# SYNCHRONIZED DATA DISTRIBUTION MANAGEMENT IN DISTRIBUTED SIMULATIONS

Ivan Tacic
Richard M. Fujimoto
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

KEYWORDS
High Level Architecture, Run-Time Infrastructure, Time Management, Data Distribution Management, Interest Management

## ABSTRACT

*Data distribution management (DDM) is concerned with the problem of efficiently distributing state information among the entities in a distributed simulation. While heavily studied in the DIS community for training applications using real-time synchronization, this problem has received considerably less attention for logical time simulations, and little performance data has appeared in this regard. This paper is concerned with evaluating the performance of a logical time synchronized DDM mechanism. A DDM algorithm based on the services specified in the DoD High Level Architecture (HLA) using routing spaces is described, as well as its implementation on a network of workstations environment. Performance measurements indicate that the time overhead to provide correct logical time synchronization of the DDM mechanisms is small relative to network communication overheads, suggesting that logical time synchronized DDM mechanisms can perform as efficiently as real-time synchronized mechanisms in networked workstation environments.*

## 1 INTRODUCTION

Consider a distributed simulation exercise composed of a collection of simulators, each modeling a vehicle moving over a two dimensional space. Assume each vehicle has a sensor that can detect other vehicles that are within a certain distance of its current location. Each vehicle should receive position update messages generated by other vehicles within range of its sensor. Some mechanism is required to determine when vehicles become visible to other vehicles, and to ensure each simulator receives messages from all vehicles within sensor range. The distributed simulation services that support this functionality are sometimes referred to as data distribution management (DDM) services.

DDM takes on a somewhat different flavor in distributed simulations constructed by "federating" existing simulators compared to a traditional parallel discrete event simulation (PDES) program. PDES programs typically assume each logical process (LP) is responsible for determining which other LPs should receive the messages it generates. By contrast, federated simulation systems typically implement this functionality in the underlying distributed simulation software, referred to as the Run-Time Infrastructure (RTI) in High Level Architecture terminology rather than within the simulation model. Each simulator (federate) specifies via interest expressions what messages are of interest.

Data distribution management (DDM) mechanisms must provide efficient, scalable support for large-scale distributed simulations. They have been extensively studied in distributed simulations for training. Work in distributed simulation environments in the SIMNET (SIMulator NETworking) project[1] and many DIS systems broadcast each state update (event) to all simulators in the exercise. It is well known that this approach does not scale because the amount of communications is $O(N^2)$ where N is the number of processors. CPUs become bogged down processing incoming messages, most of which are discarded (for large N) because they are not relevant to the simulator(s) within the processor. Further, communication bandwidth requirements become excessively large as N increases. It is estimated that 375 MBits per second per platform

1087-4097/98 $10.00 © 1998 IEEE

would be required for a simulation exercise including 100,000 players[2].

Several approaches to attacking this problem have been proposed (see [3] for a survey on this subject). Virtually all use some mechanism to only send messages to the destinations that have need of the information rather than broadcast it. For example, in the Joint Precision Strike Demonstration (JPSD)[4], federates indicate what information they wish to receive by specifying predicates on entity attributes. Many simulators, e.g., ModSAF[5], CCTT[6], and NPSNET[2] use grid cells to filter information. A two-level hierarchical filtering scheme used in an optimistic parallel simulation is described in [7], and a generalization of grid cells using a construct called routing spaces is used in STOW[8]. The focus on the work described here is on the routing space approach that has been incorporated into the baseline definition of the High Level Architecture (HLA), though many of the techniques and mechanisms are applicable in other contexts.

Thus far, most of the DDM work has focused on large-scale, real-time training simulations. By comparison, only a limited amount of work on this subject has been concerned with distributed logical time simulations[7,13,14]. Directly applying training simulation DDM mechanisms based on wallclock time semantics to logical time simulations will lead to errors, e.g., some simulators will not receive messages that they should or they will receive them in their logical time past. These problems are discussed in section 2, and an approach to efficiently solving them is described in sections 3 and 4. An implementation and evaluation are described in sections 5 and 6.

