Area Virtual Time

D. K. Arvind and J. Schneiders
Institute for Computing Systems Architecture, Division of Informatics,
The University of Edinburgh, Mayfield Road, Edinburgh EH9 3JZ, SCOTLAND.
Email: dka|josch@dcs.ed.ac.uk

Abstract

We present a novel synchronisation algorithm for distributed discrete-event simulation (DDES), called the Area Virtual Time (AVT) algorithm. We first expose two orthogonal ideas of the synchronisation policy *for DDES, which is either conservative or optimistic, and the* time-keeping mechanism*, which is based on either Local or Global Virtual Times. The AVT algorithm is based on a network of virtual time regions, which is a happy medium between the Local Virtual Time (LVT) and the Global Virtual Time (GVT). The AVT algorithm permits the different parts of the simulation model to run either under LVT or GVT time-keeping mechanisms. This is particularly suited to models which are less than homogeneous. In those cases, mapping the models entirely to either one of the time-keeping schemes would not be efficient; or, the real-time nature of the interfaces precludes the use of GVT in those parts of the model. Our results demonstrate that the AVT algorithm progresses the simulation times faster than either the LVT or the GVT schemes, and is less sensitive to variations in some key model and communication parameters - a desirable property in distributed computation.*

1. Introduction

The simulation model can be represented as a directed graph, in which nodes and arcs correspond to logical processes (LP) and communication channels, respectively. The LPs exchange information in the form of messages containing time-stamped data which are termed as *events*.

Distributed Discrete Event Simulation (DDES) can run either under a conservative or an optimistic synchronisation policy. The policy determines the manner in which simulation time is progressed. *Conservative* DDES algorithms observe the strict partial ordering in the sequence of evaluation, i.e. result at a node is deemed safe to commit if, and only if, no message with a timestamp less than the local time will ever arrive at that node. This condition is relaxed in *Optimistic* DDES algorithms, in which events can be processed at the nodes in advance of their arrival. Errors due to mis-speculation are detected should they occur, and the algorithm recovers by rolling back in time to a consistent state, and then proceeding with the simulation.

The time-keeping mechanism is orthogonal to the synchronisation policy. Time is maintained in DDES in one of two ways: each LP maintains a local clock, called the *Local Virtual Time* (LVT), based on the time-stamps of incoming events; or, the algorithm maintains a *Global Virtual Time* (GVT), which is the minimum time-stamp of all the incoming events and those in transit. An increment in GVT is notified to all the LPs, which update their respective local clocks.

Optimistic DDES algorithms such as Time Warp [JeffersonS85] use a GVT-based time-keeping mechanism, whereas conservative ones based on the Chandy-Misra-Bryant algorithms [ChandyM79, Bryant81] use LVT-based time-keeping schemes. The Composite ELSA algorithm [ArvindS92] is an exception to this rule, since it integrates both the conservative and optimistic synchronisation policies, but uses only an LVT-based time-keeping mechanism.

We propose a novel algorithm for DDES called the Area Virtual Time (AVT) algorithm which supports both conservative and optimistic synchronisation policies, and is based on a combination of LVT and GVT time-keeping mechanisms. This is a happy medium between the extremes of either each LP maintaining its LVT, or calculating the GVT over all the LPs in the simulation model.

In the AVT algorithm, a set of LPs is mapped to a Virtual Time Area (VT-Area), which gives rise to a network of VT-Areas. The choice of mapping a particular set of LPs to a VT-Area is determined by the topology of the simulation graph. In short, cyclic subgraphs are mapped to VT-Areas and the rationale for this is next explained.

The following observations are made regarding the influence of the synchronisation policy on the performance of DDES. Conservative algorithms, irrespective of the time-keeping mechanisms, generally have lower run-time overheads, but their performance is dependent on the *lookahead*, i.e., the time into the future that a value is known to hold. Optimistic algorithms which use either LVT or GVT time-keeping mechanisms, can potentially infuse more concurrency at run-time by relaxing the strict ordering of evaluation, but at a price: the overheads of completion of non-preemptable tasks and saving of states can attenuate this gain.

