Dynamically Adaptive Partition-based Data Distribution Management*

Bora İ. Kumova

İzmir Institute of Technology; Department of Computer Engineering; 35430 Turkey
borakumova@iyte.edu.tr; iyte.edu.tr /~borakumova

Abstract

Performance and scalability of distributed simulations depends primarily on the effectiveness of the employed data distribution management (DDM) algorithm, which aims at reducing the overall computational and messaging effort on the shared data to a necessary minimum. Existing DDM approaches, which are variations and combinations of two basic techniques, namely region-based and grid-based techniques, perform purely in the presence of load differences. We introduce the partition-based technique that allows for variable-size partitioning shared data. Based on this technique, a novel DDM algorithm is introduced that is dynamically adaptive to cluster formations in the shared data as well as in the physical location of the simulation objects. Since the re-distribution is sensitive to inter-relationships between shared data and simulation objects, a balanced constellation has the additional advantage to be of minimal messaging effort. Furthermore, dynamic system scalability is facilitated, as bottlenecks are avoided.

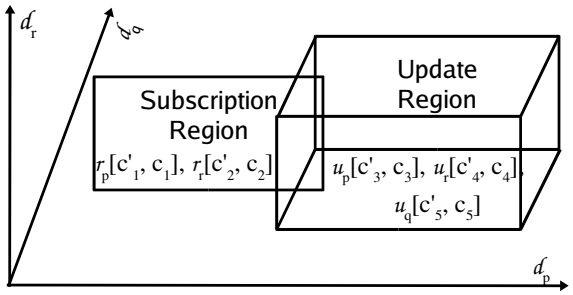
1 Introduction

Establishing and maintaining network connections for the purpose of satisfying the mutual information requirements of the participants of a distributed system is called data distribution management (DDM) [14]. Alternative denominations are relevance filtering [8], data subscription management, focus [1] data dissemination management [7] or interest management (IM) [16]. In this text we shall use these terms interchangeably. The principle goal of DDM is to deliver objects of a given distributed system the exact amount of data they are interested in, no more and no less.

The two basic filtering techniques, region-based and grid-based, have been discussed in the literature [10], [14], [2], combined with other techniques and refined extensively, in order to improve performance. In region-based filtering, simulation objects specify interest areas on the shared data, in form of update and subscription regions (*Figure 1*). Where intersecting regions represent potential communication between related objects. This technique is inherently parallelisable upon the shared data, if the data is represented as a homogeneous multi-dimensional hyperspace, like in High Level Architecture (HLA) DDM 1516 [17]. The upper bound time complexity is $O(n^2)$, for comparing in the worst case every subscription region with every update region, and O(n log n) in case of recursive interest matching algorithms [17]. In grid-based filtering, shared data is subdivided into equal-sized value ranges and interest matching is performed implicitly, when interest regions fall into the same grid. Parallelism is exploited here by performing independently within each grid.

Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) operating system [19] and Synthetic Theater of War Europe (STOW-E) [9] were early implementations of the grid approach. Where the latter combined both techniques for filtering the data difference between an intersecting region and the grid region, which seeks reducing the network transmission of unnecessary data. Some HLA DDM implementations have also adopted this approach, like U.S. Defense Modeling and Simulation Office (DMSO) Runtime Infrastructure (RTI) [5] and Pitch RTI (pRTI) [18].

*This research was partially supported by Ege University, International Computer Institute.



d_p: Data d Dimension p

r_p, $u_p[c'_1, c_1]$: Read r, Update u Interval c'_1 through c_1 on d_p

Figure 1. Sample Update/Subscription Regions in the Multi-dimensional Hyperspace as of HLA DDM 1516

While the above techniques aim at reducing the messaging load on the network, multicasting can reduce the computational load on a node, by delegating it to the related hardware unit of the network. Assigning each intersection region or grid a multicast group [15], [3] can reduce the computational load for replicating a data update propagation.

Sources for computational imbalances in distributed simulation are clusters of federates and/or clusters of data access regions. Where cluster formations are unpredictable, particularly in non-deterministic application domains. For instance, the load balancing approach proposed for HLA implementations in [4], interprets an overloaded node as a cluster of federates and resolves imbalances by migrating whole federates. Data access clusters are identified based on simulation objects' position and extents of interest on the shared data, such as focus [1], visibility [13], spheres of influence [12]. Connected intersections of such interest regions then imply data access clusters.

