# Fault-tolerant Distributed Discrete Event Simulator Based on a P2P Architecture

Jorge Luis Ramírez Ortiz
*Dept. of Electrical Engineering*
*Autonomous Metropolitan University*
*Iztapalapa, México*
*Email: jramirezort@gmail.com*

Ricardo Marcelín Jiménez
*Dept. of Electrical Engineering*
*Autonomous Metropolitan University*
*Iztapalapa, México*
*Email: calu@xanum.uam.mx*

***Abstract*—We describe the construction and performance evaluation of a new distributed discrete events simulation (DDES) tool, based on the Peer-to-Peer (P2P) paradigm. This approach allows the utilization of redundant resources in order to withstand failures on the very processing entities in charge of the simulation work. Our results show that the mechanisms supporting dependability are expensive and are only recommended for long-lasting and very processing-demanding simulations.**

***Keywords-DDES; P2P; redundancy; snapshot; fault-tolerance.***

## I. Introduction

Distributed Discrete Events Simulation (DDES) has found applications to study those systems made up from a massive number of components interacting over a time scale, where it is also necessary to reproduce their complex behavior under controlled conditions and with very fine granularity, i.e., with a high degree of realism. This is the case, for instance, of modern telecommunications systems, very large scale of integration (VLSI) circuits or even some biological models.

The basic operation of DDES consists in dividing the model that represents the system under study, in such a way that each of the resulting parts is simulated using a different computer. This also implies that the supporting computers should be connected by means of a communications network, in order to simulate the possible interactions among the parts of the model.

It is considered that the fundamentals of DDES were settled by Misra [1] with his seminal paper from 1986. Nevertheless, the major breakthroughs on the subject were achieved with the development of high speed networks over the last fifteen years about. Despite of the fact that DDES has evolved to be part of the ordinary toolbox of many research teams, there are open issues on the subject representing a challenging area of opportunity. This is the case of dependability. Suppose that in the middle of a long-lasting simulation, a given computer crashes. The ongoing simulation should be restarted from zero unless a fault-tolerant mechanism is implemented.

In the meantime, new paradigms have been developed in the fields of parallel and distributed computing. These new approaches foster the cooperative work among the components of the very system, in order to tackle complex problems. P2P systems, for instance, have found applications in situations where it is required to split up a big task to render smaller problems that can be assigned to a given number of appointed peers. Projects like seti@home [22], or einstein@home [23], and more recently folding@home [21] or rosetta@home [24], are representatives of this new trend.

In this work, we introduce the construction and performance assessment of a new prototype DDES tool based on the P2P paradigm. Our proposal supports crash failures as well as peer departures. If necessary, a missing component can be replaced by a spare peer and the ongoing simulation can be restored to a previously recorded global state, instead of starting over again. Our prototype is built using JXSE [20], the Java platform for P2P applications development.

The rest of this paper includes the following parts: Section II introduces the basic concepts of discrete event simulation, Section III gives a general view about related work, Section IV describes the operations of our prototype, Section V presents the results of the tool's performance evaluation, and Section VI shows our conclusion and future work.

## II. Basic Concepts

A discrete event simulation can be understood as a collection of logical processes. The interaction between any couple of processes is modelled by the exchange of time-stamped messages. This exchange is said to follow the restriction of local causality, if and only if each logical process dispatches all its schedulled events, including the messages that receives, according to their timestamps.

Notice that a logical process should stop any activity until it receives a message from each of the processes that interact with it, to be sure that it chooses the message with the smallest timestamp. Nevertheless, this approach poses the risk of creating a deadlock among a set of entities interacting in a circular way.

There exist two types of methods that deal with the risk of blocking in distributed simulations. On one side, we found the *optimistic procedure*. In the other side, we have the *conservative procedure*. In the first case, messages can be exchanged as they are produced. This decision may lead to a potential violation on the causal order of events, which is detected when a given entity receives a straggler message and it finds out that its corresponding timestamp is smaller than the timestamps of a number of messages already processed. This means that the straggler should have been dispatched before any of them. To fix this condition, the receiver starts a local procedure called *rollback*, that restores its local state to a previous one where the new set of messages can be processed accordingly. If necessary, the entity in charge of rollback must send the corresponding *anti-message(s)* to cancel the effect of any message that could have been issued out of order. In the other side, the conservative procedure avoids the possibility of executing actions out of chronological order. This time, entities

 ISBN: 978-1-61208-169-4 

exchange the so-called *null-messages*, which do not carry any physical meaning. The transmitter of a null-message sets a lower bound on the timestamp for the next meaningful message that could be issued. Each entity knowing the least timestamp that can receive from any of its incoming channels, will be able to dispatch any message in the right order.

