$ coordinate coincides with element x ; and X2 subscriber region counter records the number of subscriber regions, whose $(S_j)_{x_2}$ coordinate coincides with element x .

Table 5. Status of *Region Projection* (R. P.) array after List Computation step.

Each entry has (region counter, federate ID) pair.

	Region Projection Array					
	Pub		S_X1		S_X2	
	Count	FedID	Count	FedID	Count	FedID
0	*		*		*	
1	*		1	2	*	
2	1	1	*		*	
3	1	3	1	2	*	
4	1	1	1	1	*	
5	1	2	*		2	2
6	*		1	1	1	1
7	1	2	1	3	*	
8	1	2	1	1	*	
9	*		1	1	1	3
10	*		*		2	1

In this step, the publisher and subscriber regions of each federate, F_1, F_2 and F_3 , are examined. After this, the *Region Projection* element corresponding to the X_1 and X_2 coordinates of each publisher and subscriber region is updated. The state of *Region Projection* array at the end of this step is shown in Table 5. The table shows (region counter, federate ID) pair for each entry. We have abbreviated the region counter as ‘Count’ and federate ID as ‘FedID’ in the table header, respectively. 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 element of *Region Projection* array.

For each element in the *Region Projection* array, the region counter in each entry of column *Pub* holds the number of publisher regions whose $(P_i)_{x_1}$ coordinate coincides with this element. For each element of *Region Projection* array, ‘*’ in *Pub* column indicates that there are no publisher regions whose $(P_i)_{x_1}$ coordinate coincides with this element of *Region Projection* array. Thus, the region counter for each element on *Region Projection* (in column *Pub*) records the number of

publisher regions, whose $(P_i)_{x_1}$ coordinate coincides with this element. The federate ID (or FedID) corresponds to the identifier for the federate owning the publisher region P_i . For each element in the *Region Projection* array, the region counter in each entry of column S_X1 holds the number of subscriber regions whose $(S_j)_{x_1}$ coordinate coincides with this element. For each element of *Region Projection* array, '*' in S_X1 column indicates that there are no subscriber regions whose $(S_j)_{x_1}$ coordinate coincides with this element of *Region Projection* array. Thus, the region counter for each element on *Region Projection* (in column S_X1) records the number of subscriber regions, whose $(S_j)_{x_1}$ coordinate coincides with this element. The federate ID corresponds to the identifier of the federate owning the subscriber region S_j . For each element in the *Region Projection* array, the region counter in each entry of column S_X2 holds the number of subscriber regions whose $(S_j)_{x_2}$ coordinate coincides with this element. For each element of *Region Projection* array, '*' in S_X2 column indicates that there are no subscriber regions whose $(S_j)_{x_2}$ coordinate coincides with this element of *Region Projection* array. The federate ID corresponds to the identifier for the federate owning the subscriber region S_j .

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 *Publisher* (Pub) column of *Region Projection* array entries, from the Table 5 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.

The element [2] in *Region Projection* array 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 *Region Projection* array is scanned from the range of $(P_{11})_{x_1}$ to $(P_{11})_{x_2}$ (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 element [3] of *Region Projection* array in Table 5, which implies that S_{21} overlaps with P_{11} . Hence, S_{21} is added to multicast group MCG_1 . In addition to this, the *Region Projection* array is also scanned from 0 to $(P_{11})_{x_1}$ (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 element [1] of *Region Projection* array in Table 5, 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, *Region Projection* array is scanned in range $(P_{11})_{x_1}$ to $(P_{11})_{x_2}$ (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 *Region Projection* array having a publisher region entry in its *pub* position is element [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 *Region Projection* array is scanned from the range of $(P_{31})_{x_1}$ to $(P_{31})_{x_2}$ (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 element [3] of *Region Projection* array in Table 5, 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 element [4] in *Region Projection* array in Table 5, 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 *Region Projection* array is also scanned from 0 to $(P_{31})_{x_1}$ (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 element [1] of *Region Projection* array in Table 5, 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, *Region Projection* array is scanned in range $(P_{31})_{x_1}$ to $(P_{31})_{x_2}$ (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.

