Conservative Synchronization Methods for Parallel DEVS and Cell-DEVS

Shafagh Jafer, Gabriel Wainer
Dept. of Systems and Computer Engineering
Carleton University, Centre of Visualization and Simulation (V-Sim)
1125 Colonel By Dr. Ottawa, ON, Canada.
{sjafer, gwainer}@sce.carleton.ca

Keywords: Discrete-event simulation, DEVS, conservative synchronization, null message, centralized synchronization.

Abstract
We present three conservative synchronization mechanisms for parallel DEVS and Cell-DEVS. The protocols are based on the classical Chandy-Misra-Bryant null message mechanism with deadlock avoidance. Our protocols provide a novel DEVS-based conservative approach that is deadlock-free, and extracts the lookahead information from the model's specification. The protocols are integrated into the CD++ simulation toolkit, providing a conservative simulator (named CCD++) for running large-scale DEVS and Cell-DEVS models in parallel and distributed fashion. We provide a comparative study of these protocols by investigating different performance metrics including: total execution time, blocked time, memory consumption, total number of positive and null event, as well as null message ratio, showing how CCD++ provides considerable speedups, and its ability for simulating large DEVS-based models.

1. INTRODUCTION

Discrete-event modeling and simulation (M&S) has been used to study complex systems in a broad array of domains. Among the existing simulation techniques, DEVS (Discrete Event System Specification) [1] formalism provides a discrete-event M&S approach that allows construction of hierarchical models in a modular manner. DEVS is a sound formal framework based on generic dynamic systems concepts that allows model reuse, and reduction in development and testing time due to its hierarchical approach in constructing models. The Cell-DEVS [2] formalism expands DEVS to describe n-dimensional cell spaces as discrete event models, where each cell is represented as a DEVS model with explicit timing constructions.

Parallel and distributed computing has become the technology of choice to speed up large-scale simulations and to allow geographically distributed simulations. Parallel DEVS (P-DEVS) introduced a mechanism for handling simultaneous events, allowing for efficient execution of parallel models [3]. Both Cell-DEVS and P-DEVS have been implemented in CD++ [4], a M&S environment programmed in C++. Several versions of the tool have been built in order to run large-scale simulations in parallel and distributed fashion. PCD++ [5] is one of them, which allows optimistic simulation of DEVS and Cell-DEVS models based on the WARPED kernel [6]. In [7] we introduced a Conservative DEVS protocol, and built CCD++, the first purely conservative simulator for Cell-DEVS. The protocol is based on the classical Chandy-Misra-Bryant (CMB) [8-9] null message mechanism with deadlock avoidance. We refer to this protocol as the Lower-Bound Time Stamp mechanism (LBTS), the way used to compute the next global virtual time. To reduce the number of null messages, we later proposed the Global Lookahead Management (GLM) protocol [10], which maintains a central lookahead manager (LM) to identify the global minimum lookahead of the system.

We are interested in analyzing different conservative protocols in simulating large-scale DEVS-based models. Here, we first introduce a new conservative protocol, the CMB Conservative DEVS, which is similar to LBTS and it differs in the null message distribution strategy. Then, we provide a thorough performance analysis of the three conservative protocols (LBTS, GLM, and CMB) by conducting a variety of simulations. We provide a comparative study of these protocols by investigating different performance metrics including: total execution time, blocked time, memory consumption, total number of positive and null event, as well as null message ratio, showing how CCD++ provides considerable speedups, and its ability for simulating large DEVS-based models.

2. RELATED WORK

There are a number of parallel DEVS M&S toolkits including: DEVS-C++ [11], DEVS/CORBA [12], DEVSCluster [13], DEVS/P2P [14], and DEVS/RMI [15]. Aside, much work has been done using the synchronization mechanisms offered by HLA [16]. DEVS-HLA simulators have been reported in [17-19]. In [20], a new simulation algorithm for efficient distributed simulation of P-DEVS models is presented. The algorithm makes use of Java threads and performs sequential execution among the entities on each computing node while the simulation is distributed over remote nodes. We are interested in CMB-based conservative simulation by using null messages and lookahead information to synchronize among participating nodes.

The Optimistic DEVS protocol [5], and its extension, the Lightweight Time Warp protocol [21], were the first pure optimistic mechanisms allowing parallel execution of Cell-DEVS systems. Although these two protocols try to reduce the overhead of the optimistic algorithm, issues such as numerous memory consumption and large number of state savings and rollbacks remain. This is especially apparent when the number of participating nodes increases; resulting in cascaded rollbacks. In order to analyze the limitations of the optimistic DEVS protocols and the parallel execution of DEVS and Cell-DEVS, we developed DEVS-based conservative protocols. The first one was the Conservative DEVS (LBTS) protocol [7] based on the idea of lower-bound time-stamp and the classical null message protocol of Chandy-Misra-Bryant. To reduce the number of null messages, we later proposed the Global Lookahead Management (GLM) protocol [10], which maintains a centralized synchronizer that deals with null message distribution and global time advancement. Here, we propose another conservative protocol based on the classical CMB synchronization named DEVS CMB. This protocol differs from our original LBTS in the way null messages are distributed among neighboring nodes. The goal is to reduce the number of null messages compared to our LBTS protocol.