Dynamically adaptive partition-based DDM is a new approach for load balanced distributed simulation that assembles and extends the above ideas. This model is introduced here in the context of the distributed simulation platform Adaptive Parallel Discrete Event Simulator (APDES) [11] for non-deterministic applications. It possesses the following properties:

- Identifying multi-dimensional clusters within the multi-dimensional state variable (SV) space.
- Adapting the boundaries of a multi-dimensional SV partition to encapsulated clusters.
- Migrating a SV partition to an imbalanced neighbouring host and splitting/merging the partitions.
- Migrating an LP to its interest region with the highest access cost for this LP.

The principle structure of the model is introduced below. We shall use the following terms interchangeably, shared data, data dimension and state variable (SV) and refer to as d_i, and the terms federate, logical process (LP) and simulation object, although sometimes an LP may represent several simulation objects.

2 Principle Structure of the Model

The principle idea of the new DDM model is to distribute the shared data of an application over a given set of hosts, by assigning each host one partition of each data dimension and by preserving adjacency information between partitions. Since one partition has at most two adjacent partitions, changing interest region boundaries need to be communicated to at most one host, as we assume continues value changes on SVs. Partition sizes are dynamically calculated based on emerging data clusters and distributed, depending on the contained regions' access costs.

2.1 State Variable Conceptualisation

SVs are normalised within the value range [0..1], as this facilitates homogeneous and variable-sized partitioning over dynamically varying numbers of hosts. Values of a SV are assumed to change continuously to the maximum extent of the local data partition size. An LP^e may declare at most one update region $u^e_i[c'_1, c_1]$ and at most one subscription region $r^e_i[c'_2, c_2]$ per SV d_i, where $c'_1 \le c_1$, $c'_2 \le c_2$ and $c'_1, c_1, c'_2, c_2 = [0..1]$.

2.2 Data Access Evaluation

Statistics over the access to SV regions provide qualitative information about regions and enable the DDM to dynamically adapt to the application behaviour, thus facilitating load balancing and scalability. Therefore every access region is attributed with following statistical values, which are additionally stored at the owning LP's site.

Region access frequency. In the course of a simulation the access frequency of SVs may change dynamically, depending on changing interests and

interactions between LPs. The access frequency $\mathcal{F}(a^e_i[c'_1, c_1])$ of an update or subscription region $a^e_i[c'_1, c_1]$ of LP^e on SV i, gives information about how often it was accessed in the most resent time interval [t', t_0], where $t' = t_0 - \Delta f$, t_0 is the last global state change and Δf the number of some most resent global state changes. Δf may be adapted to the overall region access frequency of a particular application. This information is used to optimise the distribution of the shared data and the LPs.

Region access latency. The average time to access a region is updated with every access to that region with

$$\mathcal{L}(a_i) = (\mathcal{L}'(a_i) \ \mathcal{F}(a_i) + |a_i|_{time}) \ / \ (\mathcal{F}(a_i) + 1)$$

a_i is an abbreviation for $a_i[c'_1, c_1]$, $\mathcal{L}'$ is the previous access frequency and $|a_i|_{time}$ the time required for the most resend access. The access latency of local regions is always set to zero. In case of remote regions, the time stamp of the incoming message is compared with local time.

2.3 Data Cluster Detection

A cluster C_i is identified as an ordering relation of access region boundaries on SV d_i:

$$C_i[c_{z'1}, c_{z1}] = \{(a_i[c_{z'2}, c_{z2}], b_i[c_{z'3}, c_{z3}]) \mid c_{z'1} \le c_{z'2} \wedge c_{z'2} \le c_{z2} \wedge c_{z'3} \le\le c_{z3} \wedge c_{z'3} \le c_{z2} \wedge c_{z3} \le c_{z1} \wedge c_{z'1}, c_{z1}, c_{z'2}, c_{z2}, c_{z'3}, c_{z3} = [0..1] \wedge i = [1..s]\}$$

For any access region a_i, b_i. Where two clusters $C_i[c_{z'1}, c_{z1}]$ and $C_i[c_{y'1}, c_{y1}]$ on the same SV d_i do not intersect, iff $c_{z1} < c_{y'1}$. The number of regions in a cluster $|C_i[c_{z'1}, c_{z1}]|_{size}$ is stored separately. The smallest possible cluster is a single region without intersections $|C_i[c_{z'1}, c_{z1}]|_{size} = 1$. Cluster detection is processed simultaneously with interest matching, as both share a similar algorithmic structure.

2.4 Interest Matching

Matching interest is calculated by intersecting two boundaries of two access regions $a_i[c'_1, c_1]$ and $b_i[c'_2, c_2]$ on SV d_i, such that

$$a_i[c'_1, c_1] \cap b_i[c'_2, c_2] \neq \emptyset \rightarrow c'_2 \le c_1$$

Only c_1 and c'_2 need to be compared for each region, as the boundary values c'_1, c_1, c'_2, c_2 are sorted. Additionally, the algorithm simultaneously calculates the clusters C_i. For this purpose, region boundaries on each SV are kept sorted in ascending order, which requires an additional computational effort. The time complexity for inserting one region into the sorted list of regions is O(log n).