The influence, albeit simplified, of the time-keeping mechanism on the performance of DDES is as follows: in acyclic graphs, LVT-based time-keeping mechanisms are more efficient due to the lower communication overhead; and in cyclic graphs, the GVT-based time-keeping schemes are more efficient as they are less prone to fragmenting events. In more detail: GVT-based time-keeping mechanism coupled with an optimistic synchronisation policy (assuming good predictibility) is better suited for cyclic graphs. On the other hand, the LVT-based time-keeping mechanism are unable to exploit predictibility due to event fragmentation. For instance, for an event received at time, t_{now}, at an LP, a committed result needs to be passed round the cycle, before another result can be committed for time t_{next}. The amount of fragmentation depends on the relationship between the inter-arrival time for events and their execution time in an LP. As in acyclic graphs, event fragmentation does not hamper LVT-based time-keeping mechanisms as they are better suited due to their lower communication overheads.

2. Related Work

Jha and Bagrodia [JhaB94] had proposed the idea of a combination of a Global Control Mechanism (GCM), which is based either on null messages or GVT, and a Local Control Mechanism (LCM) which can be either conservative or optimistic. Each LP can be configured to use either LCM, and the GCM is used over the entire system. Unlike the AVT algorithm presented in this paper, there is no notion of multiple areas of virtual time. Hamnes and Tripathi [HamnesT94] adapt the degree of optimism on a per-LP and per-channel basis and use GVT as the only Global Control Mechanism.

Wood and Turner [WoodT94] address the inefficiency of using a null-message time-keeping mechanism for models with cycles, in which the LPs use a local clock and are conservative. In order to distinguish which outputs should transmit carrier-null-messages (as opposed to ordinary null-messages), the cyclic structure of the graph is determined before run-time, and the cyclic and non-cyclic outputs are marked.

Rajaei et al [RajaeiAT93] use Time Warp to synchronise between clusters of LPs, where each cluster is allocated to a processor and runs a sequential simulation algorithm. Avril and Tropper [AvrilT95] divide the simulation model into clusters each with its own Cluster Virtual Time (CVT). Each cluster has one designated input and output process in addition to the LPs of the cluster. The input process keeps track of the Input Virtual Time (IVT), and the output process records an Output Virtual Time (OVT). The output process will only send events to other clusters once the CVT reaches the timestamp of the respective events. A conservative time window (CTW) algorithm is used for synchronisation between clusters.

In contrast, the AVT-based approach does not have designated input/output nodes per cluster, and an LVT-based protocol is used between AVT regions, which can operate either conservatively or optimistically.

3. The AVT Synchronisation Algorithm

Each LP in the AVT algorithm can be configured to use any combination of the modes of synchronisation mentioned in the previous section. In the current instantiation of the algorithm, a GVT node has bounded optimism and the LVT node is conservative. However, the GVT and LVT nodes could just as well have either limited or unlimited optimism [ArvindS92].

Each cyclic area of the simulation graph is assigned to a Virtual Time Area (VT-Area), in which a GVT-style time-keeping scheme is used.

3.1. Definitions

LVT: Local Virtual Time – the time up to when the inputs are committed.

LOT: Local Optimistic Time – the time up to when an LP has speculated based on its input

Event-Message is of the form: (t_{start}, t_{end}, $t_{end_{opt}}$, $data$), where:

t_{start} is the start-time of an event
t_{end} is the time until when the event is committed to hold
$t_{end_{opt}}$ is the time until when an event is presumed to hold based on the input
$data$ is the value of the event.

Event-messages can be either: *committed*, i.e., $t_{start} < t_{end}$, and $t_{end} = t_{end_{opt}}$; *uncommitted*, i.e., $t_{start} = t_{end}$, and $t_{end} < t_{end_{opt}}$; or *partially committed*, i.e., $t_{start} < t_{end} < t_{end_{opt}}$.

AVT: Area Virtual Time is defined as the minimum LOT of the members of the VT-Area and the minimum ($t_{end_{opt}}$) of all the messages in transit in the VT-Area.

AVT-Message is used to communicate between the LPs and the AVT-manager, and is of the form (LVT, AVT) and defined as:

LVT – the sending LP's LVT, or empty when sent by the AVT-manager

AVT – the sending LP's LOT, or the updated AVT sent by the AVT-manager

Delay (δ) – the virtual execution time of an LP.