$$\begin{aligned} MCG_1: P_{11} &= \{S_{21}, S_{22}\}, \\ MCG_2: P_{31} &= \{S_{11}, S_{21}, S_{22}\}, \\ MCG_3: P_{12} &= \{S_{21}, S_{21}, S_{22}\}, \end{aligned}$$

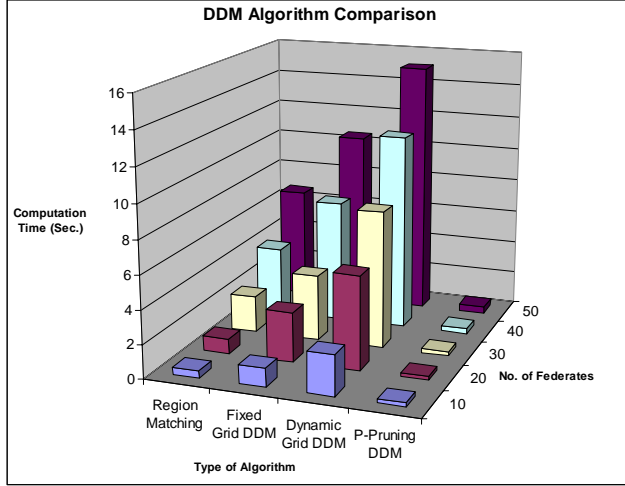


Figure 6. Performance evaluation of the P-Pruning algorithm for routing space 50 x 50 and grid size 2 x 2

$$\begin{aligned}
 MCG_4: P_{23} &= \{S_{11}\}, \\
 MCG_5: P_{22} &= \{S_{12}, S_{13}, S_{31}\}, \text{ and} \\
 MCG_6: P_{21} &= \{S_{12}, S_{13}, S_{31}\}.
 \end{aligned}$$

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:

$$\begin{aligned}
 MCG_1: P_{11} &= \{S_{21}\}, \\
 MCG_2: P_{31} &= \{S_{11}, S_{22}\}, \\
 MCG_3: P_{12} &= \{S_{21}\}, \\
 MCG_5: P_{22} &= \{S_{12}\}, \text{ and} \\
 MCG_6: P_{21} &= \{S_{12}, S_{13}\}.
 \end{aligned}$$

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. Performance Evaluation of DDM Algorithms

In this section, 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.

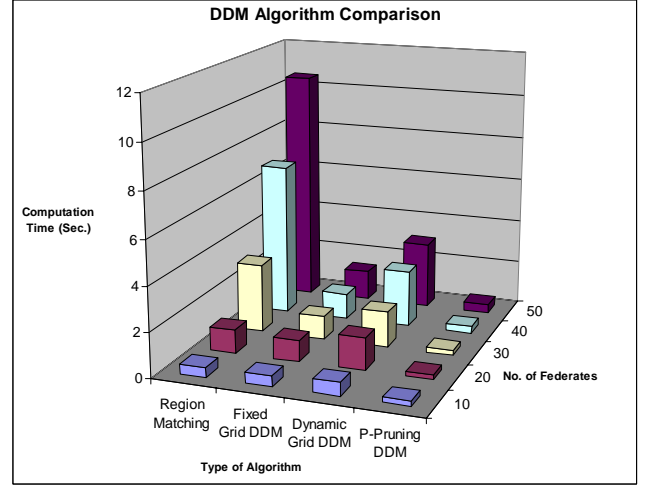


Figure 7. Performance evaluation of the P-Pruning algorithm for routing space 50 x 50 and grid size 5 x 5