The use of an optimistic synchronization mechanism also implies that the involved participants should take the responsibility of saving their local states to support rollback. Therefore this solution implies the utilization of bounded storage capacities. To prevent storage overflow, it is necessary to implement a *fossil recollection mechanism*. This is, a mechanism to dismiss previous recorded states that can not be reached, by no means, during rollback. It is known that no rollback can restore the state of any processing unit beyond the so-called *global virtual time* (GVT) [1][11][14][15].

Let $p_i$ be a process that takes part in a distributed execution, such that $\sigma_i^0$ is the initial state of $p_i$, and let $\sigma_i^k$ be the state of the process right after executing its $k$-th local event $e_i^k$. The *global state* of the distributed execution will be the $n$-tuple $\Sigma = (\sigma_1 \dots \sigma_n)$, made up with the local states of the participating processes. We now define a *cut* $C = h_1^{c_1} \cup \dots \cup h_n^{c_n}$, where $h_i^{c_k}$ is the ordered set of events dispatched at process $i$, up to its last local event $c_k$. Indeed, the cut can also be described in terms of the tuple $(c_1 \dots c_n)$. The set $(e_1^{c_1} \dots e_n^{c_n})$, including the last event on each process, is called the border of the cut. Evidently, each cut $(c_1 \dots c_n)$ defines a global state $(\sigma_1^{c_1} \dots \sigma_n^{c_n})$. Based on the "happened before" relation $(\rightarrow)$, it is possible to define a consistent cut if, for any to events $e$ y $e'$:

$$(e \in C) \wedge (e' \rightarrow e) \Rightarrow e' \in C. \quad (1)$$

Otherwise, we said that we deal with an inconsistent cut. Therefore, a *consistent global state* will be a global state corresponding to a consistent cut. There exist a well-known collection of distributed mechanisms that can be employed to produce the global state of an ongoing distributed execution [18][10].

## III. Related work

In this section, we present a collection of discrete events simulators, whose functionalities address many of the features of our proposal. By no means we intend to present an exhaustive description, but a sample of the systems that we consider to be closely related to ours.

The list of tools that we decided to consider includes the following simulators: Parsimony [2], GPDES [3], Omnet++ [4], POSE [6], $\mu$sik [7], and Aurora [8]. We also present Table I, where we describe the features that led our search. These are general purpose tools. The label "C-C" stands from "Client-to-Client", which means that the tool supports the communications between any processing unit. In contrast, "C-S" means that each client, or processing unit, is only aware of the existence of the server from which receives workload and to which sends the results of its local processing. The rest of the columns are self-explanatory.

It is apparent that only Aurora supports either fault-tolerance, or changes on the underlying communications network. Nevertheless, it is also important to point out that this system can not be considered a real DDES tool, as it does not partition the instance of the model under study to allocate each of the resulting parts to a different processing unit. Instead, it allocates a completely different instance of the model to each of the available processing components, pretty much in the spirit of systems like seti@home, folding@home, among others. In contrast, distributed DES techniques take for granted that the processing units are required to exchange information among themselves in order to produce the trace that represents the behavior of the system under study.

Fault-tolerance is a pending issue do not addressed on any of the tools of our list. The difficulties of building a DDES supporting fault-tolerance lie on the fact that it is required a thorough design including three types of redundancy: i) space or component redundancy offering spare units to replace any possible active unit that may go out of service, ii) information redundancy to regularly record a snapshot or global state of the ongoing distributed execution, and iii) time redundancy to repeat a given distributed execution from a previously recorded snapshot, when a faulty component was still active (before the last failure ocurred). For this purpose the missing unit is previously replaced by a spare unit, which now starts from the corresponding local state of the unit that replaces. Previous works [9][19] have recognized that the toughest problem comes from the construction of a global state of the underlying asynchronous execution, specially when it can not be granted the existence of FIFO channels.