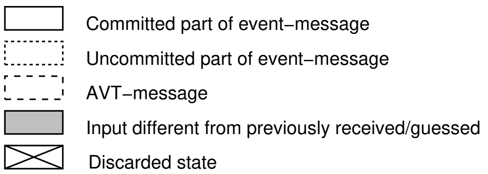
$\mathbf{min(t_{end}^{in})}$ – denotes the minimum t_{end} of all inputs to an LP

$\mathbf{min(t_{end}^{in_{LVT}})}$ – denotes the minimum t_{end} of all inputs to an LP from predecessors outside the VT-Area.

$\mathbf{min(t_{end}^{in_{AVT}})}$ – denotes the minimum t_{end} of all inputs to an LP from predecessors inside the VT-Area.

Predecessor (Successor) of an LP, A, is any other LP from which A receives its input (to which A sends its output).

Graphic Symbols: The height of a box corresponds to the interval between t_{start} and t_{end} (or $t_{end_{opt}}$).



3.2. Description of the AVT Algorithm

The instantiation of the AVT algorithm, as described in this section, assumes fixed connections between the LPs, and that the results of the evaluation are committed before event-messages are sent to the neighbouring LPs.

The format of the event-messages are similar to those used by Composite ELSA, i.e., a time interval defines the period for which a value is guaranteed to hold. The AVT messages have an additional time-stamp to denote the time until when the value is presumed to hold optimistically. The LVT nodes employ a conservative synchronisation policy, while the GVT nodes in the VT-Area employ an optimistic one. The optimism is limited by the look-ahead in the event-messages from outwith the VT-Area. When viewed from beyond, a VT-Area behaves as if it were a conservative LVT node, albeit a conglomerate one. The VT-Areas never emit an uncommitted event-message, although this is not a fundamental restriction: nodes in the VT-Areas (and indeed any LVT node) can range from conservative, to locally or globally optimistic - similar to nodes in Composite ELSA.

The AVT algorithm can be applied to LPs which can be synchronised either locally (LVT-node) or globally (GVT-node), or both (hybrid-node). Consider the logical processes in Figure 1

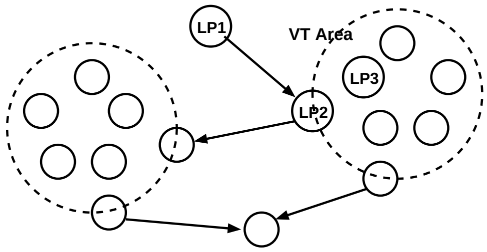


Figure 1. Simulation Graph

LP1 is an LVT-node. It can either be a conservative node (corresponding to a CMB or conservative ELSA node), or an optimistic node (corresponding to an optimistic ELSA node).

LP2 is a hybrid-node, i.e. it behaves both as an LVT- and as a GVT-node.

LP3 is a GVT-node. It can either execute either optimistically or conservatively.

3.2.1 The LVT-node

In the instantiation of the AVT algorithm described in this paper, the LVT nodes synchronise conservatively. The node is ready to progress its time once all the end-times of the input event are greater than its LVT, i.e., $min(t_{end}^{in}) > LVT$ (Figure 2). This node sends the following event to its successors:

$$(t_{start} = LVT + \delta, t_{end} = min(t_{end}^{in}) + \delta,$$
$$t_{end_{opt}} = t_{end}, f(input))$$

Its LVT is advanced to the minimum of the end-times of the input (i.e., $min(t_{end}^{in})$), and any state prior to the LVT is discarded.

3.2.2 The GVT-node

A GVT-node is ready to compute once all its input-queues contain unprocessed events (Figure 3), i.e.,

$$min(t_{end}^{in_{AVT}}) > LVT \vee min(t_{end_{opt}}^{in}) > LOT$$

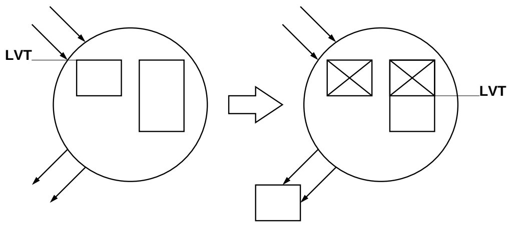


Figure 2. LVT-node in conservative mode

The following event is sent to its successors in the VT-Area:

$$(t_{start} = LVT + \delta, t_{end} = min(t_{end}^{in}) + \delta,$$
$$t_{end_{opt}} = min(t_{end_{opt}}^{in}), f(input))$$

The LVT is advanced to $min(t_{end}^{in})$, the LOT to $min(t_{end_{opt}}^{in})$, and an update AVT-message, (LVT, LOT), is sent to the AVT-manager.