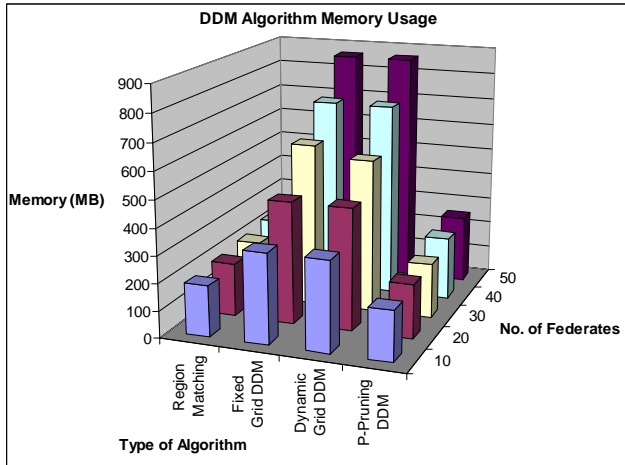


Figure 9. Comparison of memory usage by DDM algorithms for routing space 50 x 50 and grid size 2 x 2

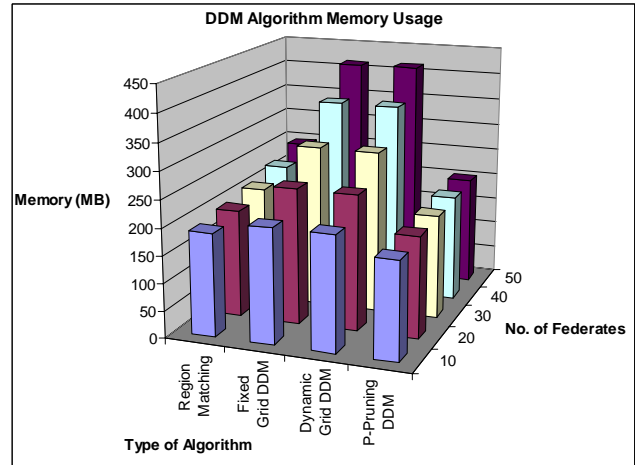


Figure 10. Comparison of memory usage by DDM algorithms for routing space 50 x 50 and grid size 5 x 5

4.1 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. Also, in the performance evaluation environment, each federate represents a single object. Hence, each federate has one publisher region and one subscriber region. In the graphs from Figures 6 through 14, 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 simulation results shown in these graphs use the data averaged for 25 instances of each condition, *i.e.*, size of routing space and number of federates. The grid cell dimensions are applicable only for the fixed-grid and dynamic-grid algorithms. Each set of input conditions had following number of federates in this simulation environment: (i) 10, 20, 30, 40, and 50 for the 50 x 50 routing space and (ii) 20 and 40 for the 100 x 100 routing space. In all the graph charts, the data corresponding to the P-Pruning DDM algorithm is referred as *P-Pruning DDM*. To ensure consistency in the simulation results, all DDM algorithms access the same federates at run-time. The graphs in Figures 6, 7, and 8 show the comparison of computation time required for different DDM algorithms. The routing space is 50 x 50 in Figures 6 and 7, while it is 100 x 100 for Figure 8. The grid size is 2 x 2 for Figure 6, and 5 x 5 for Figures 7 and 8. The results show that the P-Pruning DDM algorithm computes the multicast groups for DDM faster than any of the three algorithms.

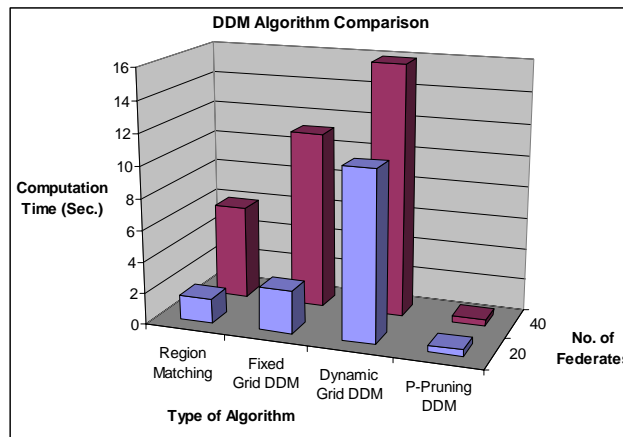


Figure 8. Performance evaluation of the P-Pruning algorithm for routing space 100 x 100 and grid size 5 x 5

We compared the memory usage at run-time for the four algorithms and the results are shown in Figures 9, 10, and 11. The routing space is 50 x 50 in Figures 9 and 10, while it is 100 x 100 for Figure 11. The grid size is 2 x 2 for Figure 9, and 5 x 5 for Figures 10 and 11. 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.