The distributed simulation is referred to as a federation, and consists of a collection of individual simulators, termed federates. A federate is essentially equivalent to a logical process in PDES terminology. If one federate can generate messages that will be received by a second, a connection is said to exist from the first to the second. The network formed by federates and their connections is referred to as the federate topology. Also, the "sending federate" refers to the federate on the sending side of the connection, and the "receiving federate" be the federate on the receiving side.

## 2 PROBLEM DESCRIPTION

Two issues must be addressed:

- each federate must receive all messages it has specified through its interest expressions.
- no federate should receive any messages in its (logical time) past.

By contrast, these issues can usually be relaxed in training applications.

To illustrate the first situation, consider two tanks T1 and T2. If T1's sensors indicate awareness of an enemy tank T2 in a certain time interval, all events generated by tank T2 with time stamp in that interval must be received by tank T1. It may happen that tank T2 generated events in this interval before (in wallclock time) T1 had specified interest in receiving these events. The RTI must ensure these previously generated events are sent to T1.

To address this problem a log of previously generated messages is maintained (see Figure 1). Those messages that are not sent to the federate when the event had been generated are retrieved from the log, and sent to the "late" federate, e.g., tank T1 in the example above. This approach is described in greater detail in[13,14]. A variation on this approach was implemented in this study, and is described in Section 4.

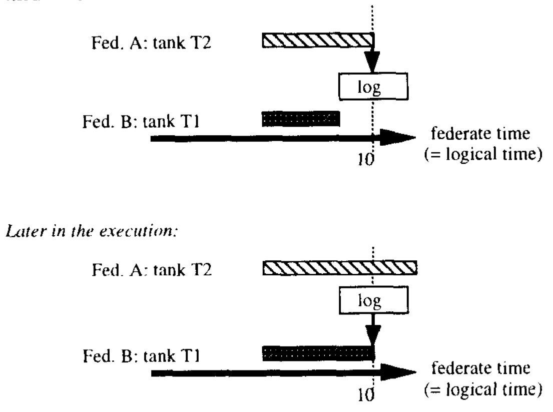

**Figure 1. Use of a log to avoid losing messages.**

The second issue is concerned with preventing federates from receiving messages in their past. This issue is concerned with ensuring proper synchronization of changes in the federate topology. For example, suppose in the previous example T2 is at logical time 10. Suppose T1 advances beyond logical time 10 because there is no connection from T2 to T1. Suppose T2 now establishes a new connection to T1 at logical time 10, and sends a message with time stamp 10, in T1's logical time past. Situations such as this must be prevented.

This problem is typically addressed in parallel simulation systems by a connection transfer protocol [9,12]. The basic idea is one logical process $LP_S$ can only establish a new connection to a second process $LP_R$ by receiving a "handle" for the connection from a third logical process $LP_X$ that already has a connection to $LP_R$. Because $LP_X$ cannot send messages into $LP_R$'s past, it is straightforward to ensure that $LP_S$ will not either. This approach is not suitable for federated simulation sytems,

RIGHTSLINK

however, because an LP (federate) can "spontaneously" open a new connection to another federate. For example, in Figure 1, T1 moving into the range of T2's sensor causes a new connection to be established from T1 to T2, independent of any other federates. Thus, there need not be a third party to coordinate establishment of the new connection, as is required in a connection transfer protocol. We propose an approach based on a concept called *connection lookahead* that is described in detail in the next section.

## 3 CONNECTION LOOKAHEAD

There are several different flavors of dynamic topology changes: sender initiated new connection, receiver initiated new connection, transfer of a connection from another federate, and removal of a connection. All but sender initiated new connection are easy to handle. In the case of receiver initiated new connection, it suffices to define that the new connection takes effect at the current logical time of the federate. Because the new connection does not take effect until the federate's current time, there is no possibility the new connection will result in the federate receiving messages in its past. This new connection can simply be taken into account when computing LBTS before allowing the federate to advance to a new logical time. On the other hand, connection transfer protocol mentioned above, can be used to handle connection transfers from another federate, while removing a connection does not require any action at all.

Sender initiated new connections appear to be the thorniest to handle. Two approaches to preventing causality errors are described below. The first constrains the behavior of the receiving federate in such a way that prevents it from advancing "too far ahead," thereby eliminating the possibility it will receive a new message in its past. This is referred to as the *receiver constrained approach.* The second places constraints on the sending federate to prevent it from sending a message into the past of a receiving federate by placing lookahead restrictions on new connections. This is referred to the *sender constrained approach.* It will be seen that the "pure" sender constrained approach has some serious deficiencies, so a variation on this idea utilizing a concept called connection lookahead is proposed as an alternate solution. To simplify the discussion, we assume each federate specifies a single connection lookahead value.