2.5 Basic Data Partitioning Concepts

Data management starts initially with gridded partitioning the shared data in form of a table and continues with some primitive operations on the table.

Initial partition distribution. A data dimension d is sub-divided into s equal-sized partitions $d_{i,j}$ and each partition assigned to a different host h_j of the distributed system with n hosts (*Table 1*).

Table 1. Initial Assignment of Data Partitions to Hosts

$d_{i,j}$ at h_j	h_1	...	h_n
d_1	$d_{1,1}$	...	$d_{1,n}$
...	...	...	...
d_s	$d_{s,1}$	...	$d_{s,n}$

The table entry at $h(i, j)$ contains the actual network address of the hosts h_j with partition $d_{i,j}$. This schema allows to find the network location of a partition $d_{i,j}$ with a single direct access. Depending on the semantics of a data dimension and the location of LPs, adjacent partitions $d_{i,j}$ and $d_{i,j+1}$ may be stored at physically closer located hosts h_j and h_{j+1} in the network, as region boundaries of access regions can move only to adjacent partitions.

Note however that in the initial configuration, the partition index j in $d_{i,j}$ and the host index j in h_j are the same, hence $h_j = h(i, j)$. Below, an extension to this schema is introduced with variable-size partitioning and partition migration, where the equality $h_j = h(i, j)$ does not hold any more for any j.

Primitive partitioning operations. In the course of a simulation, a partition $d_{i,j}$ may be merged or split with adjacent partitions $d_{i,j-1}$ or $d_{i,j+1}$. This is exemplified below on a disengaging/engaging host, where a column j that consists of partition j on all SVs d_i with i = [0..s], is merged/split with adjacent partitions, respectively.

By default, each partition $d_{s,k}$ of a disengaging host h_k is joined with one of its adjacent partitions, either $d_{i,k-1}$ or $d_{i,k+1}$ (*Table 2*).

Table 2. Merging Data Partitions from a Disengaging Host with Neighbouring Partitions

$0d_{i,j}$ at h_j	. . .	h_{k-1}	h_k	h_{k+1}	. . .
d_1	. . .	$d_{1,k-1}$; $d_{1,k}$ ←	$d_{1,k}$	$d_{1,k+1}$	. . .
. . .	. . .	. . . ←	. . .	. . .	. . .
d_s	. . .	$d_{s,k-1}$; $d_{s,k}$ ←	$d_{s,k}$	$d_{s,k-+1}$	. . .

By default, a partition d_i on host h_{k-1} is split into two, $d_{i,k-1}$ and $d_{i,k}$, and assigned to the new host h_k. (*Table 3*).

Table 3. Splitting Data Partitions and Re-assigning to a New Engaging Host

$d_{i,j}$ at h_j	. . .	h_{k-1}	h_k	h_{k+1}	. . .
d_1	. . .	$d_{1,k-1}$; $d_{1,k}$ →	$d_{1,k}$	$d_{1,k+1}$	. . .
. . .	. . .	. . . →	. . .	. . .	. . .
d_s	. . .	$d_{s,k-1}$; $d_{s,k}$ →	$d_{s,k}$	$d_{s,k-+1}$	. . .

Partition size. The size of data partitions are directly affected by the number of available hosts. Depending on the context of a SV d_i and performance considerations during a particular application, initially equal-sized partitions $d_{i,j}$ may become variable or zero-sized. Variable size partitioning allows adapting to clusters of intersecting access regions. However, a cluster that is split over more than one partition, can cause additional network messaging, as intersecting interests represent potential communication relationships. Zero partition size of $d_{i,j}$ indicates that host h_j does currently not maintain any value range of that SV, which may change during an application for load balancing purposes.

A performance issue is the ratio between partition length and access region length, respectively:

$$m_j = |a^e_i[c'_1, c_1]| \; / \; |d_{i,j}|$$

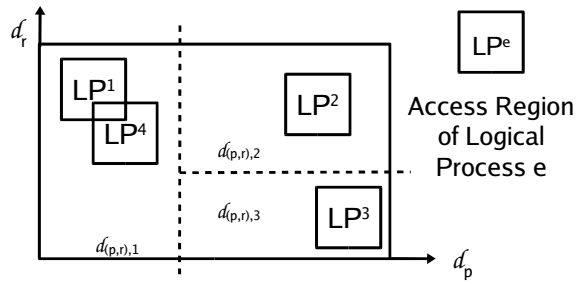
For $m \leq 1$ one message is required. For $1 < m$, m split regions of a^e_i need to be maintained at m hosts, which implies an additional network overhead of m -1 messages and an additional local overhead at m -1 hosts for pointing to the host with the owning LP^e of a^e_i. At the site of LP^e an additional local overhead of m-1 pointers to further m-1 hosts is required.