## IV. Architecture

The design of our proposal is based on two types of entities, in charge of a simulation: the coordinator, implemented by a rendezvous type peer, and the workers, implemented by full-featured edge peers [13]. A worker contacts the coordinator to offer its processing capacities. From the coordinator's perspective these cooperative peers are regarded to be either as idle, or active workers (see Figure 1).

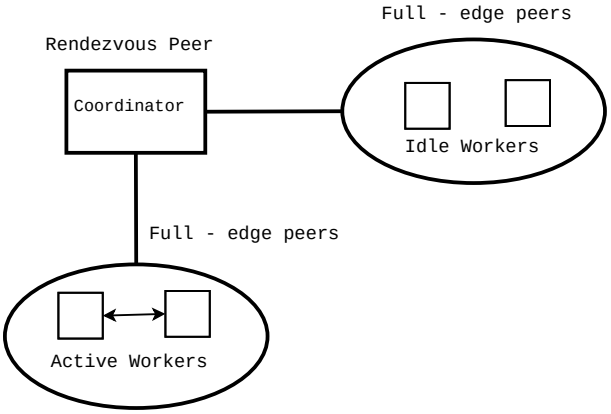



Figure 1. Architecture of our P2P-based simulator.

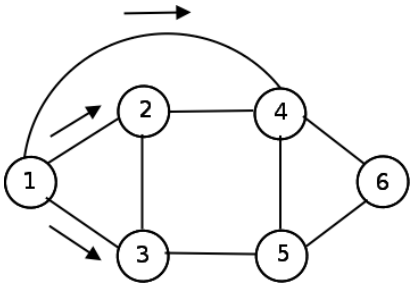
At the starting stage, the coordinator considers that any available worker is idle. The coordinator knows the graph representing the system about to be simulated (see Figure 2(a)). Then, it splits up (partitions) the graph into a fixed number o subgraphs (see Figure 2(b)) and allocates each of the resulting parts to an idle worker, which now is consider to be active (see Figure 2(c)) and (see Figure 2(d)).

 ISBN: 978-1-61208-169-4 

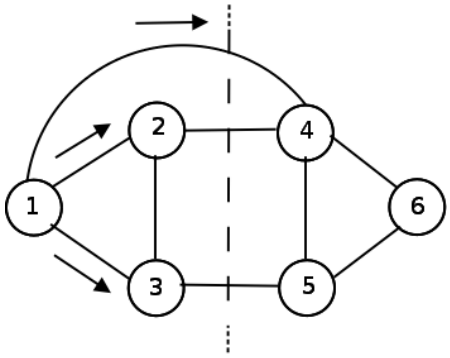
Table I
SOME GENERAL PURPOSE DDES

| Project | Comms. | Synch. | Network | Fault-tolerance |
|---|---|---|---|---|
| Parsimony | C-C | both | static | N |
| GPDES | C-S | — | static | N |
| OMNeT++ | C-C | both | static | N |
| POSE | C-C | optimistic | static | N |
| $\mu$sik | C-C | both | static | N |
| Aurora | C-S | — | dynamic | Y |

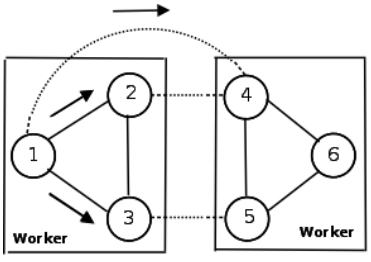
The coordinator triggers the simulation sending a message to the active worker(s) in charge of the node(s) which is (are) supposed to receive the starting event(s). On each active worker there is a single execution thread that processes all possible messages sent to the given peer.



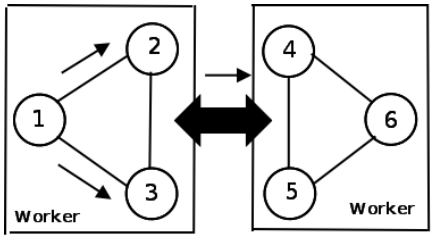

(a) System Model




(b) Partition of the system




(c) Allocation and logic communication




(d) Real communication

Figure 2. System's graph partition and allocation.