All DDM algorithms provide the multicast groups as an output. Hence, the size of the multicast groups is an important metric for evaluating the performance of DDM algorithms. The graphs in Figures 12, 13, and 14 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 Figures 12 and 13, while it is 100 x 100 for Figure 14. The grid size is 2 x 2 for Figure 12, and 5 x 5 for Figures 13 and 14. The results show that the P-Pruning DDM algorithm requires significantly fewer multicast groups as compared to fixed-grid and dynamic-grid algorithm.

4.2 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 [30]. 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 plan to address these important aspects of DDM research in our future work.

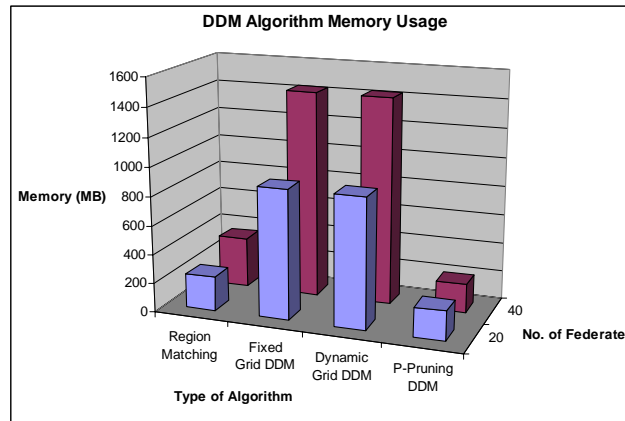


Figure 11. Comparison of memory usage by DDM algorithm for routing space 100 x 100 and grid size 5 x 5

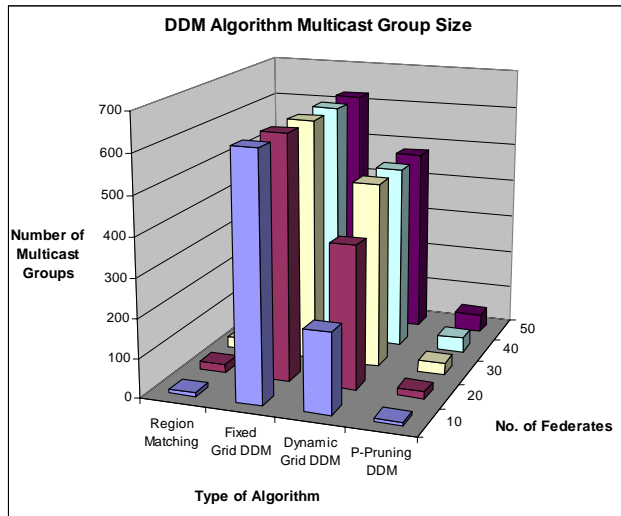


Figure 12. Comparison of multicast group size in DDM algorithms for routing space 50 x 50 and grid size 2 x 2

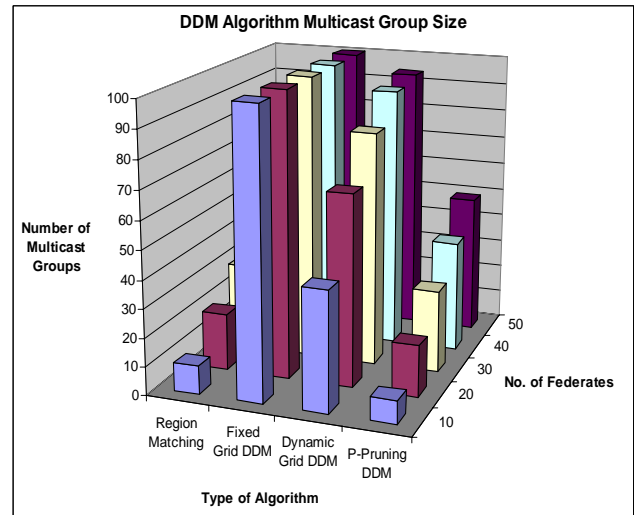


Figure 13. Comparison of multicast group size in DDM algorithms for routing space 50 x 50 and grid size 5 x 5