3. CONSERVATIVE SIMULATION IN CCD++

CCD++ is the first purely conservative simulator for running Cell-DEVS simulations in parallel and distributed fashion. The simulator is built on top of the WARPED kernel [6], which provides services for defining processes (simulation objects), scheduling, memory, file, event, communication, and time management. Simulation objects on a physical processor are grouped into a Logical Process (LP), and communicate through Message Passing Interface (MPI).

To reduce communication overhead, CCD++ adopts a flat structure that creates a *Node Coordinator* (NC), a *Flat Coordinator* (FC), and a set of *Simulators* on each node. Doing so eliminates intermediary coordinators in the LP hierarchy, reducing communication costs. The NC is a local central controller and the final destination of inter-node messages, whereas the FC routes messages between its child *Simulators* and the parent NC, as seen in Figure 1.

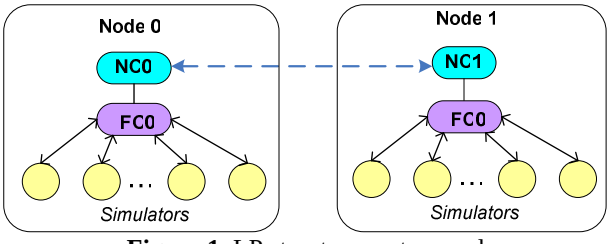

Figure 1. LP structure on two nodes

Six types of events are defined to execute the simulation in a message-driven fashion: *External* (x, t) and *output* (y, t) messages encode the input and output data; *initialization* (I, t), *collect* (@, t), *internal* (*, t), and *done* (D, t) control the execution of events at each virtual time [7].

Our conservative protocols (LBTS, GLM, and CMB) are implemented at the NC. Processes communicate only through messaging with their neighbors; there are no shared variables and no central process for message routing or scheduling. Although each LP has its own Local Virtual Time (LVT), no events are received at virtual past time. Synchronization is maintained through null messages carrying on lookahead information. The NC on each node is the central synchronizer for driving the simulation on that node. The focus on each of the protocols is on computing the lookahead values and distributing them via null messages, and deciding when to suspend or resume the LP. The NC is responsible for lookahead calculation, null message distribution, suspending the LP, receiving null messages from other LPs while the LP is blocked, and resuming the LP when all remote null messages are received. The NC drives the simulation at the LP, while *FC*, *Simulator*, etc. are unaware of the underlying synchronization mechanism.

Although our conservative protocols exploit similar parallelism level as in the classical P-DEVS simulation protocol, however, unlike P-DEVS, we do not have a global synchronizer advancing the global time. Moreover, our protocols require smaller number of messages when calculating the next time advance of the simulation [7]. In P-DEVS protocol there is a global coordinator asking all atomic components to send their next state change values. Although, P-DEVS is a risk-free optimistic protocol (not even local rollbacks occur), but it only exploits parallelism in the simultaneous occurrence of *internal* events among many components.

4. CONSERVATIVE PROTOCOLS FOR DEVS

In this section, we provide a brief overview of each of our conservative protocols, highlighting their differences.

4.1. The LBTS Protocol

In LBTS [7], processes communicate only through messages with their neighbors; there are no shared variables and no central process for message routing or process scheduling. Although each LP has its own Local Virtual Time (LVT), no event is received at the virtual past time. The null messages carry lookahead information. The protocol is deadlock-free, as null message cycles cannot occur. At the start of every synchronization phase, each LP computes its lookahead value, which is dynamically extracted from the model specifications, and forwards it to all other LPs. Then the LP suspends and waits for all remote null messages to arrive from other LPs. Once all null messages are received from all LPs participating in the simulation, it resumes and first computes its new LVT based on the lookahead values it received via the remote null messages. This lookahead and LVT computation are described in details in [7]. As we can see, the LVT of every LP at any time is equal to the Lower-

Bound Time Stamp of any unprocessed event among all LPs. The major issue of this protocol is the numerous amounts of null messages that must be distributed at the start of each synchronization phase. Each LP not only sends null messages to its direct neighbors, but also to every other LP to ensure correct computation of the LBTS value. This issue motivated us to revise the null message distribution mechanism by proposing two new protocols, the GLM and the CMB protocols which are discussed next.

4.2. The GLM Protocol

The Global Lookahead Management (GLM) protocol [10] uses the idea of safe processing intervals from the Conservative Time Window [22] algorithm and maintains global synchronization in a fashion similar to the Distributed Snapshot technique [23]. GLM reduces the number of null messages by organizing the conservative execution in such a way that every LP reports its lookahead only to the global manager rather than to every LP. A central *lookahead manager* (LM) is in charge of receiving every LP's lookahead, identifying the global minimum lookahead of the system, and broadcasting it via null messages to all LPs. The sole function of the LM is to detect the suspension phase, and to initiate the resume phase by broadcasting the *global minimum lookahead*. The simulation is divided into cycles of two phases:

(i) ***Parallel Phase***: LPs run simulation until suspension.

