An Evaluation of Partition Granularity in Exascale Parallel Simulations

Elvis S. Liu
IBM Research, Ireland and University College Dublin
Email: elvisliu@ie.ibm.com

***Abstract*—This paper is concerned with simulation technologies for the next generation high-performance computing systems that will operate at ExaFLOP (exascale). As the scale of computer simulations grows in terms of participants and simulated entities, using data filtering schemes to reduce the amount of data communication becomes increasingly important for exascale simulation systems. This paper presents a parallel data filtering algorithm, which divides the workload of filtering process across multiple processors. It also presents an evaluation on the optimal partition granularity of the parallel algorithm, which ensures an optimal use of computational resources in the simulation system.**

***Index Terms*—Exascale Simulation Systems, Interest Management, Data Distribution Management**

I. Introduction

In the coming years, we expect to reach a computational power equivalent to a thousandfold that of the current most powerful supercomputer. The next generation high-performance computing systems will achieve a computational power equivalent to ExaFLOPS (10^{18} floating point operations per second). Computational advances have opened the way for a growing number of computer simulation applications across many fields. However, exascale simulations also generate a substantial amount of data communication within the simulation systems.

The simplest data distribution approach for parallel or distributed simulations would be to have each host broadcast the data of each simulated entity (e.g., position of a vehicle) that it maintains. This might include, however, data that are not of interest to some receiving hosts. As the scale grows, providing scalable data distribution through filtering (referred to as "interest management") becomes one of the major design requirements of exascale simulation systems. The basic idea of interest management is simple: all participants of the simulation should only receive data that are of interest to them. This usually involves a process called "interest matching", which matches the "interest" between data senders and receivers. This process, however, may introduce considerable computational overhead. If the cost of interest management is too high, it would degrade the overall performance of the simulation. Over the years, numerous interest management schemes have been proposed, which sought to reduce the computational overhead and, at the same time, to maintain the high precision of data filtering. These schemes, however, are designed for serial processing which is supposed to be run on a single processor. As exascale simulations are executed on parallel or distributed systems, deploying the existing schemes on these systems would be unsuitable, and the performance cannot be guaranteed.

In our previous work [1], we presented the preliminary design of a parallel algorithm for interest management, which is suitable to deploy on multiprocessors. It facilitates workload sharing by dividing the simulated virtual worlds into a number of partitions (referred to as "zones") and distributing the interest matching process among multiple processors. In this paper, we present the theoretical background of this algorithm. We also present a performance evaluation of the algorithm, which focuses on finding the optimal granularity for the partitions.

The remainder of this paper is organised as follows. Section II briefly reviews the background and related work of zone-based interest management schemes and partition granularity. Section III presents the details of our parallel interest matching algorithm. Section IV evaluates the optimal partition granularity for our approach by experimental results. Finally, Section V concludes this paper and briefly describes our future work.

II. Background and Related Work

This section briefly reviews the related work of zone-based interest management and granularity of zones.

A. Zone-based Filtering

Zone-based filtering schemes (various other terms have been used in the literature to describe this generic approach, most noticeably "cell-based", "grid-based", and "region-based") are perhaps the most widely used approaches for interest management. It has been studied extensively in many fields such as military simulations, commercial games, and academic simulation systems. Numerous schemes have been proposed throughout the years, which usually limit the participants' interactions and communications within a small number of space subdivisions, or zones. They typically partition the simulated virtual world into a number of zones with each zone containing a subset of entities. Participants in the simulation are connected to these zones in order to receive events and updates that are generated from them.

The seamless zone-based approach enables participants to specify an area of interest (AOI), in order to subscribe to multiple partitions. A typical AOI consists of a radius of zones where the participant is joining new zones at the leading edge and leaving old zones at the trailing edge as their avatar moves around the virtual world.

The primary advantage of using seamless zones is that they provide a "seamless" view of the virtual world. In other words, this approach has a better migration transparency - the participant would not see a "loading screen" when they join a new zone. Interest matching, however, is required for this approach. Whenever the AOI moves, the system needs to determine which zone(s) it overlaps. If the number of zones is constant, the computational complexity of the matching process would be $O(m)$ where m is the number of AOIs.

Zone-based schemes that adopt uniform partitioning [2], [3], [4] divide the virtual world into zones that are static, regular, have a uniform orientation, and have uniform adjacency. The most common shapes adopted by the existing approaches are rectangles, hexagons, and triangles.