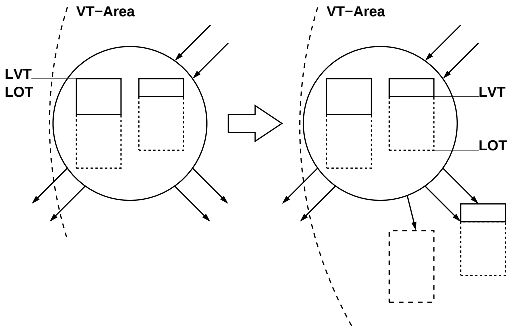


Figure 3. Computing and updating AVT in a GVT-node

A GVT-node is ready to progress conservatively if the AVT has been updated (Figure 4). The following event is sent to all its successors outside the VT-Area:

$$(t_{start} = LVT + \delta, t_{end} = AVT + \delta,$$
$$t_{end_{opt}} = t_{end}, f(input))$$

LVT is advanced to AVT, and all the states before the LVT are discarded.

3.2.3 The Hybrid-node

Hybrid nodes have split personalities: they appear as GVT/optimistic nodes to those inside the VT-Area, and as LVT/conservative nodes to those outside. When the committed inputs from outwith the VT-area are available, the hybrid node guesses the values for others from within the VT-area, evaluates based on all the inputs (both guessed and committed), and fires optimistically, but only to its neighbours within the VT-area.

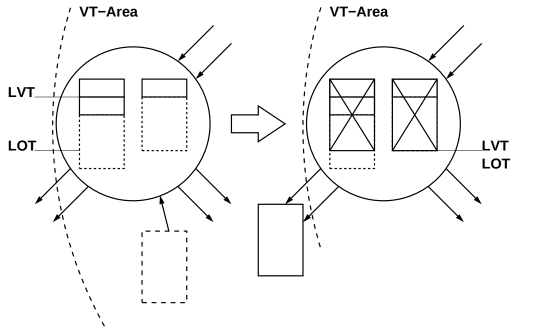


Figure 4. GVT-node progressing conservatively

The dual behaviour of the hybrid-node is described as follows:

- Once all the input-queues to the LVT part of the hybrid-node contain events, i.e.,

  $$min(t_{end}^{in_{LVT}}) > min(t_{end}^{in_{AVT}}) \wedge$$
  $$min(t_{end}^{in_{LVT}}) > LOT$$

  the LP fires optimistically (Figure 5). The following event is generated, which is stored in the LPs output-queue, and is sent to the LP's successors in the VT-Area.

  $$(t_{start} = LVT + \delta, t_{end} = min(t_{end}^{in}) + \delta),$$
  $$t_{end_{opt}} = min(t_{end}^{in_{LVT}}), f(input))$$

  The LOT is advanced to $min(t_{end}^{in_{LVT}})$.

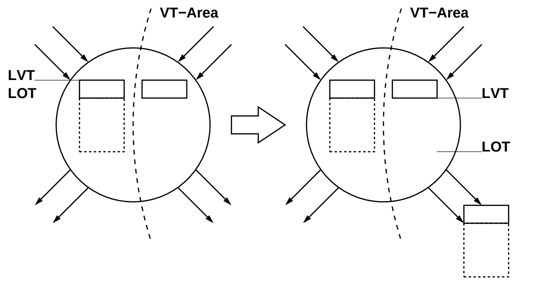


Figure 5. Speculation in a hybrid-node

- An input received from within the VT-Area is checked against the speculated value. In the event of a mis-speculation, the LP is rolled back to a safe state and the correct result is resent to its successors within the VT-Area. Otherwise, an update AVT-message is sent to the AVT-manager. This is described in more detail below:

  1. When the speculation was correct:

(a) The input is committed up to the LOT: $min(t_{end}^{in}) \geq LOT$ results in a committed output being sent to the successors outside the VT-Area; the LVT is advanced to $min(t_{end}^{in})$; and the AVT-message, (LVT, LVT), is sent to the AVT-Manager.