Our simulation tool supports two different restoration mechanisms, the so-called local rollback or restoration of type 1, which is trigered when some straggler message is received out of order and threatens the causal delivery of events. The restoration of type 2, or global rollback, happens when an active worker leaves the system or crashes, and the whole system must be restarted from a previously recorded global state. In both cases the evolution of any logical process is recorded by 3 complementary queues: *the input message queue, the output message queue, and the former states queue*.

In the case of local rollback, only the logical processes directly involved are restored to a previous state in order to guarantee the causal delivery of the late message. If necesary, this procedure may imply the transmission of some anti-messages that start a similar procedure at the receiving peers. Also, each peer is required to store by itself the states of the local processes that allocates. In contrast, global rollback happens when the coordinator detects that an active peer is missing. In this condition, it stops the overall ejecution, then it enables an idle peer to replace the missing one. Finally, the coordinator "rewinds" the whole system to a previously recorded snapshot. This also means that the coordinator is responsible for storing and updating the snapshots that regularly takes. By updating we mean that when the last state has been recorded, the coordinator eliminates the previous one, in order to maintain a limited amount of storage to support this mechanism.

It is apparent that the coordinator is also responsible for recording the global state or snapshot (SN) of the ongoing distributed execution. To accomplish this task, *it regularly stops the overall execution and sends a snapshot message to an appointed worker which starts the Chandy-Lamport protocol among the active workers*. When each worker recognizes that it has finished the protocol, it sends its local records to the coordinator which collects these results to put the pieces together and store the resulting snapshot. This procedure is the key to support the global rollback mechanism.

Similarly, the coordinator is also responsible for triggering the distributed procedure that determines the GVT. When each worker has finished its local procedure, it sends its local proposal to the coordinator, which collects this value from every worker and picks up the smallest result, now considered the new GVT. Finally, the coordinator broadcast this new value to every active peer. The GVT enables the active workers to proceed with their fossil recollection process which, in turn, complements the local restauration process. SN and GVT messages have the first and second highest priority above any possible message that can be received at any worker.

Last, the coordinator collects the traces generated by the

 ISBN: 978-1-61208-169-4 

active workers, containing the partial results of the simulation. It is able to recognize when the GVT reaches a predefined value $+inf$, which implies that the running simulation is finished.

## V. Performance assessments

We devised a comprehensive set of experiments to evaluate our tool's performance under different conditions. We also considered that the selection of the type of system to be simulated was a key aspect for this experimental assessment. We looked for a simple system with a deterministic behavior, where the complexity of its operations comes from the size of the system and the connection among its elements. We also looked for a type of system ranging from very small to massive instances.

Each instance of the system to be studied is represented by a connected graph, called the communications graph. The simulation to be run on this graph consists in the execution of the PIF algorithm (which stands from Propagation of Information with Feedback)[17]. An appointed node in the communications graph, from now on called the root, starts sending an information unit to each of the nodes that share an edge with it, i.e., to its neighbors. A node is said to be "woken" (see Figure 2(a)), when it receives this information unit for the first time, it also considers that its "father" is the node that woke it. On due time, each of the receiving nodes forwards the same unit to its corresponding neighbors but to its father. Only when the node has received this unit from each incoming link, it is able to send this information back to its father. The global effect of this simple algorithm is observed in two consecutive phases. During phase one (propagation or broadcast) we can imagine an expanding wave that goes from the root to the rest of the graph. When this wave reaches the boundaries of the graph it starts the second phase (convergecast). This time, information travels back to the root on top of the three induced on the graph during phase one.

We selected this simple algorithm to simulate, for we know in advance the place where action starts and finishes. Indeed, we fix this point. Also, we know the total number of information units to be exchanged. Besides, the simulation time depends only on the underlying graph diameter.