The majority of nonuniform partitioning schemes employ hierarchical data structures such as binary space partitioning (BSP) trees [5], k-dimensional (k-d) trees [6], [7], and quadtrees [8], [9] for space partitioning. Unlike uniform partitioning, individual zones can be chosen freely and may be modified dynamically during runtime based on whatever is most convenient from the perspective of designing the individual zones themselves.

Furthermore, some systems that are compliant to High-Level Architecture (HLA) [10] also adopt this type of filtering schemes [11].

B. Granularity

Choosing a proper granularity is one of the major considerations for all zone-based schemes. For a static partitioning of the virtual world, a significant trade-off must be made. If the zones are large, each zone would contain a large number of virtual entities and thus the participants might receive a large amount of irrelevant data. On the other hand, if the zones are small, the number of zones as well as the number of multicast groups would become large, and therefore the entity movement between zones would be more frequent. This increases the chance of subscribing to and unsubscribing from multicast groups as the participants move around the virtual world, resulting in an increase in management overheads.

The hierarchy structures described in the previous subsection also suffer from the same problem but in a different form - a trade off must be made when choosing a proper granularity of the leaf nodes or a proper height of the hierarchy. Researchers such as Van Hook et al. [8] and Steed and Abou-Haidar [6] tried to maintain a balanced hierarchy by setting a maximum height or a population threshold. These are practical solutions, however, one should also consider the characteristic of the application, and the processing power and communication speed of the entire simulation system when choosing the optimal granularity.

A study presented in [11] argued that in a system with fast CPUs and slow communication network, the optimal zone size would be rather small. On the other hand, in a simulation system with slower CPUs and faster communication the optimal zone size would be rather large. Moreover, according to Rak and Van Hook [12], setting the zone size between 2 to 2.5km provides the optimal results in terms of filtering precision and multicast group join rates. These results are, however, entirely dependent on the simulation settings.

In this paper, we perform experimental evaluations on the optimal partition granularity of our parallel simulation system. The finding of the experiments is important as it allows the system to achieve and maintain an optimal use of resources.

III. Parallel Interest Matching Algorithm

This section describes a parallel interest matching algorithm which facilitates parallelism by distributing the workload of the matching process across multiprocessors. The algorithm divides the matching process into two phases. In the first phase it employs a spatial data structure called uniform subdivision to efficiently decompose the virtual space into a number of subdivisions. We define as work unit (WU) the interest matching process within a space subdivision. In the second phase, WUs are distributed across different processors and are processed concurrently.

For the sake of consistency, aura is hereafter referred to as "regions" as per the terminology of HLA.

A. Spatial Decomposition

Uniform subdivision is a common spatial data structure, which has long been used as a mean of rapid retrieval of geometric information. The idea of using hashing for subdivision directory was first described in an early article [13] and was later discussed more generally in [14]. This section presents the formal definitions of uniform subdivision, which leads to the discussion in the subsequent sections where they are used for hash indexing and rapid WU distribution.

Formally, the virtual space $\boldsymbol{S}$ can be defined as a multi-dimensional point set that contains all entities in the virtual world. Therefore, all update or subscription regions can be regarded as the subsets of $\boldsymbol{S}$.

Definition 1. Let $[SMIN_d, SMAX_d)$ be the boundary of a space $\boldsymbol{S}$ in d dimension, for $d = 1, 2, ..., n$.

$$\begin{aligned}\boldsymbol{S} =& \{(x_1, x_2, ..., x_n) \mid x_d \in \mathbb{R} \wedge SMIN_d \leq x_d < SMAX_d,\\ & \text{for } d = 1, 2, ..., n\}.\end{aligned}$$

Alternatively, $\boldsymbol{S}$ can be expressed as the Cartesian product of its one-dimensional boundaries.

Definition 2. Let $[SMIN_d, SMAX_d)$ be the boundary of a space $\boldsymbol{S}$ in d dimension, for $d = 1, 2, ..., n$.

$$\begin{aligned}\boldsymbol{S} =& [SMIN_1, SMAX_1) \times [SMIN_2, SMAX_2) \times ...\\ & \times [SMIN_n, SMAX_n)\\ =& \prod_{d=1}^{n} [SMIN_d, SMAX_d).\end{aligned}$$

The hashing approach requires decomposing $\boldsymbol{S}$ into uniform subdivisions. Each subdivision represents a slot in the hash table, which is labelled by a multidimensional hash table index.