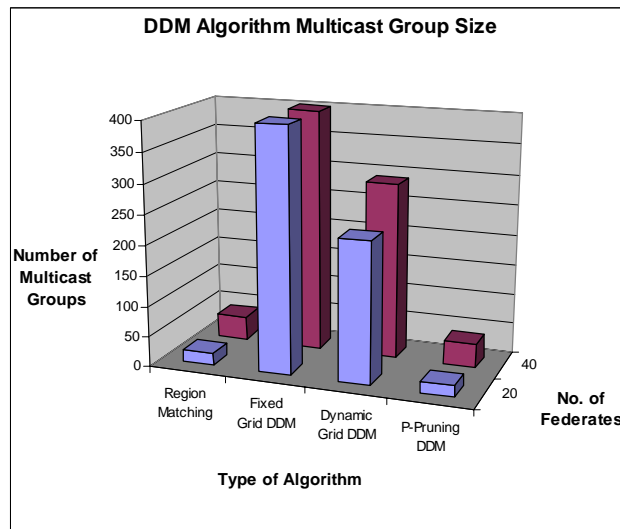


Figure 14. Comparison of multicast group size in DDM algorithms for routing space 100 x 100 and grid size 5 x 5

5. Resource-Efficient Enhancement to P-Pruning Algorithm

In this section, we describe the design and performance evaluation of a memory-efficient enhancement to the P-Pruning DDM. The simulation experiments compared the P-Pruning DDM algorithm with its memory-efficient enhancement. The study was aimed at modeling high-performance distributed simulation scenario and implemented in C++ under Windows XP running on a Pentium IV 3 GHz PC with 512 MB RAM and 2500 MB virtual memory.

5.1 Memory-Efficient P-Pruning

In memory-efficient P-Pruning algorithm, the List-Computation procedure in Section 3.4 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 *Region Projection* array in the List Computation sub-procedure.

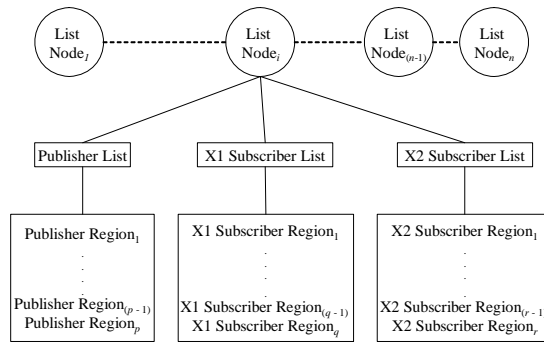


Figure 15. Representation of memory-efficient data structure

The set of nodes can be viewed as disjoint set of forests, where each node stores three different trees. We use the term disjoint set to indicate that these three lists are independent of each other and do not overlap. This representation, when implemented as a class structure, 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. The regions are listed as disjoint sets in this data structure. The improvement in the Region Projection data structure is that it reduces the memory utilized at run-time.

Data Structure Design: The structure of each node in the disjoint set is shown in Figure 15. 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 implemented using the class structure in the simulation experiments. The three lists in disjoint set are populated in List Computation procedure and the multicast groups are built using the disjoint set in the MCG population procedure.