(b) If the input is *not* committed up to LOT, i.e. $min(t_{end}^{in}) < LOT$, then an update AVT-message, (LVT, LOT), is sent to the AVT-manager (Figure 6).

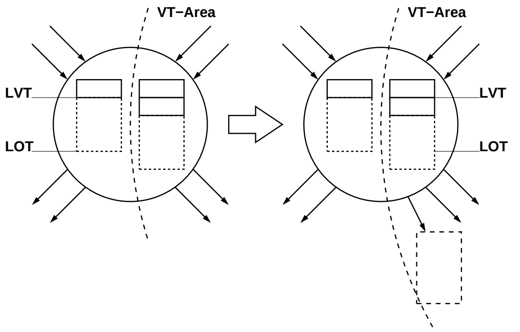


Figure 6. Update in a hybrid-node

2. Should the received input be different from the speculated one, then the LOT is rolled back to LVT, or the result is computed if it had not already been done so, and compared with the speculated one. The following three cases arise:

(a) If there is no previous result, then the following event is sent to the successors:

$$(t_{start} = LVT + \delta, \\ t_{end} = min(t_{end}^{in}) + \delta, \\ t_{end_{opt}} = min(t_{end}^{in_{LVT}}) + \delta, \\ f(input))$$

and, the following update AVT-message is sent to the AVT-manager:

$$(LVT = LVT, AVT = min(t_{end}^{in_{LVT}}))$$

(b) If the recomputed result is unchanged, then the same event and AVT-message (as in Case 1) is sent.

(c) Should the result differ from the one propagated (Figure 7), then the correct result is sent to its successors in the VT-Area, in the following event:

$$(t_{start} = LVT + \delta, \\ t_{end} = min(t_{end}^{in}) + \delta, \\ t_{end_{opt}} = min(t_{end}^{in_{LVT}}) + \delta, \\ f(input))$$

The following event is sent to the successors outside the VT-Area, with the LVT being advanced to $min(t_{end}^{in})$, and any state before LVT being discarded:

$$(t_{start} = LVT + \delta, \\ t_{end} = min(t_{end}^{in}) + \delta, \\ t_{end_{opt}} = t_{end}, \\ f(input))$$

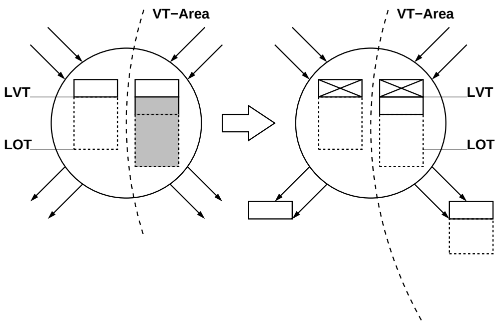


Figure 7. Roll-back in a hybrid-node

- Once an LP receives an update AVT-message from the AVT-manager, it fires conservatively (Figure 8). The committed result is sent to all the successors outside the VT-Area.

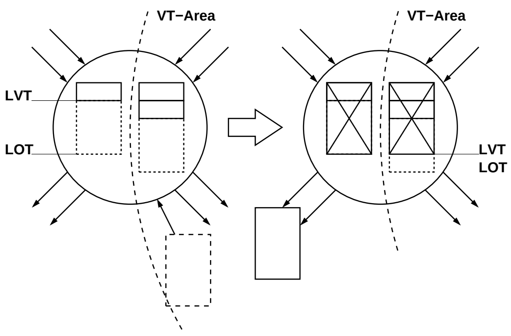


Figure 8. Hybrid-node progression based on AVT

A hybrid node progresses its LVT based on the AVT when the following conditions hold: if the updated AVT is larger than the minimum end-time of all the committed inputs, i.e., $AVT > min(t_{end}^{in})$, then the following event is sent to its successors outside the VT-Area

$$(t_{start} = LVT + \delta, t_{end} = AVT + \delta, \\ t_{end_{opt}} = t_{end}, f(input))$$

The LVT is advanced to the AVT, and any state before the LVT is discarded.

3.2.4 The AVT-manager

The behaviour of the AVT-manager is next summarised.