We stressed our tool with three different types of experiments:

a. In the first group of experiments, we tested a fixed set of systems under 3 different conditions, using a centralized simulation, using a distributed simulation with 2 active workers in charge and finally, a distributed simulation with 3 active workers. We compared the time required to finish the simulation under each of these cases.

b. In the second group of experiments, we measured the global added cost required to support a fault-tolerant system. We considered that this feature is mainly based on the capability of recording snapshots and the evaluation of the GVT. Therefore, we ran new system instances, but this time with the snapshot and GVT procedures switched on and off, respectively. Notice that, we did not inject faults, but we evaluated the price of the "insurance" mechanisms, although these mechanisms were never invoked.

c. In the third group of experiments, we wanted to evaluate the tool's resilience. In order to trigger the global restoration procedure, we dismissed an active worker immediately after the first snapshot was already stored at the coordinator. We used the same system instances from the previous study. We compared the time required to finish the simulation under each of the cases, with and without failure.

About the graph representing the model of the system under study, we tested 10 different graph orders, from 100 up to 1000 nodes. For each order we created 10 different graph instances. Each instance is a graph randomly created.

### A. First group of experiments: centralized vs distributed

In this experimental design we simulated the PIF algorithm, with two types of configurations: centralized (1 active worker) and distributed (2 and 3 active workers). Communication channels are assumed to have constant delay. Distributed simulations include snapshot and GVT mechanisms.

Results show that (Figure 3) the graph order is a key element to decide the best configuration that supports the fastest simulation, i.e., a distributed simulation is not necessarily the fastest, specially when we deal with small graph orders.

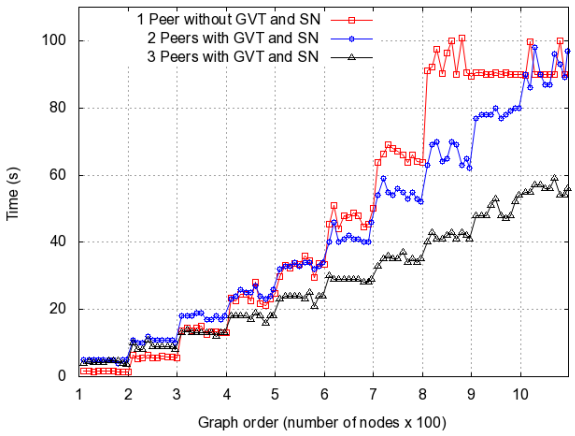


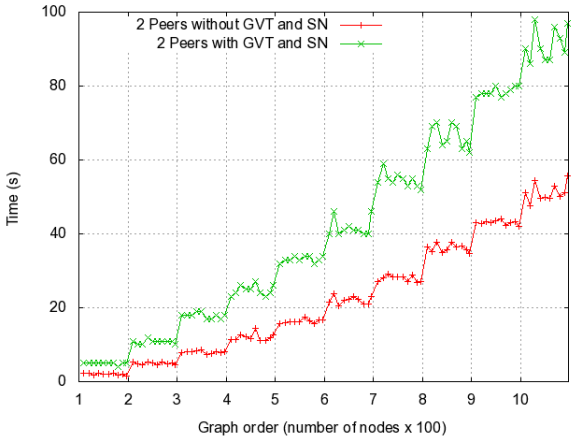

Figure 3. Centralized simulation vs distributed simulation.

### B. Second group of experiments: the added cost of fault-tolerance mechanisms

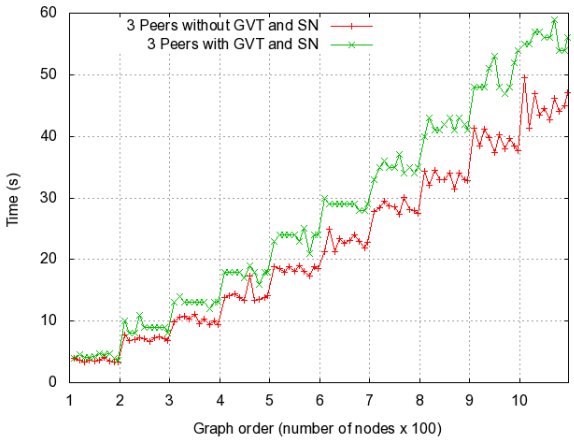
In this new set of experiments we evaluated the involved cost, measured in simulation time, required to deploy the fault-tolerance mechanisms. We consider that this capability basically depends on the utilization of the snapshot and GVT procedures. For this purpose we tested 2 distributed configurations, with 2 and 3 active workers, respectively. On each case, we measured the elapsed time required to finish a given simulation, with and without fault-tolerance mechanisms included. It is very important to quote that not any worker was dismiss, but we simply switched on and off these mechanisms to measure how much the processing time is lengthened, when these "insurance policies" are included. Also, it is important to notice that, for those cases where mechanisms are included, they are only executed once, during the overall elapsed time.