### 3.1 Solution 1: Receiver Constrained

One can prevent the receiving federate from advancing "too far" ahead of all *potential* sending federates in order to ensure it does not receive messages in its past. For example, if the receiving federate is constrained so that it cannot advance more then $L_i$ units of time ahead of each potential sending federate $F_i$, where $L_i$ is the lookahead for $F_i$, then it is guaranteed that the receiving federate will not receive any new messages in its past, no matter how the topology changes during the execution.

In its simplest form, one may assume any federate can establish a new connection to any other federate at any time during its execution. For example, a federate could indicate it can publish certain attribute values, and register a new object instance of that class, all at its current logical time. This will establish a new connection to every federate subscribed to that class. Once the connection is established, the federate can then send attribute updates for that object with time stamp equal to its current time plus its lookahead. To ensure no federate will receive a message in its past, any federate is prevented from advancing more the $L_{min}$ units of logical time ahead of any other federate, where $L_{min}$ is the minimum lookahead of any federate in the federation.

The main disadvantage of this approach is it is somewhat restricting in that it does not allow federates to advance more than the federation's minimum lookahead amount ($L_{min}$) ahead of any other federate. In effect, this approach conservatively assumes a fully connected topology among federates, so any changes in the federation topology are of no consequence. An improvement is possible if federates could deduce in advance all possible channels of communication. However, the same shortcoming of taking into account those connections on which the communication never or rarely takes place remains.

### 3.2 Solution 2: Sender Constrained

One can constrain the sending federate so that it cannot send messages in the past of the receiving federate. Suppose the sending federate $F_S$ is at simulation time $T_S$ and the receiving federate $F_R$ be at simulation time $T_R$ when a new connection is established from $F_S$ to $F_R$ (see Figure 2). If one specifies that the new connection does not take effect at the sender until it reaches logical time $T_R - L_S$, where $L_S$ is the lookahead of $F_S$, one can ensure the receiver will not receive messages in its past.

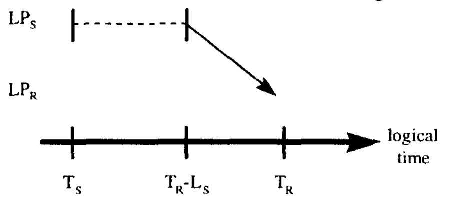

**Figure 2. Sender constrained approach to establishing a new connection.**

The problem with this approach is there is no restriction on how long a federate must wait before the new connection is established, because there is no a priori restriction on how far one federate can advance ahead of another. For example, if there are two "sub-federations" within a single federation defined so that no messages pass between them, one sub-federation could advance arbitrarily far ahead of the other. If a connection between the two sub-federations must now be established, there is no a priori limit on how much logical time must elapse before the new connection can be established.

## 3.3 Solution 3: Connection Lookahead

A third approach is to use a combination of the sender and receiver constrained approaches. Each federate specifies a new lookahead, called its *connection lookahead,* that specifies the minimum amount of time into the future the federate can establish a new connection with another federate. Again, assume we wish to establish a new connection from federate $F_S$ at logical time $T_S$ to federate $F_R$. Let $L_S$ be the lookahead of $F_S$. The smallest time stamp of any message $F_S$ can send over a new connection to $F_R$ is $T_S + CL_S + L_S$. This allows $T_R$ to advance up to $CL_S + L_S$ units of logical time ahead of $F_S$. The connection lookahead must be greater than or equal to zero. The receiver constrained approach, in effect, assumes $CL_S$ is zero.

Just as lookahead restricts how far one federate may advance ahead of another federate, connection lookahead also imposes a similar constraint. Specifically, if no connection exists from $F_S$ to $F_R$, then $F_R$ cannot advance more than $CL_S + L_S$ units of logical time ahead of $F_S$ in case a new connection is established. Assuming no constraint on which federates may begin communicating with which other federates, this effectively says no federate can advance more than W units of simulation time ahead of any other, where W is defined as max $(CL_i + L_i)$ over all federates $F_i$ in the federation.