(ii) ***Broadcast Phase***: LM broadcasts global minimum lookahead, allowing LPs to advance their LVTs.

The key characteristic of GLM is that it is asynchronous and the central LM is not expected to be a bottleneck since the only message transmissions involving it take place at the end of *Parallel* phase and *Broadcast* phase. In fact, the LM does not carry out any computation and it is only invoked when all LPs are blocked and the simulation is suspended, not introducing any overhead.

4.3. The CMB Protocol

This protocol we introduce here is a variation of LBTS to reduce the number of null messages. The protocol changes the way conservative synchronization is maintained by focusing on null message distribution only among neighboring LPs. An LP only forwards null messages to its direct neighbors as defined by the DEVS translation function. Under this scheme, at the start of every synchronization phase, the LP computes its lookahead similarly to the way it is calculated in LBTS, but the null message is only sent to its neighbors. Then the LP blocks and waits for its neighboring LPs to send their lookahead value via null messages. Once all neighbor null messages are received, the LP computes its new LVT based on the received lookahead values, and starts another lookahead computation and null message distribution round. This process continues until no smaller lookahead value can be received from neighbor LPs later in time. Once the LP has received the smallest possible lookahead value, it computes the new LVT and resumes the simulation. With the CMB protocol, the overall number of null messages is reduced, but the multiple lookahead computation and null message redistribution could have a negative effect on the simulation performance. These effects will be discussed thoroughly in the Performance Evaluation section.

4.4. Comparison of the Conservative Protocols

Figure 2 illustrates the null message distribution strategy for the three conservative protocols. As we can see, they share the following common features and characteristics:

1. They are implemented at the NC; the other DEVS processors are unaware of the underlying synchronization mechanism. The NC is the local controller and drives the simulation on that node. It is responsible for lookahead and LVT computation, LP suspension and resumption, and null message distribution and reception.
2. Lookahead and LVT computations are performed dynamically based on the model's data. The computation formulas are the same for all the three protocols.
3. Lookahead computation is performed after each LVT computation; hence, it is updated and distributed among all remote LPs every time before the LP is suspended. This strategy ensures that the lookahead value of an LP represents the latest LVT update as there is at least one lookahead computation per LVT update. Unlike other conservative algorithms, the modeler does not need to specify the lookahead, which is dynamically extracted by the protocols.
4. Null message distribution occurs before LP suspension, thus, deadlock is strictly avoided. NC only suspends the LP after performing a lookahead computation and propagating it to destination LPs via null messages.

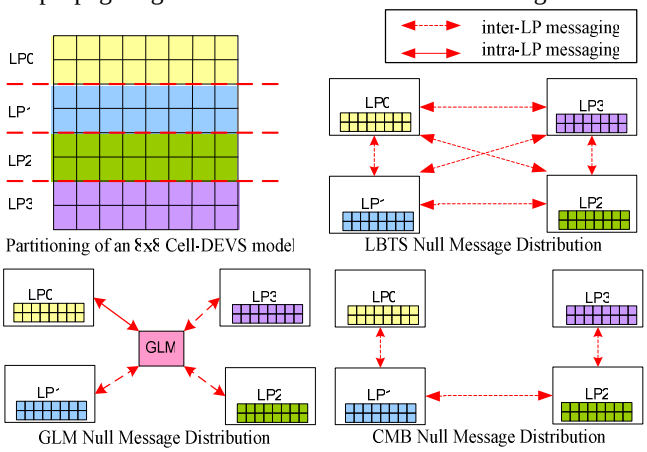

Figure 2. Null message distribution of the protocols

5. PERFORMANCE EVALUATION

To obtain a comparative study of our conservative protocols, we implemented LBTS, GLM, and CMB in CCD++, and conducted extensive tests with each protocol. Tests were carried out on a cluster of 12 compute nodes (dual 3.2 GHz Intel Xeon processors, 1 GB PC2100 266 MHz DDR

RAM) running Linux WS 2.4.21 interconnected through Gigabit Ethernet and MPICH 1.2.6. Table 1 lists the metrics collected in the experiments. The experimental results for each test case were averaged over 10 independent runs to strike a balance between data reliability and testing effort. For the test cases on multiple nodes, the results were also averaged over the participating nodes to obtain a *per-node* evaluation (i.e. BT, MEM, PEV, and NEV represent the corresponding results per one node).

We used three different Cell-DEVS models in our experiments. The first model, called *Fire*, simulates forest fire propagation in a two dimensional cell space based on Rothermel's definition [24]. The second model, named *Watershed*, is a simulation of the environmental influence on hydrological dynamics of water accumulation in a three dimensional cell space [25]. The third model, called *Synth*, is a synthetic model consisting of a grid where cells are initially set to zero and throughout the simulation, they toggle between the value of 0 and 1. Each cell has eight neighbors, which leads to high communications. The purpose of this model is to analyze parallelism with communication-intensive models. The *Fire* model is computation-intensive compared to *Watershed* model, which consists of a 3D cell space that makes it a good candidate for analyzing communication-intensive simulations.