5.2 Simulation Implementation and Analysis

In the simulation experiments, we compared the performance of P-Pruning and memory efficient P-Pruning using similar conditions as described in section 4. The simulation results shown in Figures 16, 17, and 18 use the data averaged for 100 instances of each condition, *i.e.*, size of routing space and number of federates. The graph in Figure 16 shows the comparison of memory utilized at run-time by the *memory-efficient* 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 17 shows the comparison of computation time required for the *memory-efficient* 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-efficient* version uses constant memory as compared to the P-Pruning algorithm. It also requires less computation time. The

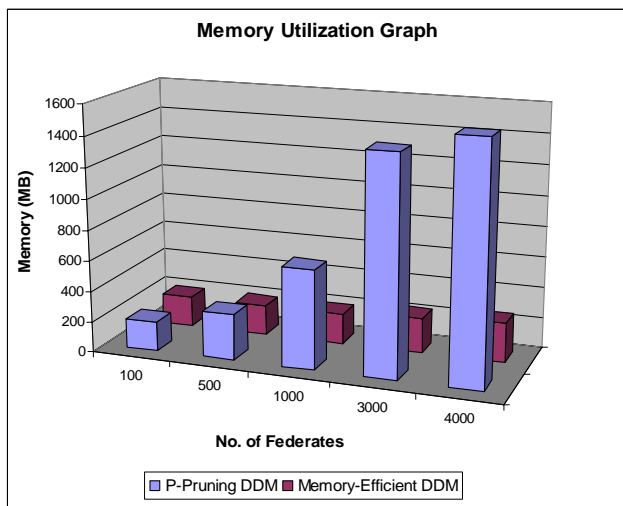


Figure 16. Comparison of memory utilization by the *memory-efficient* P-Pruning DDM algorithm

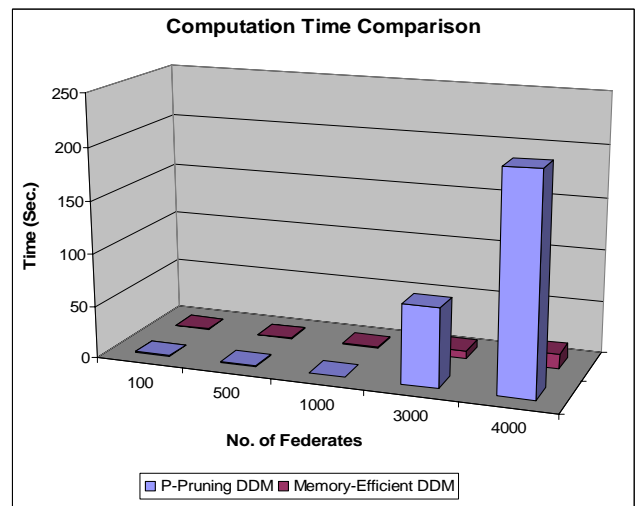


Figure 17. Comparison of computation time for routing space from 100 x 100 to 4000 x 4000

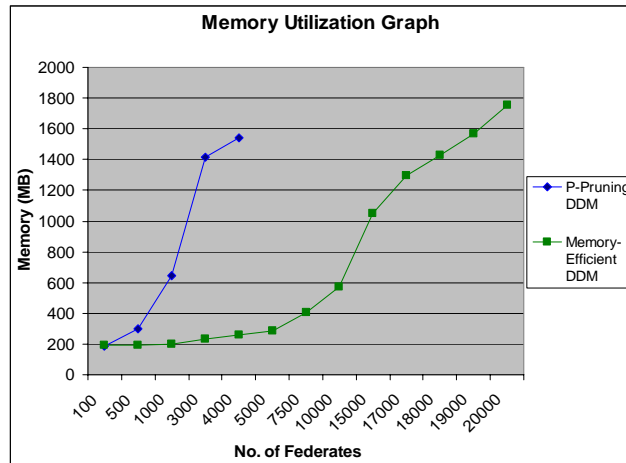


Figure 18. Memory utilization for distributed simulation with 20,000 federates

graph in Figure 18 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-efficient* P-Pruning algorithm. The P-Pruning algorithm could simulate only upto 4,000 federates due to inefficient memory utilization at run-time.