The AVT-manager computes the AVT as the minimum of all the estimates received by the GVT and hybrid nodes in the VT-Area, and the minimum, committed time-stamp of any event-message in transit within the VT-area. If AVT can be advanced, then the AVT-manager sends the updated AVT to all the nodes in the VT-area. All events up to the AVT are then committed and sent to neighbours outwith the VT-area. The LVT is brought forward to the AVT and all state information before the AVT is discarded.

When the AVT-manager receives an AVT-message, (LVT, LOT), from an LP within its assigned VT-Area, then it is stored and the AVT is re-estimated as the minimum of the LOT of all the LPs in the VT-Area and $t_{end_{opt}}$ of all events in transit within the VT-Area. If the new estimate is greater than the previous one, then the AVT-manager notifies all the LPs in the VT-Area whose LVT is less than the AVT.

4. The Simulation Environment

The behaviour of the AVT algorithm was modelled in C++ and simulated in a sequential event-driven simulator (also written in C++). The simulator also models a distributed system connected by a network. All the significant operations, such as computing results and sending event messages, are assigned costs.

The network model assumes a fixed cost and a minimum fixed delay for each message. The *bandwidth* of the network determines the number of messages that can be transmitted over the network at any time. Messages are delayed until the network bandwidth becomes available. Thus the network model neither takes into account the relative sizes of messages nor the effects of thrashing, although congestion is accounted for.

We distinguish between two notions of virtual time. The virtual time according to the AVT algorithm is called *virtual time*. The virtual time according to the simulated distributed system on which the AVT algorithm is run is referred to as *simulated real time*.

The parameters of interest are:

System Parameters

Number of Processors

Communication Bandwidth This determines the number of messages the network can transmit at any time. Bandwidth can be made unlimited, in which case every message is delayed by a fixed amount of *simulated real time*.

Event Propagation Delay The amount of *simulated real time* required to send an event from one processor to another.

AVT Propagation Delay The number of units of *simulated real time* needed to send an AVT-message.

Input Delay The *simulated real time* required to read an input value from an external source.

End of *simulated real time* The number of units of *simulated real time* for which the distributed system is simulated.

Algorithm Parameters

Mode One parameter each from the two sets - {LVT, AVT, GVT} and {Conservative, Optimistic} - is selected.

AVT Processing Delay The amount of *simulated real time* needed to process an AVT update at the AVT-manager.

Model Parameters

Event Processing Delay The amount of *simulated real time* needed to compute a result in an LP.

Input Interval The amount of virtual time between the arrival of two input events to the model.

Delay The virtual time between receiving an event in an LP and producing an output event.

Output Change Probability A value between 0% and 100% which gives the probability of the output of an LP changing should one of the inputs change

5. Results

The benchmark under simulation has cyclic (2 LPs) and acyclic parts (3 LPs). The model was run under all the different combinations of the following parameters:

Number of Processors {1, 2, 5}

Communication Bandwidth 1

Event Propagation Delay {1, 3, 10, 32, 100} units of *simulated real time*

AVT Propagation Delay Same as *Event Propagation Delay*

Input Delay 0 units of *simulated real time*

End of *Simulated Real Time* 10000 units

Mode {LVT, AVT, GVT} and *Conservative* (i.e., the VT-area as a unit operates conservatively although the LPs within the VT-area may progress optimistically).

AVT Processing Delay 0 units of *simulated real time*

Event Processing Delay 100 units of *simulated real time*

Input Interval 100 units of virtual time.

Delay {1, 3, 10, 32, 100} units of virtual time.

Output Change Probability {0%, 25%, 50%, 75%, 100%}

The graph in Figure 9 compares the progress in Virtual Time for the three time-keeping schemes (y-axis) while changing the simulation parameters along four dimensions - Delay, Event Propagation Delay (denoted as "Message Delay" in the graph), Number of Processors, and Output Change Probability ("Change Prob." in the graph). In the two-processor case, the cyclic part was mapped to one processor and the acyclic part to the other. The graphs in Figures 10–13 show details of Figure 9.

As the value of the *Delay* is incremented, the progress of the virtual time improves, and in the extreme, to a spike, when the *Delay* is the same as the *Input Interval*. In this case the lookahead is right up to the arrival of the next event, resulting in no event fragmentation.

We also observe that in the case of mapping to two processors, the performance of the AVT algorithm is more stable over changes to the parameters, compared to the GVT time-keeping scheme, which is hit by network delays, and to the LVT time-keeping mechanism, which only performs well in the cases of good look-ahead (aka Delay (δ)). In the mapping to five processors, it is evident that the AVT algorithm degrades more gracefully with increases in network delays.