Table 1. Performance metrics

Metrics	Description
T	Total execution time of the simulation (sec)
BT	Total blocked time during the simulation (sec)
MEM	Maximum memory consumption (MB)
PEV	Total number of positive events executed
NEV	Total number of null events executed
NMR	Null message ratio (NEV / PEV)

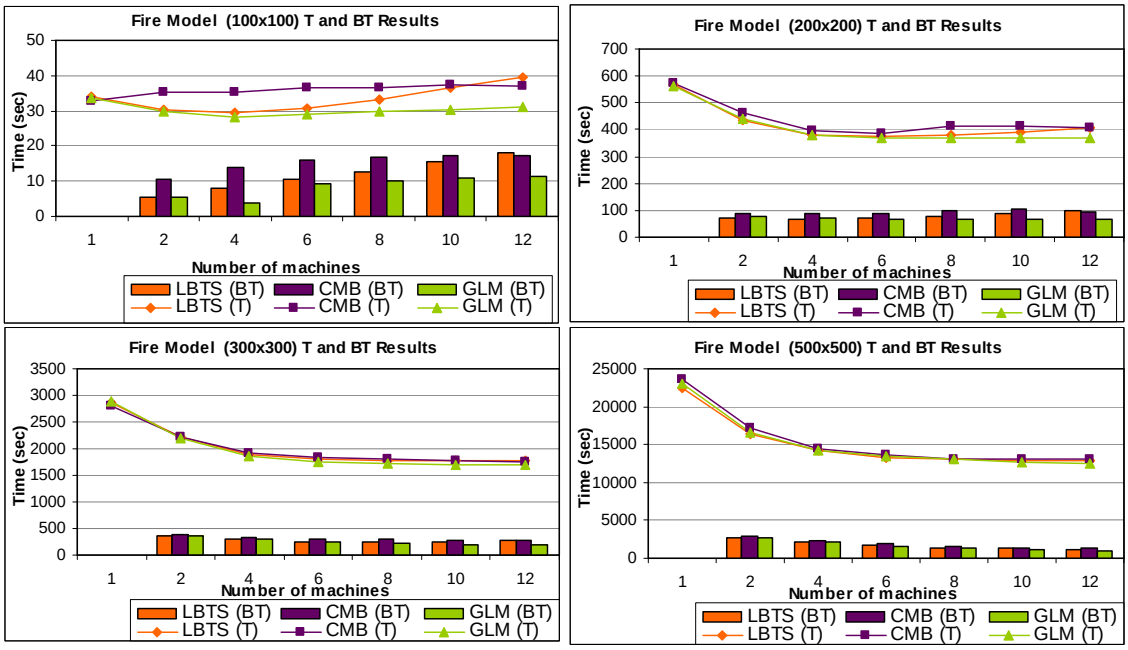

Figure 3. Fire model T and BT results

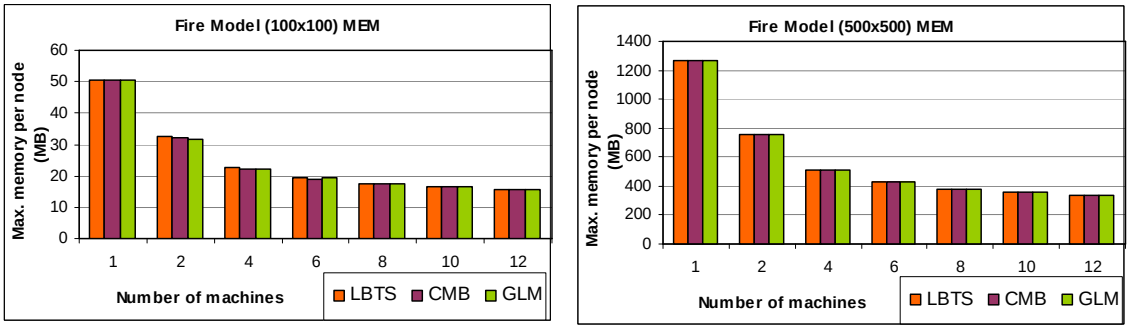

Figure 4. *Fire* model memory consumption results

Figure 3 illustrates the T and BT results for *Fire* model. The LBTS and GLM protocols reduce the execution time when more nodes are participating. However, this is only true until a certain point, where after that adding more nodes do not reduce the execution time. This is due to the overhead of the protocol, where increased number of null messages and blocking times start to have a negative impact on the overall performance. In terms of the BT, GLM produced the smallest results in all cases, while CMB resulted in the largest blocked time values. Although CMB produces less null messages, it ends up with larger total number of null messages and blocked periods compared to LBTS protocol (because its strategy consists of multiple rounds of null message distribution). Memory consumption per node is reduced in the same manner for all different sizes (we only present the results for two sizes due to space limitation) as seen in Figure 4. The maximum memory consumption per node drops considerably as more nodes are engaged for all the three protocols.

