**ASimJava: a Java-based library for distributed simulation**

Ewa Niewiadomska-Szynkiewicz and Andrzej Sikora

Abstract—The paper describes the design, performance and applications of ASimJava, a Java-based library for distributed simulation of large networks. The important issues associated with the implementation of parallel and distributed simulation are discussed. The focus is on the effectiveness of different synchronization protocols implemented in ASimJava. The practical example—computer network simulation—is provided to illustrate the operation of the presented software tool.

Keywords—parallel computations, parallel asynchronous simulation, computer networks simulation.

1. Introduction

The main difficulty in networks simulation is the enormous computational power needed to execute all events involved by packets transmission through the network. Recently computer network simulation has been an active research area. Numerous software systems have been engineered to aid researchers. The popular commercial and publicly released packet-level simulators like OPNET, NS [10] or OMNeT [11] require costly shared-memory supercomputers to run even medium size network simulation. Since parallel and distributed simulation is fast becoming the dominant form of model execution, the focus is on experiments carried out on parallel computers or computer networks. The ASimJava general structure enables to do parallel and distributed discrete-event simulations, [1] that can be described in terms of logical processes (LPs) and communicate with each other through message-passing. LPs simulate the real life physical processes PPs. Each logical process starts processing as a result of event occurrence (from the event list or having received a new message). It performs some calculations and generates one or more messages to other processes. The calculation tasks executed in parallel require explicit schemes for synchronization. Two simulation techniques are considered [5]: synchronous and asynchronous. Synchronous simulation is implemented by maintaining a global clock (global virtual time—GVT). Events with the smallest time-stamp are removed from the event lists of all LPs, for parallel execution. Parallelism of this technique is limited, because only events with time-stamps equal to that of the global clock can be executed during an event cycle. Asynchronous simulation is much more effective due to its potentially high performance on a parallel platform. In asynchronous simulation each logical process maintains its own local clock (local virtual time—LVT). Local times of different processes may advance asynchronously. Events arriving at the local input message queue of a logical process are executed according to the local clock and the local schedule scheme. Synchronization mechanisms fall into two categories: conservative and optimistic. They differ in their approach to time management. Conservative schemes avoid the possibility of causality error occurring. These protocols determine safe events, which can be executed. Optimistic schemes allow occurrence of causality errors. They detect such error and provide mechanisms for its removal. The calculations are rolled back to a consistent state by sending out antimes-sages. It is obvious that in order to allow rollback all results of previous calculations have to be recorded.

2. Description of ASimJava

2.1. General description

The ASimJava general structure enables to do parallel and distributed discrete-event simulations, [1] that can be described in terms of logical processes (LPs) and communicate with each other through message-passing. LPs simulate the real life physical processes PPs. Each logical process starts processing as a result of event occurrence (from the event list or having received a new message). It performs some calculations and generates one or more messages to other processes. The calculation tasks executed in parallel require explicit schemes for synchronization. Two simulation techniques are considered [5]: synchronous and asynchronous. Synchronous simulation is implemented by maintaining a global clock (global virtual time—GVT). Events with the smallest time-stamp are removed from the event lists of all LPs, for parallel execution. Parallelism of this technique is limited, because only events with time-stamps equal to that of the global clock can be executed during an event cycle. Asynchronous simulation is much more effective due to its potentially high performance on a parallel platform. In asynchronous simulation each logical process maintains its own local clock (local virtual time—LVT). Local times of different processes may advance asynchronously. Events arriving at the local input message queue of a logical process are executed according to the local clock and the local schedule scheme. Synchronization mechanisms fall into two categories: conservative and optimistic. They differ in their approach to time management. Conservative schemes avoid the possibility of causality error occurring. These protocols determine safe events, which can be executed. Optimistic schemes allow occurrence of causality errors. They detect such error and provide mechanisms for its removal. The calculations are rolled back to a consistent state by sending out antimes-sages. It is obvious that in order to allow rollback all results of previous calculations have to be recorded.
2.2. System architecture

One of the principle goals of the ASimJava was portability and usage in heterogeneous computing environments. Two versions of ASimJava are implemented: parallel and distributed. It is possible to join both of them in one simulator (Fig. 1). The Java messaging service (JMS) API provided by Sun Microsystems is used for interprocess communication in the case of distributed version. The asynchronous version of distributed simulation is applied in ASimJava.