## 4 A DDM LAYERED ARCHITECTURE

An approach that assures federates will receive all the messages they are supposed to receive is convenient to view as two layers of software (see Figure 3). The upper interest management layer (IM) is providing a federate with a simple interface for expressing its interest expressions and maps them to the lower distribution list layer (DL) which is analogous to a collection of newsgroups, where users (e.g. federates) either subscribe (*Add* operation) or unsubscribe (*Delete* operation) to them at desired times. This approach was initially proposed in [14], and now we extend it by refining the IM layer, so that multiple overlapped regions per federate may coexist.

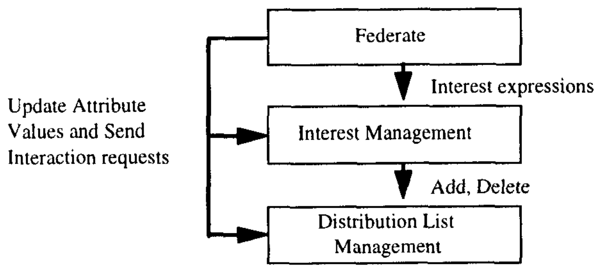
Federate
Interest expressions
Update Attribute Values and Send Interaction requests
Interest Management
Add, Delete
Distribution List Management

**Figure 3. DDM Organization.**

## 4.1 INTEREST MANAGEMENT LAYER

The fundamental concept underlying interest expressions is the *routing space.* A routing space is a normalized multidimensional coordinate system in which federates indicate interest in receiving or providing updates via *subscription* and *update regions.* Regions are rectangular (in N dimensions) and are specified by indicating *extents,* with one extent for each dimension. Each extent indicates the portion of that dimension covered by the region. A federate may issue **ModifyRegion** for changing region extents and **UpdateAttributeValue** to the IM layer (see [10] for the complete reference). When an attribute is updated, a message is sent only to those federates whose subscription region overlaps with the update region.

Here, we describe one approach to implementing the IM based on partitioning the routing space into fixed non-overlapping cells and creating a distribution list for each cell. We will assume identical cells here to simplify the presentation.

When the UpdateAttributeValue service is invoked, it causes an invocation of Update in the DL layer for each cell that includes a portion of the update region for the modified attribute. This may result in some messages being sent to federates whose subscription region lies outside the update region (but includes one or more cells in common with the update region). However, such messages can be filtered at the destination.

ModifyRegion for an update region requires only the IM to record the new extents. ModifyRegion for a subscription region at logical time T, requires the IM to determine what Add/Delete operations are to be passed to the DL. Actually, as will be described momentarily, the mechanism is more complex than it would initially appear, so we define two operations internal to the IM layer, *Subscribe* and *Unsubscribe*, that can be viewed for now as being synonymous with Add and Delete, respectively.

If there is no ModifyRegion for the same region at logical times greater than T, the IM invokes

Subscribe/Unsubscribe operations on cells to define the new region (see example in Figure 4-a).

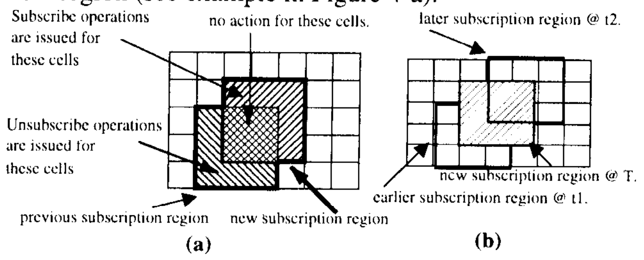
Subscribe operations are issued for these cells
no action for these cells.
Unsubscribe operations are issued for these cells
previous subscription region
new subscription region
(a)
later subscription region @ t2.
new subscription region @ T.
earlier subscription region @ t1.
(b)

**Figure 4. New subscription region extents, if this is the latest or not (in logical time) change for that region (situations *(a)* and *(b)* respectively).**

If a version of a given region already exists at time t2 (t2>T, Figure 4-b), the set of Subscribe/Unsubscribe operations depend on the cell status at t1, t2, and T, as shown in Table 1, where t1 is the logical time of the most recent (in logical time) ModifyRegion operation earlier than T (t1 < T < t2). A '1' in this table indicates the federate is subscribed at the specified time, and a '0' indicates it is not subscribed. For example, the third row in the table indicates that if the federate is not subscribed to a cell at t1 and t2, and a subscription is made at T, the IM will issue Subscribe@T, Unsubscribe@t2.