The results of the *Watershed* model are given in Figure 5. Since the model is communication-intensive we can see that for all the protocols, the execution time drops as more nodes are engaged. The performance improves even with small sizes (compared to *Fire*). GLM provides the best performance in all cases, and the worst performance is for CMB. In most cases, only the BT of CMB is larger than the T of GLM and LBTS. In all cases, CMB takes longer with 2 nodes compared to 1 node. The large overhead of the protocol overcomes the benefits of parallelism. However, as the number of processors increases, the execution time and the blocked period of CMB starts to drop. For BT, the tests show that, similar to the *Fire* model, GLM has the lowest blocked time; then comes LBTS, and finally CMB. For the *Watershed* model, execution time and BT reduction rate for various sizes of the model were very close. The three protocols have the same performance gain regardless of the size of the model, which is merely due to the numerous events that are distributed throughout the simulation (this 3D model includes a large number of neighbors that must be updated more often). The MEM results are given in Figure 6. As in the *Fire* model, memory consumption per node drops as the number of machines increases. All the three protocols resulted in very close MEM values, showing that the three protocols perform the same in terms of memory consumption.

The T, BT, and MEM for the *Synth* model are shown in Figure 7. This model allows analyzing the performance of each of the protocols when full parallelism takes place. As can be observed from the execution results, for all the protocols, the simulations have benefited from the full parallelism such that the performance continuous to improve as the number of nodes increases.

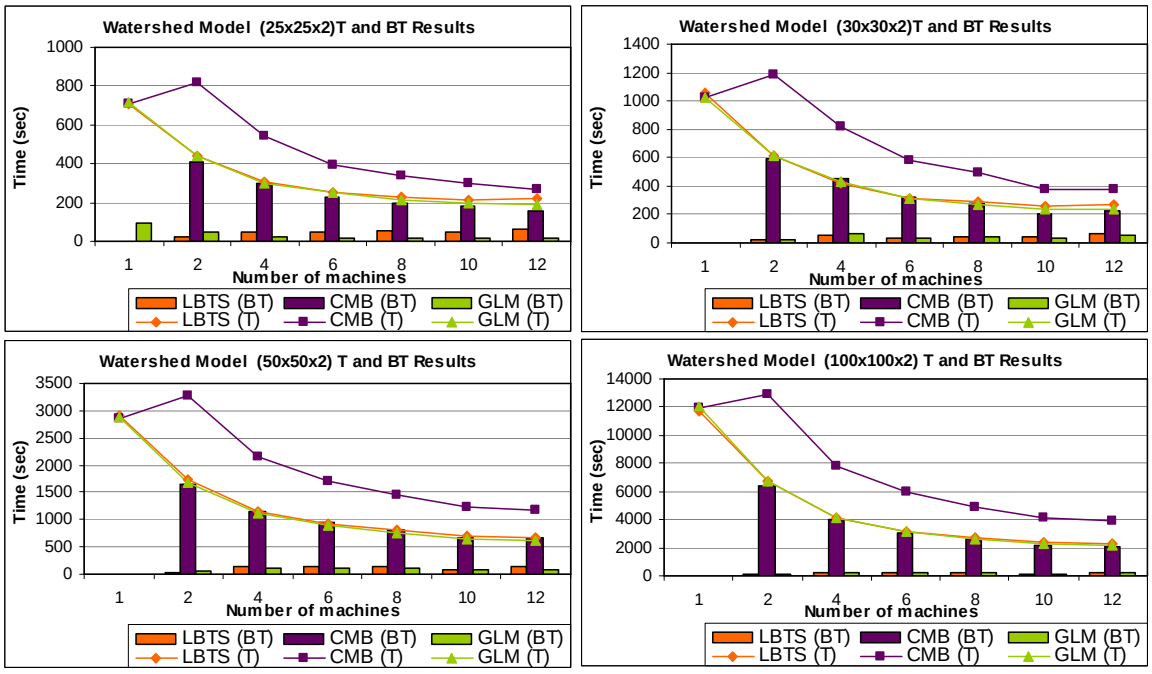

Figure 5. *Watershed* model T and BT results

For GLM and LBTS, the BT value is considerably low compared to the T value in each case. The BT values are still too high with the CMB protocol compared to the other two protocols. As in previous models, the GLM resulted in best performance, while the CMB protocol has the worst results in every scenario. However, due to the nature of the model, the results are overall better than those obtained from the *Watershed* or *Fire* model. As shown by the mem-

ory consumption graph (for the 400x400 size) memory usage per node improves remarkably with the increase of number of machines. The memory consumption is the same for all protocols, and for all the three different sizes of the model, however we only presented the graph of the 400x400 size due to space limitations.