Access region replication. For the purpose of minimising messaging effort, for each subscribed remote SV region, a local copy is maintained. Where one DDM objective is to minimise the total number of replica within a simulation, as each replica requires one message for synchronising with each update on the original remote data.

2.6 Multi-dimensionality of State Variables and Data Clusters

By default all SVs are homogeneous and no semantics are defined between value ranges of a SV or between different SVs. However supporting such semantics helps identifying multi-dimensional data clusters in the context of user model semantics. Consider for instance the state variables d_p and d_r in (*Figure 2*). If they are evaluated independently, then following LP clusters would result: (LP^1, LP^4) and (LP^2, LP^3) on d_p and (LP^1, LP^2, LP^4) and (LP^3) on d_r. A somehow logically combined evaluation of d_p and d_r however, for instance as physical area, apparently results in different LP clusters (*Figure 2*). In this case, LP^2 does no more appear in the cluster with LP^1 and LP^4, as it is not intersecting with them on the physical area.

Declaring compound SVs. A group of SVs d_1, .., d_q, with $1 \leq q \leq s$, where s is the total number of SVs of



$d_{(p,r),(i,j)}$: Partition (i,j) of SV $d_{(p,r)}$

Figure 2. Sample Relationships Between Access Regions of Logical Processes in a Two-dimensional Data Space

the application can be introduced as one compound SV (d_1, .., d_q) by declaring

$$d_{(1,..,q)} = \{ (d_1, \ldots, d_q) \mid q \leq s \}$$

In general, the more SVs are bound to each other by grouping relationships, the smaller the probability for involving access regions incorrectly in multi-dimensional data clusters. Hence, the smaller the probability for LPs to depend on each other over shared data.

A particular SV d_k may be grouped multiple times. Resulting consistency relationships between groups however is in the responsibility of the user application. For instance whether updates on d_k over one group should have consistent interpretation in an other group that includes d_k as well.

Detecting multi-dimensional data clusters. A multi-dimensional data cluster $C_{(1,...,q)}[c_{z'1}, c_{z1}, ..., c_{z'q}, c_{zq}]$ is detected in two steps: Firstly, the above described cluster detection algorithm is applied to all SVs, independently from whether a SV is bound or not. Secondly, for each compound SV $d_{(1,..,q)}$, related access regions $a^e_{(1,..,q)}$ and $a^f_{(1,..,q)}$, of LP^e and LP^f respectively, are compared pairwise, if they intersect on at least one SV d_1, ..., d_q, such that

$$[c_{z'1}, c_{z1}] \cap [c_{y'1}, c_{y1}] \neq \emptyset \vee \ldots \vee [c_{z'q}, c_{zq}] \cap [c_{y'q}, c_{yq}] \neq \emptyset \rightarrow C^{e,f}_{(1,..,q)}[c_{z'1}, c_{z1} \vee c_{y1}], \ldots, [c_{z'q}, c_{zq} \vee c_{yq}]$$

The time complexity for detecting q-dimensional clusters is $O(q * g) = O(n^2)$, for comparing in the worst case all g LPs' access regions on all q SVs. With a recursive algorithm, the complexity reduces to O(n log n).

Space complexity for variable-size and multi-dimensional partitions. On one hand, less dependent LPs require less messaging effort and can be re-located more independently. On the other hand, bound SVs increase the space requirement at each host. For instance, the number of adjacent multi-dimensional rectangular partitions increases exponentially with $s^3 * 2^s$ for increased number of s dimensions, which is a space complexity of $O(n^3\, 2^n)$. It enables however encapsulating an s-dimensional cluster within an s-dimensional SV partition. of variable edge size. For instance, for a 3-dimensional partition, the addresses of $3 * 2^3 - 3 = 21$ to 24 hosts need to be stored at each host, depending on the location of a partition at SV boundaries. The number increases with the number of adjacent partitions at each edge or surface.

2.7 Optimisation of Partition Distribution

Partition distribution over the hosts is dynamically optimised through variable-size partitioning, for adapting to cluster formations, and partition migration, for balancing the messaging load between hosts.