**Table 1. IM1 actions for a new subscription region at time T.**

| region@t1 | region@T | region@t2 | IM1 actions |
|---|---|---|---|
| 0 | 0 | 0 | No_action |
| 0 | 0 | 1 | No_action |
| 0 | 1 | 0 | Sub@T, Unsub@t2 |
| 0 | 1 | 1 | Sub@T, Unsub@t2 |
| 1 | 0 | 0 | Unsub@T Sub@t2 |
| 1 | 0 | 1 | Unsub@T Sub@t2 |
| 1 | 1 | 0 | No_action |
| 1 | 1 | 1 | No_action |

The IM layer is complicated by the fact that a federate may subscribed to a cell multiple times, e.g., if more than one entity within the federate have indicated subscriptions to regions overlapping the same cell. For this reason, counters are used to indicate the number of times a federate is subscribed to a cell. Each counter shows the number of subscription regions that are overlapping a specific cell (denoted as Counter:value@time). Subscribe/Unsubscribe@t will increase/decrease by 1 the counters with time stamps greater than or equal to t, respectively (see Figure 5). When the counter value increases to 1, an Add is generated, and when it drops to 0, a Delete is generated. Improvements are possible if Subscribe/Unsubscribe operations are issued for a specified interval (see Subscribe_in_interval), because it limits searching through a sequence of counters. Unsubscribe_in_interval is similar.

```
Subscribe(T){
 find closest counter C:n@t such that t<=T;
 if(t<T) create new counter C_new:n@T;

 for every counter C:n@t such that t>=T{
   C.n=C.n+1;
   if(C.n==1) issue ADD@t;
   else if(C.n==0) issue DELETE@t;
 }
}

Unsubscribe(T){
 find closest counter C:n@t such that t<=T;
 if(t<T) create new counter C_new:n@T;

 for every counter C:n@t such that t>=T{
   C.n=C.n-1;
   if(C.n==0) issue DELETE@t;
 }
}

Subscribe_in_interval(start_time:Ts,stop_time:Tu){
 find closest counter C:n@t such that t<=Ts;
 if(t<Ts) create new counter C_new:n@Ts;
 find closest counter C:n@t such that t<=Tu;
 if(t<Tu) create new counter C_new:n@Tu;

 for every counter C:n@t such that Ts<=t<Tu{
   C.n=C.n+1;
   if(C.n==1) issue ADD@t;
   else if(C.n==0) issue DELETE@t;
 }
}
```

**Figure 5. Counters are used to generate Add/Delete operations.**

## 4.2 DISTRIBUTION LIST LAYER

DDM can be viewed as maintaining a collection of *distribution lists* indicating which federates should receive attribute updates at what times. If operations were issued in time stamp order, data distribution would be trivial. In that case, Add/Delete operations would simply update the distribution list, and Update operations would transmit messages to the destinations in the list. Because the operations do not arrive in time stamp order, different versions of the distribution list corresponding to different points in logical time have to be maintained. It is also necessary to keep a log of updates in order to furnish "late" subscribers with updates that are being issued but not sent to them, as was described in section 2.

For the completeness of this approach we must address fossil collection issues as well. Having in mind that UpdateAttributeValue and ModifyRegion and consequently appropriate IM and DL layer operations adhere to lookahead constraints, memory reclamation can be performed for distribution lists and logs that are older than the Global Virtual Time (GVT). Complete details on design of the DL layer can be found in [14].

## 5 IMPLEMENTATION

Multiple versions of different data structures are needed to correspond to different logical times. In our implementation we are using double linked lists, sorted by version time stamps. The data structures each federate has to maintain are shown in Figure 6. There is a multiple version data structure for each of the regions (u/s denote update/subscription regions) defined by the federate. Each element represents a set of extents for the region at a time given by the time stamp. Each of the cells in the routing space has a counter with its associated versions, as well as logs for Add/Delete operations, received from other federates. Instead of having one log, we have a log for every federate. This will speed up recording a new Add/Delete entry from a federate by not having to traverse log elements of other federates. More importantly, it will speed up finding a distribution list at a given time t, by not having to traverse irrelevant elements from some federates (i.e. elements whose time stamps are less than t).