The AVT algorithm maintains this superiority over the other two time-keeping mechanisms for different *Output Change Probability* values.

The results demonstrate that the AVT algorithm performs better or just as well as the other two mechanisms for simulation models containing a mixture of cyclic and acyclic parts.

6. Conclusions and Future Work

We have proposed a novel synchronisation scheme called the Area Virtual Time (AVT) algorithm which uses the apt time-keeping mechanism for simulation graphs containing a combination of cyclic and acyclic parts.

The cyclic part of the graph is mapped to a GVT-based time-keeping mechanism, and the acyclic part to an LVT-based time-keeping mechanism.

Our results demonstrate that the AVT algorithm progresses simulation time more efficiently than either the GVT- or LVT-based time-keeping schemes, and the algorithm is less sensitive to variations in key model and communication parameters - a desirable property in a distributed computation.

The AVT algorithm is currently being extended to simulate efficiently simulation graphs in which connections between nodes change dynamically.

References

[**ArvindS92**] D. K. Arvind and C. Smart. Hierarchical Parallel Discrete Event Simulation in Composite ELSA. In *6th Workshop on Parallel and Distributed Simulation (PADS92)*, Pages 147–156, 1992.

[**AvrilT95**] H. Avril and C. Tropper. Clustered time warp and logic simulation. *Proceedings of the 9th workshop on Parallel and distributed simulation*, 1995, Pages 112–119

[**Bryant81**] R. E. Bryant. A switchlevel model and simulator for MOS digital systems. *IEEE Transactions on Computers*, C-33(2):160177, Feb. 1981.

[**ChandyM79**] K. Chandy and J. Misra. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, 1979.

[**HamnesT94**] Donald O. Hamnes and Anand Tripathi. Investigations in adaptive distributed simulation. *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*, 1994, Pages 20–23

[**JeffersonS85**] D. Jefferson and H. Sowizral. Fast Concurrent Simulation Using the Time Warp Mechanism. In *Proceedings of the Conference on Distributed Simulation*, pages 63–69, July 1985.

[**JhaB94**] Vikas Jha and Rajive L. Bagrodia. A unified framework for conservative and optimistic distributed simulation. *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*, 1994, Pages 12–19

[**RajaeiAT93**] Hassan Rajaei, Rassul Ayani and Lars-Erik Thorelli. The local Time Warp approach to parallel simulation. *Proceedings of the 1993 Workshop on Parallel and Distributed Simulation*, 1993, Pages 119–126

[**WoodT94**] Kenneth R. Wood and Stephen J. Turner. A generalized carrier-null method for conservative parallel simulation. *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*, 1994, Pages 50–57

Abbreviations in Figures 10–13:

EPD = Event Propagation Delay (message delay)