From the performance evaluation study of two versions of the P-Pruning DDM algorithm, it is evident that the *memory-efficient* 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. Conclusions and Future Work

In this paper, we presented a comparative study of the P-Pruning algorithm for DDM. In our simulation experiments, 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 three other DDM algorithms in a simulated environment. The P-Pruning methods avoids the computational overheads of other DDM algorithm by populating the multicast group, first only on the basis of X-axis information, and pruning the unwanted subscriber regions within multicast groups in another step. The memory-efficient P-Pruning algorithm requires less computation time and utilizes memory at run-time more efficiently as compared to the P-Pruning algorithm. It should be suitable for high performance distributed simulation applications as it improves the scalability of the P-Pruning algorithm.

7. Acknowledgements

The authors would like to thank the reviewers for their valuable comments. This research was partially supported by NSF under grant EIA 0086251 and ARO under grant DAAD19-01-1-0502. The views and conclusions herein are those of the authors and do not represent the official policies of the funding agencies or the University of Central Florida, Orlando.

8. References

- [1] Gupta, P. and Guha, R. K. 2005. "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.
- [2] Gupta, P. and Guha, R. K. 2006. "Design, analysis, and performance evaluation of the P-Pruning DDM algorithm", *In preparation for submission to Simulation Modelling Practice & Theory Journal*.
- [3] Burns, R. C., Rees, R. M., and Long, D. D. 2001. "Efficient data distribution in a Web server farm", *IEEE Internet Computing*, Vol. 5, No. 4, 56-65.
- [4] Chen, L. and Choi, H. 2001. "Approximation algorithms for data distribution with load balancing of Web servers", In Proceedings of the 2001 IEEE International Conference on Cluster Computing, Los Angeles, CA, pp. 274-281.