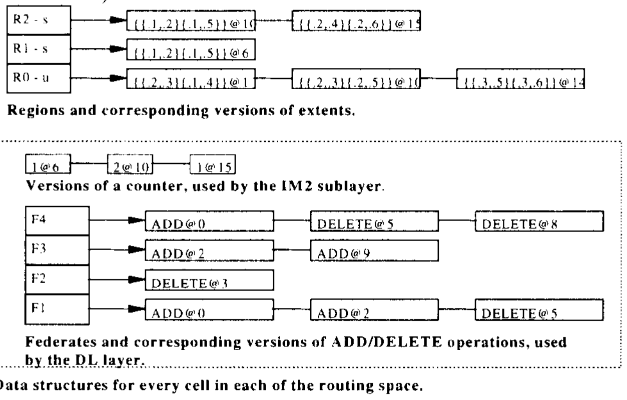
R2 - s
{{.1,.2}{.1,.5}}@10
{{.2,.4}{.2,.6}}@15
R1 - s
{{.1,.2}{.1,.5}}@6
R0 - u
{{.2,.3}{.1,.4}}@1
{{.2,.3}{.2,.5}}@10
{{.3,.5}{.3,.6}}@14
Regions and corresponding versions of extents.
1@6
2@10
1@15
Versions of a counter, used by the IM2 sublayer.
F4
ADD@0
DELETE@5
DELETE@8
F3
ADD@2
ADD@9
F2
DELETE@3
F1
ADD@0
ADD@2
DELETE@5
Federates and corresponding versions of ADD/DELETE operations, used by the DL layer.
Data structures for every cell in each of the routing space.

**Figure 6. Data structures for federate:F0.**

An UpdateAttributeValue@7 issued for an attribute associated with an update region that is equal to a cell depicted above, causes a backward search through the Add/Delete logs. Traversed elements are: F4:8,5;F3:9,2;F2:3;F1:5. The distribution list for this update is {F3}, and a message containing the update is being sent to F3. Note that we could have many log elements for F4 between (2,5), that would have to be searched in the case of a unique log, but not in our case.

Issuing a ModifyRegion for an update region causes a new set of extents to be recorded, while the same operation for a subscription region requires additional actions as explained in section 4.1.

There is an optimization where the search starts from the last log's entry that has previously been accessed, which could eliminate the influence of distance between federates in searching through the log. Instead, we would have either constant time or time proportional to the log density ratios of different federates.

## 6 PERFORMANCE EVALUATION

### 6.1 WORKLOAD

The benchmark application for our synchronized DDM (SDDM) system is a distributed simulation of moving entities (tanks/aircraft) across a simulated world (routing space(s)). Entities are simulated by federates that may reside on different processors. Each entity is associated with update and subscription region (1 cell for update, and 6x6 cells for subscription regions). The path of each entity follows a random walk, with the entity equally likely to move in any new direction (north/south/east/west). Logical time stamps for ModifyRegion/UpdateAttributeValue are taken from an exponential distribution with mean of 0.1sec and 0.2sec for update and subscription regions, respectively. Events are fired when the corresponding wall clock time (logical time + constant) is reached.

The hardware platform we used was a cluster of Sun-Ultra1 workstations with 167 MHz Sparc-Ultra processors connected by the 100 Mbps Ethernet. All messages are transmitted using (unicast) TCP/IP message sends.

### 6.2 OVERHEADS

According to algorithms presented in section 4, it is easy to conclude that SDDM overheads depend directly on DL and IM layer logs densities as well as on the distance between federates. Although they are highly application specific, our experiments were designed to give an upper bound like behavior by taking measurements at the federate which is trailing all other federates in logical time. In this way, the search through the logs is the most expensive for that federate, and represents an upper bound performance at the other federates. We will also see that SDDM scales well with the number of entities (upper bound is assumed again by associating a region with each object). More importantly, the overheads tend to asymptotically decrease toward unsynchronized DDM (UDDM) overheads.