Four synchronization protocols are provided:
- conservative protocol with null messages (CMB),
- window conservative protocol (WIN),
- Time Warp (TW),
- Moving Time Window protocol (MTW).

The short description of these protocols is presented in the Appendix.

2.3. Design overview

Current version of ASimJava is composed of five components:
- Graphical user interface (GUI)–responsible for user-system interactions.
- Basic library–a collection of classes implementing basic elements of a simulator, such as: logical processes, events, event lists, messages passing, etc.
- Synchronization protocols library–the library of classes implementing four synchronization algorithms (CMB, WIN, TW and MTW).
- Communication library–the library of classes that provides communication between the user interface and the simulator.
- Toolboxes–the collections of classes implementing basic elements of different systems. Currently available: computer network toolbox that is the collection of classes implementing elements of computer networks, such as router, hub, switch, etc. The package is flexible and can be easily extended by other toolboxes of classes, which are specific to a chosen case study.

The simulator built upon ASimJava classes has hierarchical structure. The simulated system is partitioned into several subsystems (subtasks), with respect to functionality and data requirements. They are implemented as LPs. Next, each LP can be divided into smaller LPs. Hence, the logical processes are nested (Fig. 2). Calculation processes belonging to the same level of hierarchy are synchronized.
2.4. User interface

ASimJava provides the graphical environment (shell) for supporting the considered case study implementation. The most important tasks of the user interface are as follows:

- supporting the process of defining a considered application,
- presenting of the calculation results,
- providing the communication with the user (during design and experimental phases).

The main element of the interface is the graph editor—the graphical tool for inserting the scheme of the simulated system. It is organized in nested manner, too. The user can start from dividing the considered system into several subsystems and inserting the scheme of it. Next, he can divide each subsystem into smaller ones (Fig. 3). For example, in the case of computer networks we can start from the decomposition of our application into local area networks (LANs), and next we can insert all elements that form each LAN (work stations, routers, switches, etc.). Within the next step the user is asked to provide some information related to the nodes and arcs of the inserted graph. In the case of nodes the required data are: parameters specific to the edited element and event list, in the case of arcs—maximal and minimal flows. The created graph together with all inserted data can be saved into the XML file.

2.5. Simulation under ASimJava

The experimental phase begins when all decisions regarding the simulated system are made. The simulation starts. The adequate programs corresponding to the nodes of the system graph are executed. The results of the calculations are displayed. The user employs monitoring and analysis of the current situation. All results may be recorded into the disc file during the experiment.

3. Case study results

The AsimJava library was used to perform simulation of several systems. In this paper the applications of a simple manufacturing system and a computer network are presented. The objective of all tests was to compare the effectiveness of considered synchronization protocols.

3.1. Manufacturing system

The first case study was related to simulation of a simple distributed manufacturing system, as presented in Fig. 4. A considered system consists of two sources P1 and P2, eight work stations P3–P10 and one sink P11. Jobs enter the manufacturing system at work stations P3 or P4. When a job has been serviced at the work station Pi it proceeds to the next work station. Service times at different work stations are different. Jobs may be queued at a station awaiting service. A work station takes one job from its input queue when it is free, services that job, and then sends it to the queue of the following work station. All work stations service the jobs in a first come, first served

Fig. 3. The simulated system scheme inserting.

Fig. 4. Example 1: manufacturing system.
basis. The job leaves the system after being serviced at work station P9 or P10. It is collected at the sink P11. Simulation experiments were performed under following assumptions:

- Two variants of application were considered:
  - variant A: each source generated 2 jobs;
  - variant B: each source generated 25 jobs.

- The time intervals in which the sources generate jobs and the service times at different work stations are given in the Table 1.