Definition 3. Let $[SMIN_d, SMAX_d)$ be the boundary of a space $\boldsymbol{S}$ in d dimension. The boundary can be uniformly divided into N_d sub-boundaries with unit length L_d, such that

$$L_d = \frac{SMAX_d - SMIN_d}{N_d}$$

$\forall N_d \in \mathbb{Z}^+$, $\forall L_d \in \mathbb{R}^+$, for $d = 1, 2, ..., n$.

Definition 4. Let $[SMIN_d, SMAX_d)$ be the boundary of a space in d dimension, for $d = 1, 2, ..., n$. The boundary is uniformly divided into N_d sub-boundaries with unit length L_d. The uniform subdivision $\boldsymbol{Z}$ of $\boldsymbol{S}$ is labelled by a multidimensional hash table index $(z_1, z_2, ..., z_n)$, such that

$$\begin{aligned}&\boldsymbol{Z}(z_1, z_2, ..., z_n)\\ &=\{(x_1, x_2, ..., x_n) \mid x_d \in \mathbb{R} \wedge SMIN_d + z_d L_d \le x_d\\ &\quad < SMIN_d + (z_d + 1)L_d,\ \text{for } d = 1, 2, ..., n\}\end{aligned}$$

for $z_d = 0, 1, ..., N_d - 1$.

Similar to all axis-aligned point sets, the uniform subdivision can be expressed as the Cartesian product of its one-dimensional boundaries, which is given in **Definition 5**.

Definition 5. Let $[SMIN_d, SMAX_d)$ be the boundary of a space in d dimension, for $d = 1, 2, ..., n$. The boundary is uniformly divided into N_d sub-boundaries with unit length L_d. The uniform subdivision $\boldsymbol{Z}$ of $\boldsymbol{S}$ can be defined as

$$\begin{aligned}&\boldsymbol{Z}(z_1, z_2, ..., z_n)\\ &=[SMIN_1 + z_1 L_1, SMIN_1 + (z_1 + 1)L_1)\\ &\quad \times [SMIN_2 + z_2 L_2, SMIN_2 + (z_2 + 1)L_2)\\ &\quad \times ...\\ &\quad \times [SMIN_n + z_n L_n, SMIN_n + (z_n + 1)L_n)\\ &=\prod_{d=1}^{n}[SMIN_d + z_d L_d, SMIN_d + (z_d + 1)L_d)\end{aligned}$$

for $z_d = 0, 1, ..., N_d - 1$.

Theorem 1. Given a set of all hash table indices

$$\boldsymbol{HI} = \{(z_1, z_2, ..., z_n) \mid z_d = 0, 1, ..., N_d - 1 \wedge d = 1, 2, ..., n\}$$

where N_d is the number of subdivisions of space $\boldsymbol{S}$ in d dimension. Then, $\boldsymbol{S}$ can be expressed as the union of all uniform subdivisions, such that

$$\boldsymbol{S} = \bigcup_{k \in \boldsymbol{HI}} \boldsymbol{Z}(k).$$

Proof: By **Definition 2**, we derive

$$\begin{aligned}\boldsymbol{S} =&[SMIN_1, SMAX_1) \times [SMIN_2, SMAX_2) \times ...\\ &\times [SMIN_n, SMAX_n)\\ =&\bigcup_{z_1=0}^{N_1}[SMIN_1 + z_1 L_1, SMIN_1 + (z_1 + 1)L_1)\\ &\times \bigcup_{z_2=0}^{N_2}[SMIN_2 + z_2 L_2, SMIN_2 + (z_2 + 1)L_2)\\ &\times ...\\ &\times \bigcup_{z_n=0}^{N_n}[SMIN_n + z_n L_n, SMIN_n + (z_n + 1)L_n)\\ =&\bigcup_{k \in \boldsymbol{HI}} \boldsymbol{Z}(k).\end{aligned}$$

■

B. First Phase: Hashing

During the simulation, regions are hashed into the hash table. The algorithm uses the coordinate of a region's vertex as a hash key. Given a key k, a hash value $H(k)$ is computed, where $H()$ is the hash function. The hash value is an n-dimensional index, which can be matched with the index of a space subdivision, and therefore indicating that which subdivision the vertex lies in. Hence, the regions with hash key k are stored in slot $H(k)$. The hash function is given in **Definition 6**.