Figure 7 shows the number of entries traversed in the DL log as a function of the number of regions. That is easily translated to total search time during updates, using Figure 8 (this time includes recording a new update which takes 11us; we believe the irregular behavior of this

RIGHTSLINK

curve is due to variations in the time required by the C++ memory allocator). Since in our benchmark application all federates, except the one whose measurements are shown, have the same execution characteristics, the overhead for four/eight federates (three/seven DL logs) are approximately three/seven times larger than for two federates (one DL log). The UDDM's behavior is equivalent to SDDM searching through 1 log element, as depicted. The search time initially increases as the number of objects (entities) increases because the number subscribed to a cell increases. However, after some number of objects, the curve starts to descend, since IM generates fewer Add/Delete operations due to the increased probability that the federate is already subscribed to the cell.

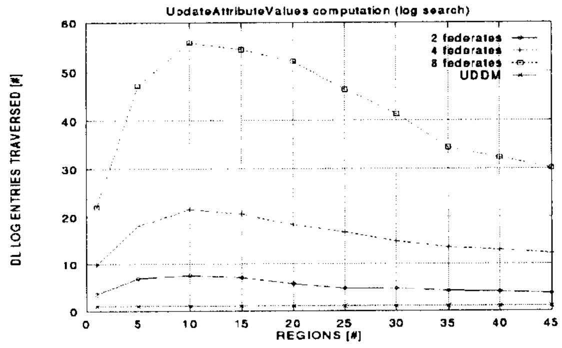

**Figure 7. Search time in the DL's log as a function of the number of regions.**

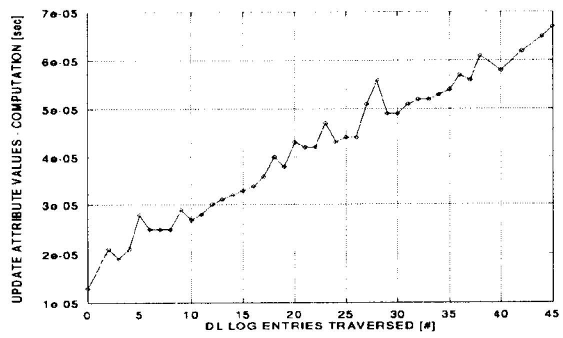

**Figure 8. Time to do an update as a function of the number of entries in the DL's log traversed.**

There is no additional communication overhead to do updates in SDDM relative to UDDM, because operations on the distribution lists can be done locally. However, network traffic caused by the update messages themselves (as opposed to overhead messages) will, in general, differ between the two. SDDM may generate either more or less messages than UDDM, since SDDM and UDDM may be utilizing different distribution lists in order to determine if it is necessary to send an update. Figure 9 shows communication time during updates, and as for searching time, it increases in proportion to the number of federates.

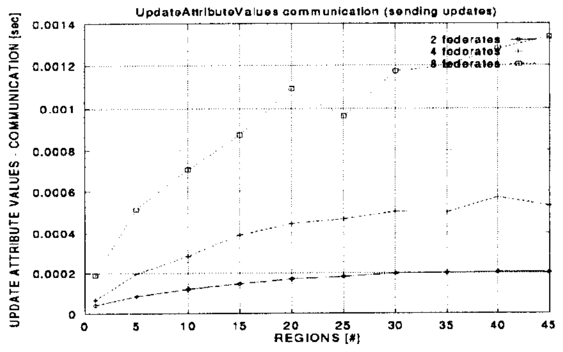

**Figure 9. Communication time during updates.**

Figure 10 shows the overall time required to perform update operations for synchronized and unsynchronized DDM. It is apparent that SDDM performs almost as well as UDDM in this implementation. This is because communication overheads dominate the execution time.

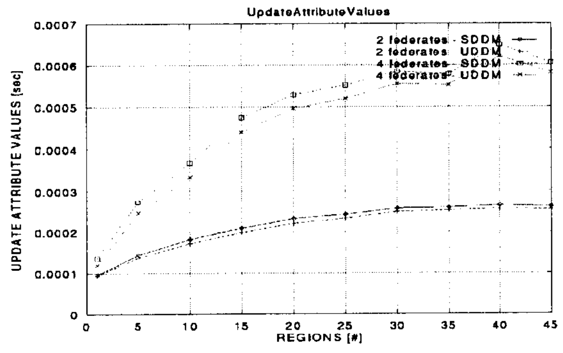

**Figure 10. Total UpdateAttributeValue time.**

## 7 ACKNOWLEDGMENTS

Work on this project was funded under a contract DAKF11-97-D-0001-0003 from DMSO. Comments from the HLA time management and data distribution management working groups are acknowledged.