Variable-size partitioning. A single partition $d_{i,j}$ is split between two clusters $C_i[c_{z'1}, c_{z1}]$ and $C_i[c_{z'2}, c_{z2}]$ at $(c_{z1} + c_{z'2}) / 2$. A multi-dimensional partition $d_{(1,...,q)}$ is split between two clusters $C_{(1,...,q)}[c_{z'1}, c_{z1}, ..., c_{z'q}, c_{zq}]$ and $C_{(1,...,q)}[c_{y'1}, c_{y1}, ..., c_{y'q}, c_{yq}]$ at each SV partition (1, ..,q), such that $(c_{z1} + c_{y'1}) / 2$, ..., $(c_{zq} + c_{y'q}) / 2$. Consequently, variable-size partitions may emerge in the course of a simulation (*Table 4*).

Table 4. Sample Variable-size Data Partitions

$d_{i,j}$ at h_i(i,j)	h_i(i,j)			
d_1	$d_{1,1}$ at h_1	$d_{1,2}$ at h_4	$d_{1,3}$ at h_3	$d_{1,4}$ at h_5
d_2	$d_{1,1}$ at h_3	$d_{1,2}$ at h_2	$d_{1,3}$ at h_1	$d_{1,4}$ at h_4	$d_{1,5}$ at h_5
d_3	$d_{1,1}$ at h_2	$d_{1,2}$ at h_3	$d_{1,3}$ at h_5	$d_{1,4}$ at h_1	$d_{1,5}$ at h_4

Note again the two kinds of indexing a host with h_j for referring to all partitions on that host and with $h(i,j)$ for referring to the same partition index j of all SVs i. Accordingly, index j of two partitions $d_{i1,j}$ and $d_{i2,j}$ may be equal for different SVs $i1 \neq i2$, but not necessarily located on the same host $h(i1,j) \neq h(i2,j)$.

Partition migration. Variable-size partitioning allows adapting to data clusters, but can cause growing clusters in several partitions $d_{i,j}$ to concentrate on one host $h(i,j)$, which in turn can lead to load imbalances. In order to balance the load in such a case, clusters on that host are migrated to hosts $h(i,j-1)$ or $h(i,j+1)$ and merged with their adjacent partitions $d_{i,j-1}$ or $d_{i,j+1}$, respectively.

Candidates for migration are chosen from the list of all clusters $C_i[c_{z'1},c_{z1}]$ on host h_j, which is sorted in descending order after cluster access cost $sort_{desc}(\mathcal{A}(C_i[c_{z'1},c_{z1}]))$. Clusters with relative high access cost are those, whose owning LPs are located remote, thus migrating them will at least not increase the overall messaging effort of the application. Note that in this process a cluster is not distinguished from a single access region that has no intersections. The migration is continued for all clusters, until the accumulated cluster access costs of neighbouring hosts are equal, relative to their local load:

$$\Delta\mathcal{H}_{j-1} * \mathcal{H}_{j-1} = \Delta\mathcal{H}_j * \mathcal{H}_j = \Delta\mathcal{H}_{j+1} * \mathcal{H}_{j+1}$$

for hosts h_{j-1}, h_j and h_{j+1}, respectively. Where

$$\mathcal{H}_j = \sum_{s_i}\sum_z \mathcal{A}(C_i[c_{z1}, c_{z2}])$$

Where z denotes the number of regions on SV d_i. Partition migration is initiated for all hosts in every second phase of $2 * \Delta f$ state changes, alternating with the task for optimal placement of LPs. This schema balances the load in a simulated annealing fashion, where the grade of annealing is adapted over Δf for a given application. $\Delta\mathcal{H}_j$ is the load factor of host j and is re-calculated based on the local processor load.

Cluster splitting. In cases where the above equality for load balancing cannot be achieved on the overloaded host, the next cluster in the list that exceeds the equality is split, just like a partition is split. A split cluster is principally undesired, as it increases messaging effort, but is unavoidable, if balanced load is required. This capability enables the DDM to adapt to applications that may develop increasingly growing clusters or, in the worst case, even a single "all-one-cluster".

2.8 Optimal Placement of Logical Processes

For the purpose of balancing the process load between the hosts, each LP^e is placed on host k with the highest access cost $\mathcal{A}^e_k$ for LP^e:

$$\max_v\left(\sum^{s_i}(\mathcal{A}_k(r^e_{i,k}) + \mathcal{A}_k(u^e_{i,k}))\right) \text{ iff } r^e_{i,k} \cap u^f_{i,k} \neq \emptyset$$

Where $v \leq s$ is the total number of regions of LP^e, e, f = [1..g] and $e \neq f$, and $\mathcal{A}^e_k = \mathcal{F}^e_k * \mathcal{L}^e_k$ with $i \leq s$ of s SVs. A subscription region without any intersections with update regions is excluded from the calculation, as it has always zero access cost, wherever LP^e is located.