Definition 6. Let $[SMIN_d, SMAX_d)$ be the boundary of a space in d dimension, for $d = 1, 2, ..., n$. The boundary is uniformly divided into N_d sub-boundaries with unit length L_d. The hash function for transforming a key k_d into a hash value is defined as

$$H : \mathbb{R}^n \to \mathbb{Z}^n, H(k_d) = \lfloor \frac{k_d - SMIN_d}{L_d} \rfloor$$

There are two important properties of using a hash table for spatial decomposition. First, hash table collision means that regions in the same slot are potentially overlapped with each other; therefore, further investigation on their overlap status is required. This process is left to the second phase of the algorithm. Second, if a region lies in multiple space subdivisions, it would be hashed into all of them. The algorithm assumes that the size of region is much smaller than a space subdivision. Therefore, a region would exist in at most four slots in the two-dimensional space (at most eight slots in the three-dimensional space). This assumption ensures that the computational complexity of the hashing process would be bounded by a constant.

Figure 1 illustrates the basic concept of the spatial hashing for two-dimensional space. In the figure, region A is hashed into slot (0,1); region B is hashed into slots (0,0), (0,1), (1,0) and (1,1); region C is hashed into slots (1,1) and (1,2); region D is hashed into (1,0), (1,1), (2,0) and (2,1).

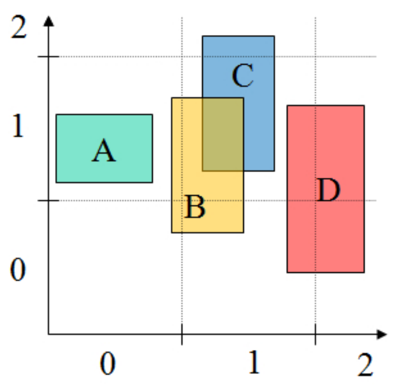

Fig. 1. Hashing for Space Subdivisions

The basic steps to construct a hash table are given in Algorithm 1. Note that if not all vertices of a region are hashed into the same slot, then the region exists in multiple subdivisions.

Algorithm 1: Algorithm for Hash Table Construction (Region)

```
Data: $\boldsymbol{S}$: a $n$-dimensional virtual space
Data: $\boldsymbol{Z}$: an uniform subdivision of $\boldsymbol{S}$
Data: $k$: a $n$-dimensional hash table index
Data: $\boldsymbol{R}$: a region
Data: $v$: a vertex of $\boldsymbol{R}$
Data: $H()$: a hash function
Data: $HashTable$: a hash table
1  begin
2      Decompose $\boldsymbol{S}$ into a list of $\boldsymbol{Z}$;
3      foreach $\boldsymbol{Z}$ do
4          Determine the index $k$ for $\boldsymbol{Z}$;
5      end
6      foreach $k$ do
7          $HashTable.AddSlot(k)$;
8      end
9      foreach $\boldsymbol{R}$ do
10         foreach $v$ of $\boldsymbol{R}$ do
11             $HashTable.Slot[H(v)].AddRegion(\boldsymbol{R})$;
12         end
13     end
14 end
```

The hash table is constructed at the initialisation stage. During runtime, the position and size of regions may be frequently modified. Therefore, the algorithm needs to perform rehashing for the regions at every time-step. The complexity of this process is $O(n + m)$ where m is the number of subscription regions and n is the number of update regions.

C. Second Phase: Sorting

After the hashing stage, each slot of the hash table represents a WU which will be distributed across different processors. The algorithm then places the WUs on a task queue. Each processor fetches WUs from the queue and performs interest matching for the corresponding space subdivisions. Since only one processor has the authority to manage each space subdivision, there will be no ambiguous matching result. As discussed in [15], the task queue approach is desired for task distribution and provides very good load sharing for multiprocessors. When a processor finishes processing a WU, it would fetch another WU from the task queue immediately unless the queue is empty. Therefore, no processor would be idle until all WUs are fetched. The worst case happens only when all regions reside in a single space subdivision. In this situation, a single processor would be responsible for the matching of all of them.

The spatial decomposition approach essentially transforms the large-scale interest matching process into several individual sub-problems. When a WU is being processed, each processor carries out a matching process only for the regions within the WU. The matching process employs a sorting algorithm [1], which makes use of the concept of dimension reduction and is theoretically the most efficient serial algorithm for interest matching.