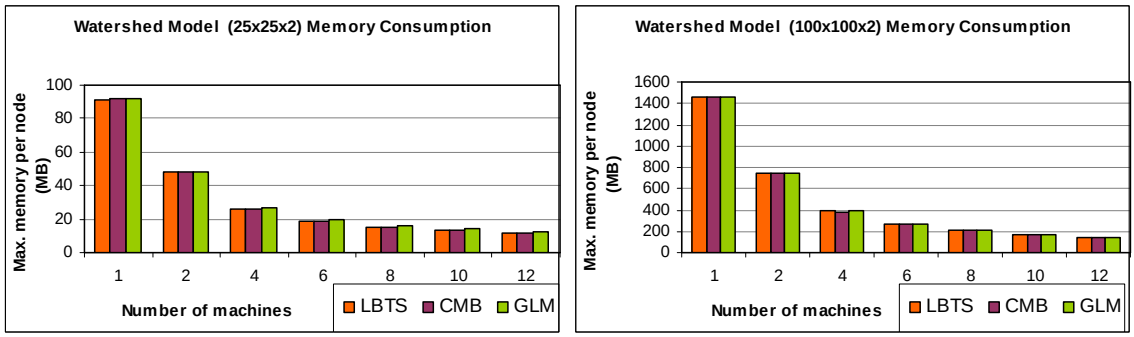

Figure 6. *Watershed* model memory consumption results

We are also interested in investigating the results of the three protocols in terms of the total number of null messages and the null message ratio. Figure 8 shows the NMR (i.e. NEV/PEV) results for various sizes of the *Fire* model. Looking at the GLM graphs, we see that this protocol produces the smallest NMR in all cases. The CMB protocol, compared to the LBTS protocol, produces smaller NMR values after a certain number of participating nodes, which is 4, 6, 6, and 8 nodes for 100x100, 200x200, 300x300, and 500x500 sizes respectively. This behavior is explained by the fact that as the number of machines increases, the synchronization overhead associated with CMB gets smaller than that of produced by the LBTS protocol. Meaning, with smaller number of machines, the total null messages produced by the LBTS protocol are less than the number of null message distribution rounds in CMB, thus resulting in lower NMR compared to the CMB protocol. On the other hand, when more nodes are participating, the total number of null messages that are distributed by the LBTS protocol are much higher than those produced by the CMB protocol, although the CMB protocol causes more synchronization rounds per each synchronization phase when more nodes are engaged.

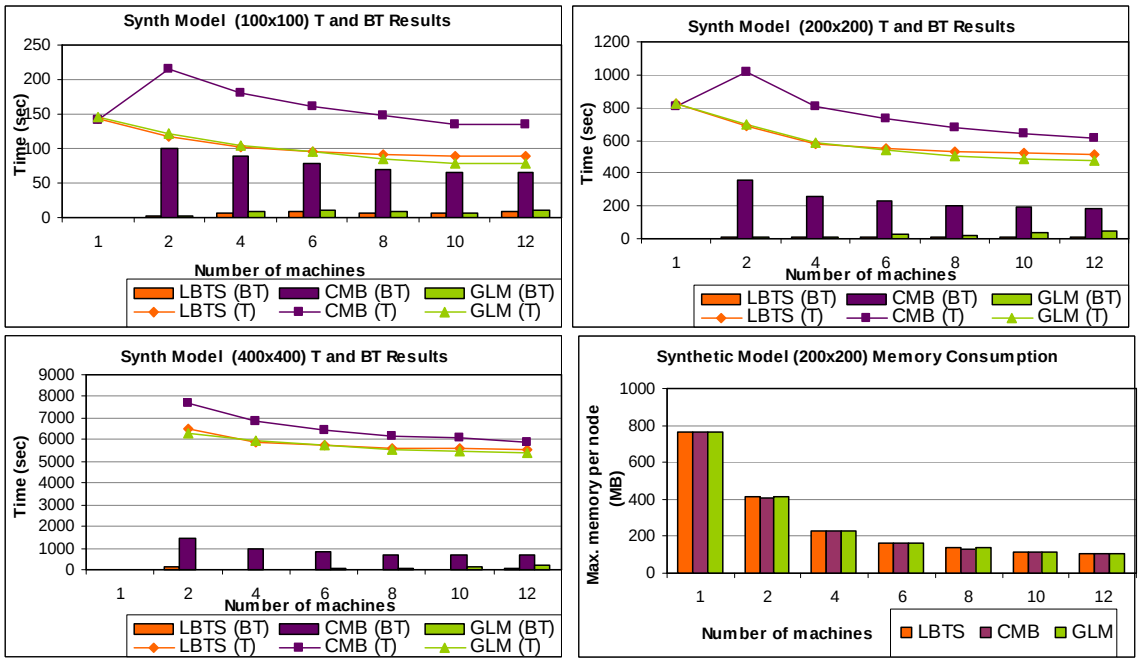

Figure 7. *Synth* model T, BT, and MEM results

As expected, the GLM protocol results in the smallest number of null messages (average NEV per node) in all cases. Similar to the NMR results, the CMB outperforms the LBTS protocol after a certain point, while with smaller number of machines it shows worse results compared to LBTS. The NMR results for the *Watershed* model are illus-

trated in Figure 9. Similar to *Fire* model, the best results are obtained with GLM, and the CMB protocol outperforms the LBTS when more nodes are used.