## 8 SUMMARY

A mechanism is described to realize properly synchronized data distribution in distributed simulations using logical time. It includes a connection lookahead approach that allows dynamic network topology changes where federates can advance further away from each other, yielding the better performance. Also, it advocates a two-layer architecture that allows federates to have multiple overlapped regions, while ensuring data is routed to subscribers based on interest expressions and logical time semantics at the same time. This mechanism is applicable to a variety of data distribution schemes. An approach to implementing the interest expression layer (IM) for routing spaces such as those defined in the HLA is described.

Initial performance measurements indicate the time to perform update operations in synchronized DDM is only slightly larger than unsynchronized DDM because communication overheads dominate the execution time. Moreover, the approach scales well and overheads tend to asymptotically decrease toward unsynchronized DDM overheads. Additional experimentation and optimization are currently in progress.

## 9 REFERENCES

[1] C. Kanarick, "A Technical Overview and History of the SIMNET Project," in *Advances in Parallel and Distributed Simulation*, vol. 23: Society for Computer Simulation, 1991, pp. 104-111.

[2] M. Macrdonia, M. Zyda, D. Pratt, and P. Brutzman, "Exploiting Reality with Multicast Groups: A Network Architecture for Large-Scale Virtual Environments," in *1995 IEEE Virtual Reality Annual Symposium*, 1995, pp. 11-15.

[3] K. Morse, "Interest Management in Large Scale Distributed Simulations," University of California, Irvine Technical Report TR 96-27, 1996.

[4] E. T. Powell, L. Mellon, J. F. Watson, and G. H. Tarbox, "Joint Precision Strike Demonstration (JPSD) Simulation Architecture," in *14th Workshop on Standards for the Interoperability of Distributed Simulations*. Orlando, Florida, 1996, pp. 807-810.

[5] K. L. Russo, L. C. Shuette, J. E. Smith, and M. E. McGuire, "Effectiveness of Various New Bandwidth Reduction Techniques in ModSAF," in *Proceedings of the 13th Workshop on Standards for the Interoperability of Distributed Simulations*, 1995, pp. 587-591.

[6] T. W. Mastaglio and R. Callahan, "A Large-Scale Complex Environment for Team Training," *IEEE Computer*, vol. 28, pp. 49-56, 1995.

[7] J. S. Steinman and F. Wieland, "Parallel Proximity Detection and the Distribution List Algorithm," in *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*. Edinburgh, Scottland, 1994, pp. 3-11.

[8] D. J. Van Hook, J. O. Calvin, M. K. Newton, and D. A. Fusco, "An Approach to DIS Scalability," in *Proceedings of the 11th Workshop on Standards for the Interoperability of Distributed Simulations*, 1994, pp. 347-356.

[9] T. D. Blanchard, T. W. Lake, and S. J. Turner, "Cooperative Acceleration: Robust Conservative Distributed Discrete Event Simulation," in *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*. Edinburgh, Scotland, 1994, pp. 58-64.

[10] Defense Modeling and Simulation Organization, "HLA Interface Specification, V. 1.0," U.S. Department of Defense, Washington D.C. August 1996.

[11] Defense Modeling and Simulation Organization, "Data Distribution and Management Design Document, V. 0.2," U.S. Department of Defense, Washington D.C. December 1996.

[12] R. Bagrodia and W.-T. Liao, "Maisie: A Language for the Design of Efficient Discrete-Event Simulations", in *IEEE Transactions on Software Engineering,* 1994, (20):4, pp. 225-238.

[13] T. D. Blanchard and T. Lake, "A Lightweight RTI Prototype with Optimistic Publication", in *Spring Simulation Interoperability Workshop*, Orlando, Florida, 1997, pp. 551-560.

[14] Ivan Tacic, Richard M. Fujimoto, "Synchronized Data Distribution Management in Distributed Simulations", in *Spring Simulation Interoperability Workshop*, Orlando, Florida, 1997, pp. 303-312.