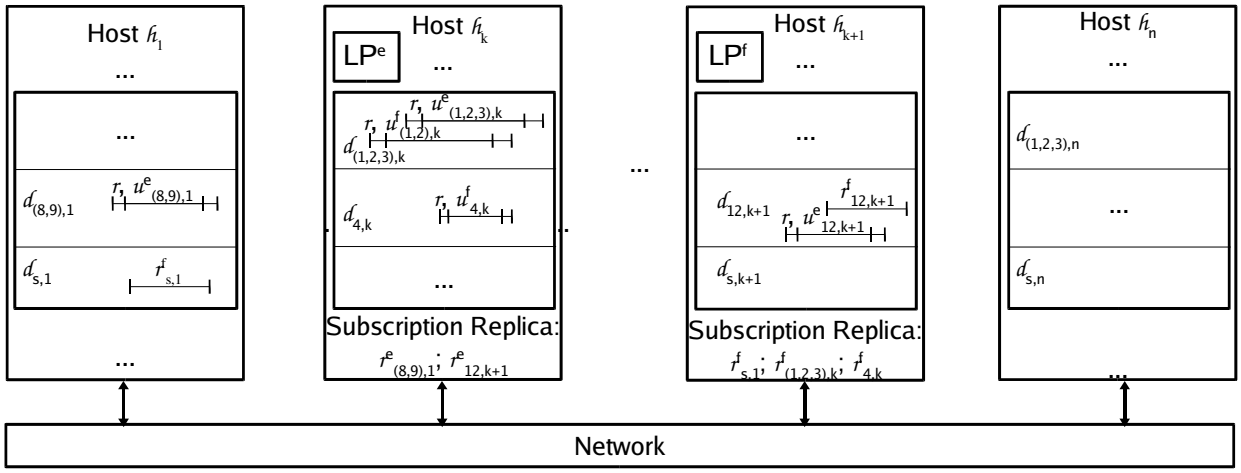
A sample distribution of access regions and LP placement for hypothetical access costs is sketched in (*Figure 3*). Note that, for instance subscription region $r^f_{s,1}$ was not included in the calculation for the placement of LP^f, as it does not intersect with any update region. Also, compound SV $d_{(1,2,3)}$ is depicted as one-dimensional region, for simplicity, but is to be interpreted as cubical region.

2.9 Primitive Operations on Shared Data

LPs can access the shared data over three primitive operations: reading and updating a region and changing region boundaries. Each operation initiates a well defined sequence of messages for keeping the shared data globally consistent, by complying with the above sketched data distribution schema [11].

Message content. The network message sent for synchronising a remote SV is uniform. It contains the SV value and a local time stamp.

Reading subscribed data. When LP^e subscribes to a region $r^e_i[c'_1, c'_1]$ within partition $d_{i,j}$ on host $h(i, j)$, then the region is copied to the host where LP^e is located. Any following read operation from LP^e within $r^e_i[c'_1, c_1]$ is then performed locally. Thus two messages are required for each new subscription, if the partition is remote. Hence the messaging overhead $\mathcal{P}$ for reading subscribed data is $\mathcal{P}_r = 2$.



$d_{i,j}$: Partition j of State Variable i

$r, u^g_{i,j}$: Subscription/Update Regions r/u on $d_{i,j}$ of LP^g

h(i, j): Network Address of Partition j of SV d_i on Host h(i, j)

max($\mathcal{A}_1(r, u^e_{(8,9),1})$, $\mathcal{A}_{k+1}(r, u^e_{12,k+1})$, $\mathcal{A}_k(r, u^e_{(1,2,3),k})$) = k : Placing LP^e on Host h((1,2,3), k) = h_k

max($\mathcal{A}_k(r, u^f_{(1,2,3),k})$, $\mathcal{A}_k(r, u^f_{4,k})$, $\mathcal{A}_{k+1}(r^f_{12,k+1})$) = k+1 : Placing LP^f on Host h(12, k+1) = h_{k+1}

Figure 3. Sample Data Distribution and Optimised Placement of Two Logical Processes for Given Data Access Schema

Updating associated data. If an update within a region $u^e_i[c'_1, c_1]$ is remote, with respect to the associated LP^e, then one message is required. If the update region intersects with a total number of v remote subscription regions, then further v messages are required to propagate the update on the original data to v replica, where v = [0..n] and n is the total number of hosts. Thus at most v + 1 messages are required. The messaging overhead for updating associated data becomes than $\mathcal{P}_u$ = v + 1.