IV. Experimental Evaluation

This section presents the experimental evaluation on the optimal partition granularity of the parallel algorithm. Two sets of experiments were carried out to compare the performance of two approaches, namely:

1) Discrete interest matching by parallel algorithm (PDIM)
2) Space-time interest matching by parallel algorithm (PCIM)

PDIM [1] and PCIM [16] are the two parallel algorithms, which exploit parallelism by distributing the workload across multiple processors. The PDIM approach is designed for discrete interest matching approach, which performs interest matching at discrete time intervals, while the PCIM approach performs space-time interest matching approach, which perform space-time interest matching in order to capture more events in the simulation. The PCIM approach usually requires more computational effort than the PDIM approach.

A. Implementation and Experimental Set-ups

The two algorithms were implemented in C++. Message communication was constructed based on Open MPI protocols, such as $MPI\_Bcast()$, $MPI\_Send()$, and $MPI\_Recv()$; all processes were synchronised by the $MPI\_Barrier()$ call, which is a simple lock-step synchronisation protocol. The experiments were executed on the eScience Cluster at the Midland e-Science Centre. Each worker node has an Intel Xeon 3GHz 4-core processor with 2GB main memory. A Myrinet backplane is used to give 2+2Gbps programmable interconnection between the worker nodes.

The following set-up was used for the experiments.

- *Entity Distribution*: The entities are distributed randomly across the virtual space.
- *Entity Movement*: All entities move in a random direction and undergo linear translational motion.

- *Entity Speed*: The speed factor (SF) represents the average speed of the entities in proportion to its region length.
- *Number of Dimensions*: All simulations were performed in three-dimensional space.
- *Number of Regions* An update region and a subscription region were associated with each moving entity.
- *Execution Time Measurement*: Average execution time of the matching algorithms was measured over 10,000 time-steps.
- *Number of WUs*: The number of WUs is dependent on the granularity of spatial decomposition, which was assigned statically. The optimal granularity value was determined through experiments, which are presented in Section IV-B.
- *Number of Nodes*: All experiments of the parallel algorithms were run on 10 nodes.

B. Granularity of Spatial Decomposition

The spatial decomposition approach described in Section III-A requires an optimal granularity to achieve an optimal use of resources. This is similar to the virtual world partitioning problem as we have discussed in Section II-B. In this section, we present the results of a set of experiments that we have conducted to determine the optimal granularity of partitioning.

Since uniform subdivisions are employed for the parallel algorithm, the granularity of spatial decomposition is dependent on the number of sub-boundaries per dimension (denoted by N_d in **Definition 3**). We take $N_1 = N_2 = N_3$, which implies that each subdivision is a cube in shape. We measured the execution time of PDIM and PCIM, with N_d extending from 2 to 10. The number of entities was set to 20000 and the SF was set to 20.

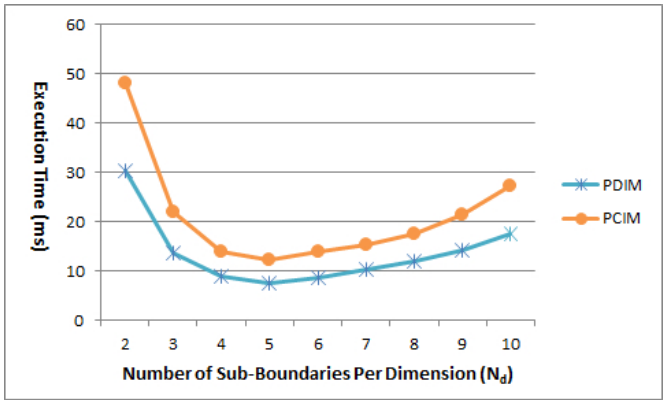

Fig. 2. Comparing the Execution Time of Parallel Interest Matching Approaches (Number of Sub-Boundaries varies)

The results are given in Figure 2. As we can see in the graph, there is some significant runtime overhead when N_d is equal to 2 (i.e., the number of WUs is equal to 8). This is due to the fact that the number of WUs is less than the number of nodes, which implies four physical cores would be idle at each time step of simulation, resulting in a poor utilisation of computational resources. However, if a large N_d is chosen, the size of job queue becomes large and thus overhead would be introduced due to the increase in the frequency of job queue access. According to results shown in Figure 2, we can conclude that, for the current experimental set-up, the optimal value of N_d for both PDIM and PCIM is 5.