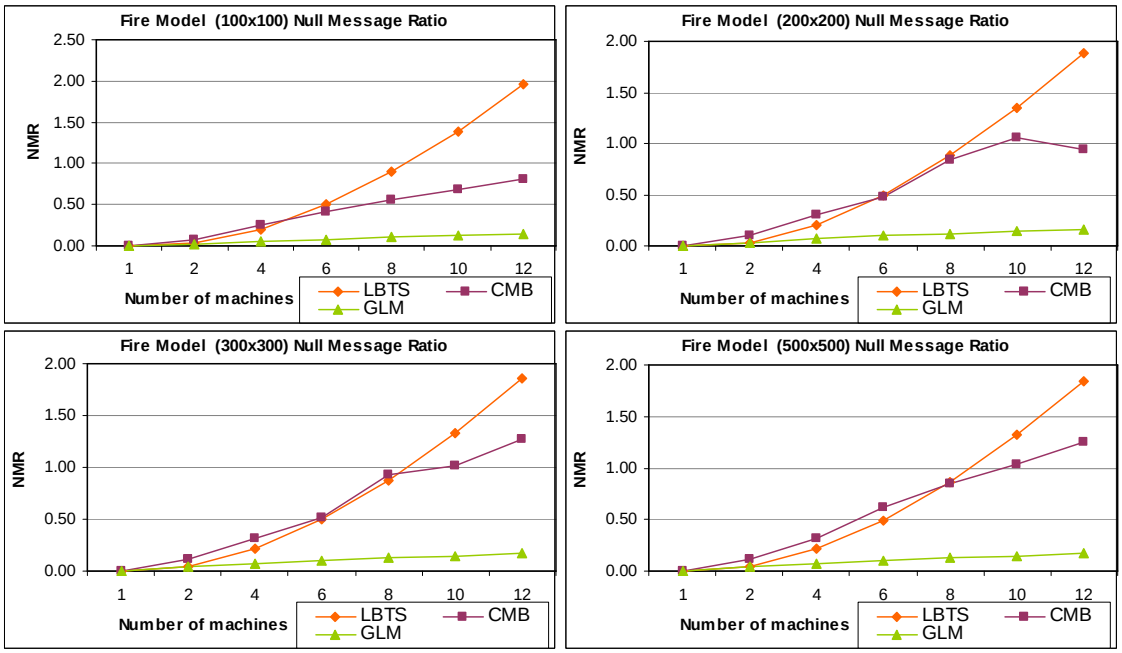

Figure 8. *Fire* model NMR results

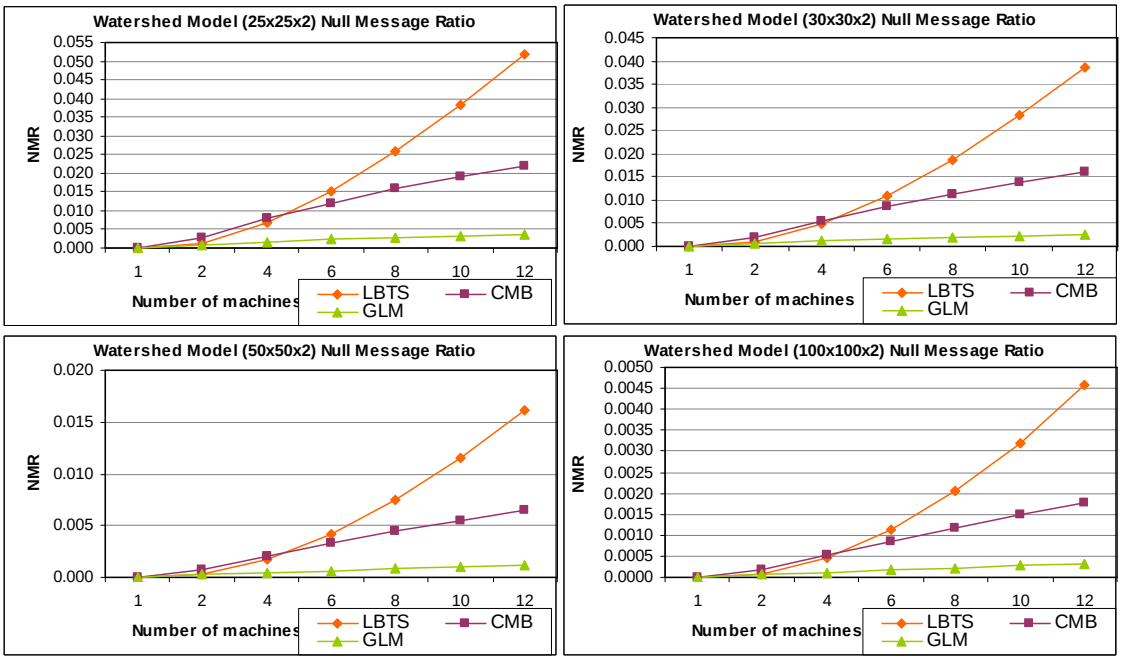

Figure 9. *Watershed* model NMR results

6. CONCLUSIONS

We presented a comparative study of three conservative synchronization protocols (LBTS, CMB, and GLM) for DEVS and Cell-DEVS applications. The protocols differ in the strategy of null message distribution. The goal is to analyze the effect of different conservative synchronization mechanism on the overall performance of the simulation. We conducted thorough experiments using communication-

intensive and computation-intensive Cell-DEVS models to analyze different metrics such as total execution time, blocked time, memory consumption, total number of positive and null event, as well as null message ratio. The results showed that the GLM protocol outperformed the other two protocols at every scenario. In most cases, CMB outperformed LBTS when small number of nodes were participating. However, as the number of processors increased, LBTS produced better results compared to CMB. We are currently working on a thorough testing analysis by conducting sensitivity analyses using larger and more complex models on both CCD++ with different conservative protocols and the purely optimistic simulator (PCD++) [5] to provide a reference guide on whether to use a conservative simulator or an optimistic one and under what circumstances one outperforms the other.