Updating region boundaries. Changing region boundaries, caused for instance by a moving simulation object, requires a new calculation of intersections with other regions. If the change on the boundary is within the boundaries of this partition, then no messages are required. Otherwise, one message is send to the host with the related adjacent partition. One further message is sent back to the region owner, if the new boundary has crossed an adjacent partition on an other host. Thus at most two messages are required for this operation, yielding a messaging overhead of $\mathcal{P}_b$ = 2.

2.10 Performance Evaluation

The overall performance of the DDM model in terms of messaging and time complexity for primitive operations on shared data is sketched in (*Table 5*) and compared with the region and grid-based algorithms for the following two cases:

- Evenly distributed interest: interest regions and the overall access cost of the LPs are evenly distributed over the hosts (peers).
- Single-clustered interest: interest regions and the overall access cost of the PLs are concentrated on one host (server).

Server. All techniques perform equal for the case that all s SVs are maintained at a server. For read operations, the worst case is assumed, where each read is preceded by an update on the original data and therefore requires a message for replication. Messaging complexity is only O(n), as in the worst case all n hosts may send a message to the server in one state change. Time complexity is O(n log n), since interest matching is performed only on the server. For instance, the region technique combined with the grid on a server has been proposed in [20].

Region; peer. If all SVs are fully replicated at all peers, peer-to-peek communication is required with the region technique. Interest has to be matched at all n hosts over all s SVs, no matter whether all interest regions are single-clustered or not.

Table 5. Messaging, Time and Space Complexity of Operations of DDM Techniques for Evenly Distributed and Single-clustered Interests

Technique	Distribution	Operation				Data Connectivity (Space)
		Read or Update (Messaging)		Move (Messaging/Time)		
		Evenly Distributed	Single Cluster Evenly	Distributed/Matching	Cluster/Matching	
All	Server	O(n)	O(n)	O(n)/O(n log n)	O(n)/O(n log n)	O(0)
Region	Peer	$O(n^2)$	$O(n^2)$	$O(n^2)$/O(n log n)	$O(n^2)$/O(n log n)	O(0)
Grid	Peer	$O(n^2)$	O(n)	$O(n^2+G)$/O(0)	O(n)/O(0)	$O(n\ 2^n)$
Partition	Peer	$O(n^2)$	$O(n^2)$	$O(n^2+P)$/O(0)	$O(n^2)$/O(0)	$O(n^3\ 2^n)$
Grid with Region	Peer	$O(n^2)$	O(n)	$O(n^2+G)$/O(log n)	O(n)/O(n log n)	$O(n\ 2^n)$
Partition with Region	Peer	$O(n^2)$	$O(n^2)$	$O(n^2+P)$/O(log n)	$O(n^2)$/O(log n)	$O(n^3\ 2^n)$

Grid or partition; peer; read or update. In case of gridded or partitioned distribution of the SVs, peer-to-peer communication complexity is equal for read/update operations. It reduces to O(n) for the single cluster located at a single peer in the grid, but remains equal, if the cluster is split over all peers within partitions.

Grid or partition; peer; move. In case of move operations, peer-to-peer communication complexity is principally equal with the grid and partition techniques, as the administrative messaging overheads $\mathcal{G}$ and $\mathcal{P}$ are principally equal. Interest matching reduces to O(log n) at each host. If an application develops a single cluster, then messaging reduces to O(n) in the grid, assuming that moving interest regions will remain within the cell on that peer. If the single cluster is split over all peers within partitions however messaging remains at $O(n^2+\mathcal{P})$. Interest matching increases to O(n log n) within the single grid cell and reduces to O(log n) within each peer's partition.

3 Conclusions

The introduced IM model inherits its adaptability from the dynamics of the data distribution schema, particularly the partition adaptation to data cluster formations and the optimal LP placement, by considering network latencies. Where LP clusters are recognised indirectly through the local processor load. The model benefits from the fact that no receiver-side filtering is required, as all LPs receive the exact amount of messages they require. The administrative overhead is restricted to a few network messages for maintaining partition adjacency information. These capabilities of the model facilitate further load balancing as well as scalability. The local overhead for interest matching and cluster detection is notable, but does not introduce more administration than the grid approach. The additional administration for adjacency maintenance does scale with increased partition granularity, but remains still minimal and therefore does not dramatically influence the scalability. In fact, the administrative overhead is expected to scale almost linear with increased number of peers.

Remarkable is the nearly constant performance of the partition technique that converges in all cases to the messaging complexity $O(n^2)$ and time complexity O(log n), which facilitates dynamic adaptability. Although the grid technique outperforms the partition technique in some cases, adaptability and dynamically scalability are the significant advantage over the grid technique. Achieving continuously balanced load throughout an application is the only way to guarantee real-time response at all times.