- [5] Pullen, J. M., *et al.* 2004. "Using Web services to integrate heterogeneous simulations in a grid environment", In Proceedings of the International Conference on Computational Science, Krakow, Poland.
- [6] Lee, P. 1997. "Efficient algorithms for data distribution on distributed memory parallel computers", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 8, 825-839.
- [7] Petty, M. D. 2002. "Comparing high level architecture data distribution management specifications 1.3 and 1516", *Simulation Practice and Theory*, Vol. 9, No. 3-5, 95-119.
- [8] Morse, K. L. and Petty, M. D. 2001. "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, 13-15 Aug., Cincinnati, OH, pp. 58-65.
- [9] Van Hook, D. J., Rak, S. J., and Calvin, J. O. 1996. "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.
- [10] Dahmann, J. S., Fujimoto, R. M., and Weatherly, R. M. 1998. "The DoD high level architecture: an update", In Proceedings of the 1998 Winter Simulation Conference, Washington, DC.
- [11] Van Hook, D. J. and Calvin, J. O. 1998. "Data distribution management in RTI 1.3.", In Proceedings of the 1998 Spring Simulation Interoperability Workshop, paper no. 98S-SIW-206, March 9-13, Orlando, FL.
- [12] Guha, R. K. and Bassiouni M. 2002. "Simulation methods and applications", *Simulation Practice and Theory*, Vol. 9, No. 3-5, 91-93.
- [13] Pardo-Castellote, G. 2003. "OMG data distribution service: architectural overview", *IEEE Military Communications Conference, MILCOM*, Vol. 1, 242-247.
- [14] Bourkerche, A. and Roy, A. 2002. "Dynamic grid-based approach to data distribution management", *Journal of Parallel and Distributed Computing*, Vol. 62, 366-392.
- [15] Tan, G., Ayani, R., Zhang, Y. S., and Moradi, F. 2000. "Grid-based data management in distributed simulation", In Proceedings of the 33rd Annual Simulation Symposium, 16-20 April, Washington, DC, pp. 7-13.
- [16] Tan, G., Liang, X., Moradi, F., and Taylor, S. 2001. "An agent-based DDM for high level architecture", In Proceedings of the 15th Workshop on Parallel and Distributed Simulation, 15-18 May, Lake Arrowhead, CA, pp. 75-82.
- [17] Tan, G., Zhang, Y., and Ayani, R. 2000. "A hybrid approach to data distribution management", In Proceedings of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications, 25-17 August 2000, San Francisco, CA, pp. 55-61.
- [18] Raczy, C., Tran, G., and Yu, J. 2005. "A sort-based DDM matching algorithm for HLA", *ACM Transactions on Modeling and Computer Simulation*, Vol. 15, No. 1, 14-38.
- [19] Boukerche, A. and Dzermajko, C. 2004. "Performance evaluation of data distribution management strategies", *Concurrency and Computation: Practice and Experience*, Vol. 16, No. 15, 1545-1573.
- [20] Morse, K. L. and Zyda, M. 2002. "Multicast grouping for data distribution management", *Simulation Practice and Theory*, Vol. 9, No. 3-5, 121-141.
- [21] Morse, K. L. 2000. "An adaptive, distributed algorithm for interest management", Ph.D. Dissertation, Information & Computer Science Department, University of California, Irvine, CA.
- [22] Petty, M. D. 1997. "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, FL.
- [23] FDK—Federated simulations development kit. <http://www.cc.gatech.edu/computing/pads/fdk.html>
- [24] Fujimoto, R., McLean, T., Perumalla, K., and Tadic, I. 2000. "Design of high-performance RTI software", In Proceedings of Distributed Simulations and Real-time Applications (DS-RT), San Francisco, CA.
- [25] Gupta, P. and Guha, R. K. 2006. "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, Spring Simulation Conference, April 2-6, 2006, Huntsville, AL.
- [26] Koh, J. B., Lee, B. S., Cai, W. T., and Turner, S. J. 2000. "Multicasting fast messages in RTI-Kit", In Proceedings of the 2000 Spring Simulation Interoperability Workshop, paper no. 00S-SIW-039, March 26-31, Orlando, FL.
- [27] Riley, G., Ammar, M., Fujimoto, R., Park, A., Perumalla, K., and Xu, D. 2004. "A federated approach to distributed network simulation", *ACM Transactions on Modeling and Computer Simulation*, Vol. 14, No. 2, pp. 116-148.
- [28] Tadic, I. and Fujimoto, R. M. 1998. "Synchronized data distribution management in distributed simulations", In Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation, 26-29 May, 1998, Banff, Alberta, Canada, pp. 108-115.
- [29] Wang, J. Z. and Guha, R. K. 2001. "A novel data caching scheme for multimedia servers", *Simulation Practice and Theory*, Vol. 9, No. 3-5, 193-213.
- [30] Ayani, R., Moradi, F., and Tan, G. 2000. "Optimizing cell-size in grid-based DDM", In Proceedings of Fourteenth Workshop on Parallel and Distributed Simulation, 28-31 May, 2000, Bologna, Italy, pp. 93-100.

9. Author Biographies

Pankaj Gupta is received the Ph.D. degree in computer science from the University of Central Florida (UCF), Orlando in 2007. His research interests include distributed simulations, parallel computing, graph theory and discrete optimization. He was recipient of the 2001 Graduate Merit Scholarship from the Office of Graduate Studies, UCF.

Ratan K. Guha received the Ph.D. degree in computer science from the University of Texas at Austin in 1970. He is currently a professor of computer science at the University of Central Florida, Orlando. His research interests include

distributed systems, parallel and distributed simulation, computer networks, security protocols, modeling and simulation, and computer graphics. He has authored more than 100 papers published in various computer journals, book chapters, and conference proceedings. His research has been supported by grants from the US Army Research Office, US National Science Foundation, STRICOM, PM-TRADE, and the State of Florida. He has served as a member of the program committee for several conferences, as the general chair of CSMA 1998 and CSMA 2000, and as the guest coeditor of a special issue of the Journal of Simulation Practice and Theory. He is a member of the ACM, IEEE, IEEE Computer Society, and SCS and has served as a member of the Board of Directors of SCS.