Results show that (see Figures 4(a) y 4(b)) the extra resources, required to support fault-tolerance, depend on the number of involved peers that share these mechanisms. In these particular cases, 3 peers apparently have a smaller impact on

 ISBN: 978-1-61208-169-4 

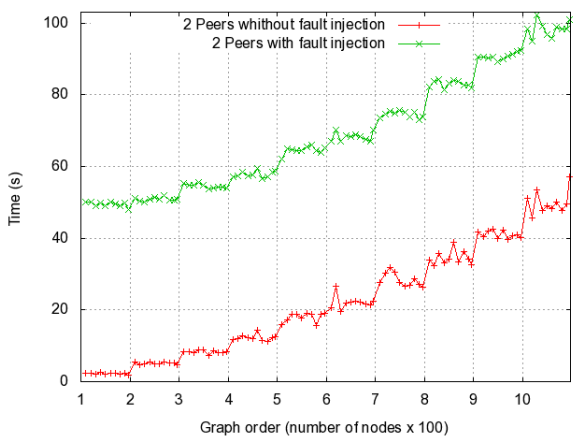



(a) 2 Workers

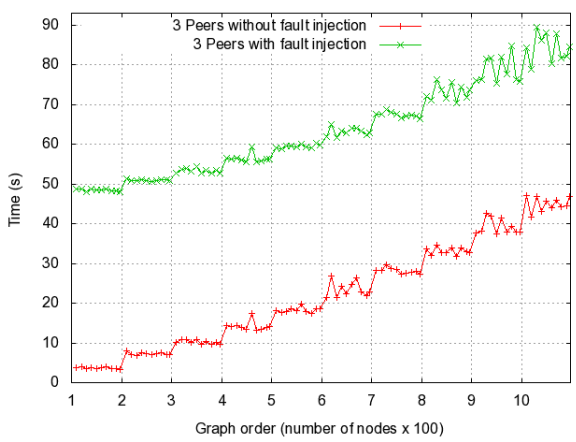



(b) 3 Workers

Figure 4. Simulation with/without GVT and SN.




(a) 2 Workers




(b) 3 Workers

Figure 5. The effect of fault injection

the simulation added cost, compared to the impact observed on the configuration made up with 2 peers only.

### C. Third group of experiments: fault injection

Finally, in this set of experiments we wanted to know whether it is worth using fault-tolerance mechanisms or it is better to start over a simulation from the very beginnig. In order to compare these two possibilities, we assume that an active worker may go out of service an instant before the finalization of the underlying simulation. We tested 2 distributed configurations with 2 and 3 active workers. In both cases we compared the elapsed simulation time with and without fault injection. A fault is produced by dismissing an active worker immediately after the snapshot and the GVT mechanisms have been executed. Results show that (see figs. 5(a) y 5(b)) the graph order is the key to answer this question.

## VI. Conclusion and Future Work

In this work, we described the construction of a prototype that demonstrates the viability of a DDES tool based on P2P entities. The design of our proposal is based on two types of peers: the coordinator, implemented by a rendezvous type peer, and the workers, implemented by full-featured edge peers. This solution supports workers failures, as well as departures. We focused our work on measuring the costs associated to dependability. Fault-tolerance mechanisms are expensive and it is worth their utilization provided that we deal with a long lasting simulation. New open issues are pending, such as the optimal snapshot recording frequency, the efficient storage of the global state, the possibility of load rebalancing on the fly and the possibility of supporting a failure at the very coordinator. In contrast to the "@home" type applications, DDES requires a strong interaction between participants. Therefore, we consider that this new approach will turn into a feasible solution only when final users, behind the altruistic peers, will be connected by high-speed channels.

## References

[1] J. Misra, "Distributed discrete event simulation," Computing Surveys, vol. 18 Issue 1, pp. 39-65, Mar. 1986.