Our next goal is to examine the practical performance of the partition-based DDM technique for some representative deterministic and non-deterministic applications in critical constellations. The space complexity $O(n^3\ 2^n)$ of a partition, for pointing to variable-size adjacent partitions, might prove critical in practice, especially for the case where clusters are closely located, even for only s = 3 dimensions. The performance however is expected to improve in general for increased partition granularity.

4 References

[1] Benford, Steve; Fahlen, Lennart; 1993; "A Spatial Model of Interaction in Virtual Environments", *Proc. Third European Conference on Computer Supported Cooperative Work* (ECSCW'93)

[2] Boukerche, Azzedine; Roy, Amber; 2002; "In Search of Data Distribution Management in Large Scale Distributed Simulations"; *Summer Computer Simulation Conference*

[3] Boukerche, Azzedine; Amber, Roy; 2002; "Dynamic Grid-Based Approach to Data Distribution Management"; *Journal of Parallel and Distributed Computing* (JPDC); Elsevier

[4] Cai, W.; Turner, S. J.; Zhao, H.; 2002; "A Load Management System for Running HLA-Based Distributed Simulations over the Grid"; *Workshop on Distributed Simulation and Real-Time Applications* (DS-RT'02)

[5] HLA 1.3; U.S. DoD DMSO; 1996; "HLA Data Distribution Management Design Documentation Version 0.5, 1997"; http: // hla.dmso.mil

[6] Helfinstine, B.; Wilbert, D.; Torpey, M.; Civinskas, W.; 2001; "Experiences with Data Distribution Management in Large-Scale Federations"; *Winter Simulation Interoperability Workshop* (SIW), Orlando

[7] Horling, Bryan; Mailler, Roger; Lesser, Victor; 2004; "Farm: A Scalable Environment for Multi-Agent Development and Evaluation"; *Advances in Software Engineering for Multi-Agent Systems*; Springer

[8] van Hook, Daniel J.; Rak, Steven J.; Calvin, James O.; 1994; "Approaches to Relevance Filtering"; *Distributed Simulation* (DIS) Workshop

[9] van Hook, Daniel J.; Calvin, James O.; Newton, Michael K.; Fusco, David A.; 1994; "An Approach to DIS Scaleability"; 11th Workshop on Standards for the Interoperability of Distributed Simulations

[10] van Hook, Daniel J.; Calvin, James O.; Spring 1998; "Data Distribution Management in RTI 1.3"; *Simulation Interoperability Workshop* (SIW); IEEE

[11] Kumova, Bora İ.; 2005; "Scalable Interest Management in Distributed Simulation of Intelligent Systems"; *Symposium on Design, Analysis, and Simulation of Distributed Systems* (DASD'05); The Society for Modelling & Simulation International (SCS)

[12] Logan, B.; Theodoropoulos, G.; 2000; "Dynamic Interest Management in the Distributed Simulation of Agent-Based Systems"; Proceedings of 10th AI, Simulation and Planning Conference (RAE2001); Society for Computer Simulation International

[13] Merchant, F.; Bic, L. F.; Dillencourt, M. B.; 1998; "Load Balancing in Individual-Based Spatial Applications"; *Conference on Parallel Architectures and Compilation Techniques* (PACT'98)

[14] Morse, Katherine L.; Steinman, Jeffrey S.; 1997; "Data Distribution Management in the HLA Multidimensional Regions and Physically Correct Filtering"; *Summer Simulation Interoperability Workshop* (SIW); IEEE

[15] Morse, Katherine L.; Bic, Lubomir; Tsai, Kevin; 1999; "Multicast Group for Dynamic Data Distribution Management"; *Proc. Of the 31th Society for Computer Simulation Conference* (SCSC)

[16] Morse, Katherine L.; 2000; "An Adaptive Distributed Algorithm for Interest Management"; *PhD Thesis*; University of California, Irvine

[17] Petty, Mikel D.; 2001; "Comparing high level architecture data distribution management specifications 1.3 and 1516"; *Simulation Practice and Theory*; Elsevier

[18] pRTI; "Pitch RTI"; http:// www.pitch.se

[19] Steinman, Jeff S.; Wieland, Frederick; 1994; "Parallel Proximity Detection and the Distributed List Algorithm"; *Workshop on Parallel and Distributed Simulation* (PADS); New York IEEE

[20] Tan, Gary; Zhang, Yu Song; Ayani, Rassul; 2000; "A Hybrid Approach to Data Distribution Management"; *Workshop on Distributed Simulation and Real-Time Applications* (DS-RT); IEEE Computer Society