7. REFERENCES

[1] Zeigler, B., T. Kim, and H. Praehofer. 2000. Theory of modeling and simulation. 2nd Edition. Academic Press.

[2] Wainer, G., Giambiasi, N. "N-dimensional Cell-DEVS models". Discrete Event Dynamic Systems 12(2): 135–157. 2002.

[3] Chow, A. C. and B. Zeigler. 1994. "Parallel DEVS: A parallel, hierarchical, modular modeling formalism". In Proceedings of the Winter Computer Simulation Confer-ence, Orlando, FL.

[4] Wainer, G. 2002. CD++: A toolkit to develop DEVS models. Software, Practice & Experience, 32:1261-1306.

[5] Q. Liu, G. Wainer, "Parallel environment for DEVS and Cell-DEVS models". SIMULATION 83(6), 2007, pp.449-471.

[6] Radhakrishnan, R., Martin, D. E., Chetlur, M., Rao, D. M., Wilsey, P. A. "An object-oriented Time Warp simulation kernel". ISCOPE, LNCS 1505, 1998, pp. 13-23.

[7] Jafer, S., Wainer, G., "Conservative DEVS - A Novel Protocol for Parallel Conservative Simulation of DEVS and Cell-DEVS Models". DEVS/TMS'10. 2010.

[8] Chandy, K. M.; Misra J. "Distributed simulation: A case study in design and verification of distributed programs". IEEE Transactions on Software Engineering. pp.440-452. 1978.

[9] Bryant, R.E. "Simulation of packet communication architecture computer systems". Massachusetts Institute of Technology. Cambridge, MA. USA. 1977.

[10] Jafer, S., Wainer, G., "Global Lookahead Management (GLM) Protocol for Conservative DEVS Simulation". Procedings of IEEE DS-RT, Washington DC. 2010.

[11] Zeigler, B.; Moon, Y.; Kim, D.; Kim, J. G. "DEVS-C++: A high performance modeling and simulation environment". 29th Hawaii International Conference on System Sciences. 1996.

[12] Zeigler, B.; Kim, D.; Buckley, S. "Distributed supply chain simulation in a DEVS/CORBA execution environment". Proceedings of the 1999 Winter Simulation Conference. 1999.

[13] Kim, K.; Kang, W. "CORBA-based, Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-hierarchical One". International Conference on Computational Science and Its Applications (ICCSA).Assisi, Italy. 2004.

[14] Cheon, S.; Seo, C.; Park, S.; Zeigler, B. "Design and implementation of distributed DEVS simulation in a peer to peer network system". Advanced Simulation Technologies Conference. Arlington, VA. 2004.

[15] Zhang, M.; Zeigler, B.; Hammonds, P. "DEVS/RMI – An auto-adaptive and reconfigurable distributed simulation environment for engineering studies". DEVS Integrative M&S Symposium (DEVS'06). Huntsville, AL. 2006.

[16] IEEE std 1516.2-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification. Institute of Electrical and Electronic Engineers, New York, NY, 2001.

[17] B.P. Zeigler, G. Ball, H.J. Cho and J.S. Lee, "Implementation of the DEVS formalism over the HLA/RTI: Problems and solutions". Simulation Interoperability Workshop, Orlando, FL, 1999.

[18] T. Lake, B.P. Zeigler, H.S. Sarjoughian and J. Nutaro,"DEVS Simulation and HLA Lookahead". Simulation Interoperability Workshop, Orlando, FL, 2000.

[19] G. Zacharewicz, N. Giambiasi and C. Frydman, "A New Algorithm for the HLA Lookahead Computing in the DEVS/HLA Environment". European simulation Interoperability Workshop, Toulouse, France, 2005.

[20] Himmelspach, J., R. Ewald, S. Leye, and A. M. Uhrmacher. "Parallel and Distributed Simulation of Parallel DEVS Models".In Proceedings of the SpringSim '07, DEVS Integrative M&S Symposium, 249–256: SCS.

[21] Liu, Q., Wainer, G. "Lightweight Time Warp - A Novel Protocol for Parallel Optimistic Simulation of Large-Scale DEVS and Cell-DEVS Models", DS-RT 2008.

[22] Lubachevsky, B.D. "Efficient distributed event-driven simulations of multiple-loop networks". Communications of ACM 32, (January 1989), 111-123.

[23] L Lamport, KM Chandy. "Distributed snapshots: Determining global states of distributed systems". ACM Transactions on Computer Systems, 1985.

[24] Rothermel, R., "A Mathematical Model for Predicting Fire Spread in Wild-land Fuels". Research Paper INT-115. Ogden, UT: U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station. 1972.

[25] Wainer, G., "Applying Cell-DEVS methodology for modeling the environment". SIMULATION 82(10), 2006, pp.635-660.