NP = Number of Processors

OCP = Output Change Probability

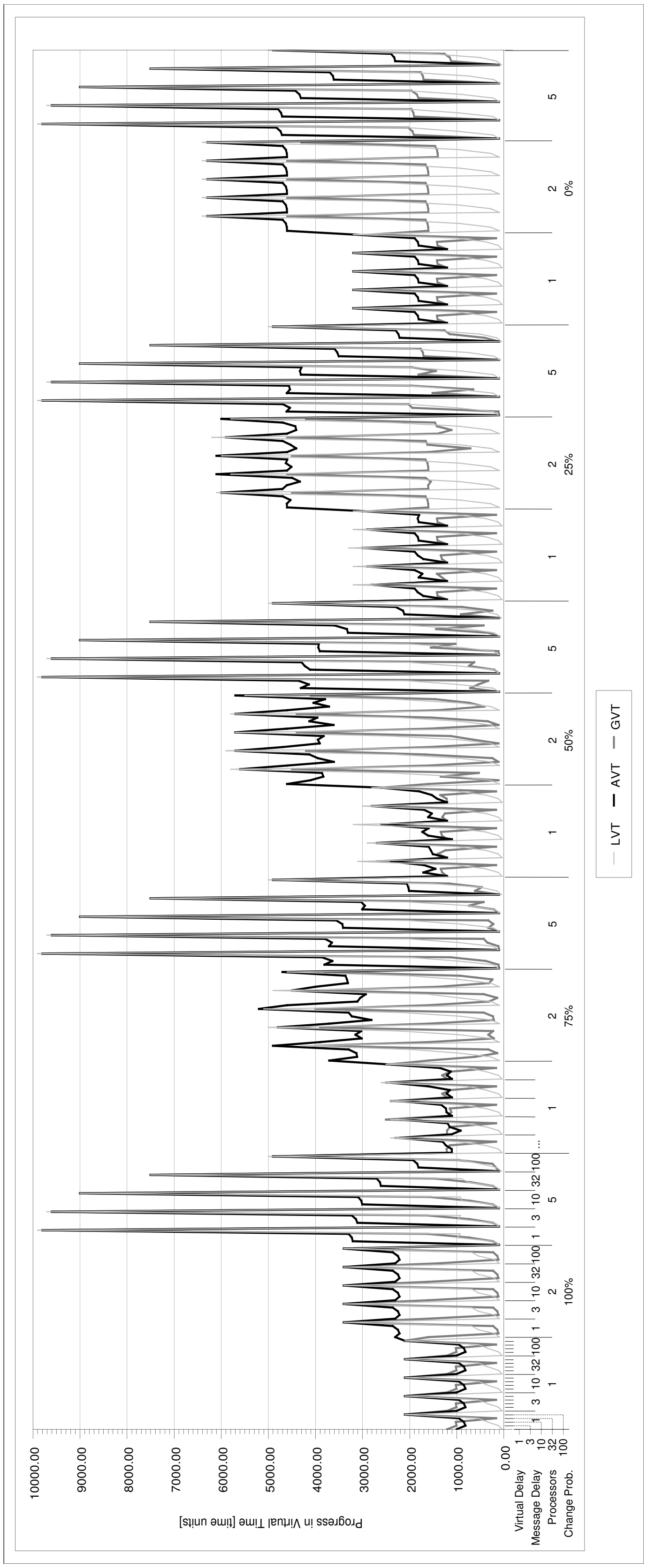


Figure 9. Progress of Virtual Time for 10000 Units of Simulated Real Time

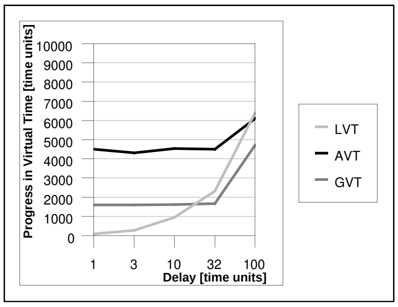


Figure 10. Progress for Delay=1, 3, 10, 32, 100; EPD=1; NP=2; OCP=75%

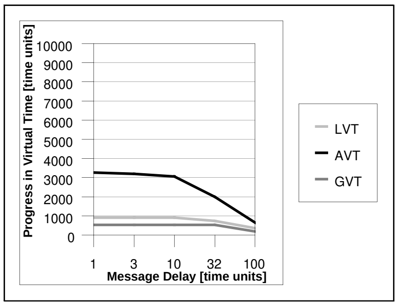


Figure 11. Progress for Delay=32; EPD=1, 3, 10, 32, 100; NP=5; OCP=0%

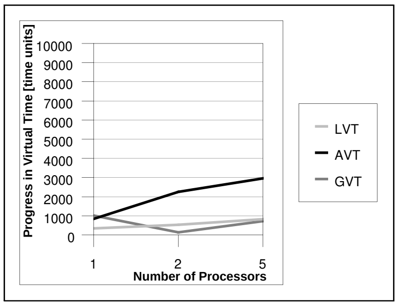


Figure 12. Progress for Delay=10; EPD=10; NP=1, 2, 5; OCP=0%

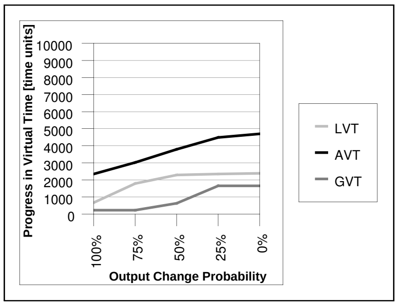


Figure 13. Progress for Delay=32; EPD=32; NP=2; OCP=0%, 25%, 50%, 75%, 100%