V. Conclusions and Future Work

The interest management schemes enhance the scalability of the exascale simulation systems by filtering irrelevant data communication on the network. Over the years, many efficient filtering algorithms were proposed to speed up the interest matching process. However, they were designed for serial processing which is supposed to be run on a single processor. As the problem size grows, using these algorithms does not satisfy the scalability requirement of exascale simulations since the single processor may eventually become a bottleneck. In our previous work [1], we have presented the preliminary design of a parallel interest matching algorithm which is suitable to deploy on a multiprocessor computer. This algorithm partitions the simulated virtual world and distributes the interest management process across multiple processors. In this paper, we presented the detailed theoretical background of the parallel algorithm. We also presented an experimental evaluation on the optimal partition granularity of the proposed algorithm. Using the optimal granularity in future experiments would be the most efficient way to achieve and maintain an optimal use of resources in the simulation system.

Our future work will concentrate on evaluating the runtime efficiency of the parallel algorithm. We will test and compare its performance under different entity behaviors, number of nodes, and occupation density.

VI. Acknowledgment

This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

References

[1] E. S. Liu and G. K. Theodoropoulos, “An Approach for Parallel Interest Matching in Distributed Virtual Environments,” in *Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2009)*, October 2009.

[2] D. J. Van Hook, S. J. Rak, and J. O. Calvin, “Approaches to Relevance Filtering,” in *Proceedings of the 11th Workshop on Standards for the Interoperability of Distributed Simulations*, 1994, pp. 26–30.

[3] S. Srinivasan and B. R. D. Supinski, “Multicasting in DIS: A Unified Solution,” in *Proceedings of the 1995 Electronic Conference on Scalability in Training Simulation (ELECSIM 1995)*, April-June 1995.

[4] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham, “Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments,” in *Proceedings of the 1995 IEEE Virtual Reality Annual International Symposium (VRAIS ’95)*, 1995, pp. 2–10.

[5] J. W. Barrus, R. C. Waters, and D. B. Anderson, “Locales and Beacons: Efficient and Precise Support For Large Multi-User Virtual Environments,” in *Proceedings of IEEE 1996 Annual International Symposium on Virtual Reality (VRAIS ’96)*, 1996, pp. 204–213.

[6] A. Steed and R. Abou-Haidar, “Partitioning Crowded Virtual Environments,” in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST) 2003*, 2003, pp. 7–14.

[7] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, L. Petrak, and G. Carle, “Peer-to-Peer-based Infrastructure Support for Massively Multiplayer Online Games,” in *Proceedings of the 4th Annual IEEE Consumer Communications and Networking Conference (CCNC 2007)*, January 2007, pp. 763–767.

[8] D. J. Van Hook, S. J. Rak, and J. Calvin, “Approaches to RTI Implementation of HLA Data Distribution Management Services,” in *Proceedings of the 15th Workshop on Standards for the Interoperability of Distributed Simulations*, 1997.

[9] K. Pan, X. Tang, W. Cai, S. Zhou, and H. Zheng, “Hierarchical interest management for distributed virtual environments,” in *Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation*, ser. SIGSIM-PADS ’13, 2013, pp. 137–146.

[10] DMSO, “High Level Architecture Interface Specification Version 1.3,” 1998.

[11] G. Tan, R. Ayani, Y. Zhang, and F. Moradi, “Grid-Based Data Management in Distributed Simulation,” in *Proceedings of the 33rd Annual Simulation Symposium (SS 2000)*. Washington, DC , USA: IEEE Computer Society, April 2000.

[12] S. J. Rak and D. J. Van Hook, “Evaluation of Grid-Based Relevance Filtering for Multicast Group Assignment,” in *Proceedings of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, March 1996, pp. 739–747.

[13] M. Rabin, “Probabilistic algorithms,” in *Algorithms and Complexity: New Direction and Recent Results*. New York: Academic Press Inc, 1976, pp. 21–39.

[14] J. L. Bentley and J. H. Friedman, “Data Structures for Range Searching,” *ACM Computing Survey*, vol. 11, pp. 397–409, December 1979.

[15] S. P. Dandamudi and P. S. P. Cheng, “A Hierarchical Task Queue Organization for Shared-Memory Multiprocessor Systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 1, pp. 1–16, 1995.

[16] E. S. Liu and G. Theodoropoulos, “A Continuous Matching Algorithm for Interest Management in Distributed Virtual Environments,” in *Proceedings of the 24th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2010)*, May 2010.