[2] B. Preiss, "The parsimony project: a distributed simulation Testbed in Java," Proc. 1999 International Conference On Web-Based Modelling and Simulation, vol.31 of Simulation Series, pp. 89-94, San Francisco, CA, January 1999. Society for Computer Simulation.

[3] L.M. Campos. "A general-purpose discrete event simulator," Symp. on Performance Evaluation of Computer and Telecomunication Systems, Orlando, USA, Jul. 2001, pp. 1-12.

 ISBN: 978-1-61208-169-4 

[4] OMNET++ User Manual. Internet: http://www.omnetpp.org/doc/manual/usman.html#sec378, 28.07.2011.

[5] B. Preiss and I. MacIntyre. "Yaddes yet another distributed discrete event simulator," User Manual. Internet: http://www.brpreiss.com/reports/ccng/E-197/report.pdf, 28.07.2011.

[6] T.L. Wilmarth and L.V. Kale, (2004). "POSE: getting over grainsize in parallel discrete event simulation," Proc. International Conference on Parallel Processing. Los Alamitos, CA: IEEE Computer Society, vol. 1, Aug. 2004, pp. 12-19, doi:10.1109/ICPP.2004.1327899.

[7] K.S. Perumalla, (2005). "$\mu$sik a microkernel for parallel distributed simulation systems," Proc. 19th Workshop on Principles of Advanced and Distributed Simulation, Los Alamitos, CA: IEEE Computer Society, Jun. 2005, pp.1-12, doi:10.1109/PADS.2005.1.

[8] A. Park and R.M. Fujimoto, " Aurora: an approach to high throughput parallel simulation," Proc. 20th Workshop on Principles of Advanced and Distributed Simulation, Los Alamitos, CA: IEEE Computer Society, Jun. 2006, pp. 3-10, doi:10.1109/PADS.2006.11.

[9] O. P. Damani and V. K. Garg "Fault-Tolerant Distributed Simulation," Proc. Twelfth Workshop on Parallel and distributed simulation , May. 1998, pp.38-45.

[10] K. M.Chandy and L. Lamport, "Distributed Snapshots: determining global states of distributed systems," ACM Transactions on Computer Systems, Feb. 1985, 3(1), pp. 63-75.

[11] D.R Jefferson, "Virtual time," ACM Transactions on Programming Languages and Systems, Jul. 1985, 7(3), pp. 404-425.

[12] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," IEEE Transactions on Software Engineering, Jan. 1987, pp. 23 - 31.

[13] J. Verstrynge, Practical JXTA cracking the puzzle, Dawning Streams, Inc.,Lulu Enterprises, Inc., Netherlands, 2008.

[14] Y. Lin and P.A. Fishwick, "Asynchronous parallel discrete event simulation," IEEE Trans. on Systs., Man and Cibernetics, Part A: Systems and Humans, Jul. 1996, pp. 397-412.

[15] A. Ferscha, Parallel and Distributed Simulation of Discrete Event Systems, Handbook of Parallel and Distributed Computing, McGraw Hill, 1995.

[16] R.M. Fujimoto, "Parallel and Distributed Simulation," Winter Simulation Conference, P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans (eds.), Dec. 1999, pp. 122-131.

[17] A. Segall. "Distributed network protocol," IEEE Trans. on Information Theory, Jan. 1983, 29(1): pp. 23-35.

[18] L. Lamport. "Time, clocks and the ordering of events in a distributed system," Communications of the ACM, Jul. 1978, 21(7):pp. 558-564 .

[19] Y. Lin, "Design Issues for Optimistic Distributed Discrete Event Simulation," Journal of information science and engineering, 2000, Vol. 16, pp. 243-269.

[20] JXSE. Internet: http://jxse.kenai.com/, 28.07.2011.

[21] folding@home. Internet: http://folding.stanford.edu/, 23.05.2011.

[22] seti@home. Internet: http://setiathome.berkeley.edu/, 23.05.2011.

[23] einstein@home. Internet: http://einstein.phys.uwm.edu/, 23.05.2011.

[24] rosetta@home. Internet: http://boinc.bakerlab.org/rosetta/, 23.05.2011.

 ISBN: 978-1-61208-169-4 