- The experiments were performed in the network of four Celeron 433 computers. The allocation of LPs simulating adequate physical processes to the computers was as follows: computer 1: LP1, LP3, LP5; computer 2: LP2, LP4, LP6; computer 3: LP7, LP8; computer 4: LP9, LP10, LP11.

### Table 1

<table>
<thead>
<tr>
<th>Variant</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
<th>P10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

The sources, the sink and each work station were simulated by 11 logical processes. If a job \( j \) arrived at \( P_i \) at time \( t \) then its service begins either immediately (at time \( t \)), if \( P_i \) is idle, or it begins right after the departure of the \((j - 1)\) job from this \( P_i \). Let \( t_{A_j} \) be the time of arrival of job \( j \) at \( P_i \), \( t_{D_j} \) the time of departure of job \( j \) from \( P_i \) and \( \Delta t_j \) the service time at this \( P_i \). Then we obtain:

\[
t_{D_j} = \max(t_{A_j}, t_{D_{j-1}}) + \Delta t_j.
\]  

(1)

The results of numerical experiments are presented in Tables 2 and 3. Different aspects were considered with respect to (w.r.t.) applied synchronization protocols:

- time of simulation;
- number of additional messages sent by calculation processes (CMB—null messages, WIN—global messages used for lengths of time windows calculation, TW—antimessages, MTW—antimessages and additional messages used to synchronization);
- number of rollbacks at TW;
- different lengths of time window at MTW.

It can be observed (Table 2) that the speed of simulation strongly depends on the applied protocol. The best results were obtained for hybrid approach MTW, the worst for conservative CMB. This was connected with different degrees of parallelism in the case of conservative and optimistic approaches, and overheads (additional messages, rollbacks).

The hybrid techniques seem to be promising w.r.t. optimistic TW. It may be profitable to decrease degree of available parallelism, increase the number of additional messages but reduce the number of rollbacks. In the case of combining TW and window techniques, the main problem is to estimate the proper length of the window. The simulation results strongly depend on this parameter (Table 3).

### Table 2

<table>
<thead>
<tr>
<th>Variant</th>
<th>CMB [s]</th>
<th>WIN [s]</th>
<th>TW [s]</th>
<th>MTW [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>247.83</td>
<td>25.81</td>
<td>34.55</td>
<td>20.93</td>
</tr>
<tr>
<td>B</td>
<td>316.86</td>
<td>88.43</td>
<td>52.34</td>
<td>41.25</td>
</tr>
</tbody>
</table>

### Table 3

<table>
<thead>
<tr>
<th>Time window size</th>
<th>Simulation time [s]</th>
<th>Additional messages</th>
<th>Number of rollbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>54.87</td>
<td>134</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>45.09</td>
<td>71</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>41.25</td>
<td>36</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>44.87</td>
<td>33</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>63.82</td>
<td>34</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>59.87</td>
<td>23</td>
<td>1</td>
</tr>
</tbody>
</table>

It seems that the length of each window should be calculated adaptively, taking into account the considered application and available hardware platform. So, the degree of parallelism reduction remains a topic of hot debate.

### 3.2. Computer network

The second case study was related to simulation of an IP network. The library of classes implementing network hardware (host, hub, switch, router, etc.) was developed. The experiments were performed for the network consisting of five LANs and one server room (13 servers), as presented in Fig. 5. Two scenarios of traffic with small packets of transmitted data (variant A) and large packets (variant B) were considered.

### Table 4

<table>
<thead>
<tr>
<th>Variant</th>
<th>WIN [s]</th>
<th>MTW-1 [ms]</th>
<th>MTW-10 [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>18 186</td>
<td>18 090</td>
<td>15 742</td>
</tr>
<tr>
<td>B</td>
<td>72 072</td>
<td>71 097</td>
<td>63 996</td>
</tr>
</tbody>
</table>

The experiments were performed in the network of four Celeron 433 computers. The allocation of logical processes simulating the considered computer network to the com-
puters was as follows: computer 1: LAN1, LAN2; computer 2: LAN3, LAN4; computer 3: LAN5; computer 4: server room.

The results obtained for two synchronization protocols—conservative WIN and hybrid MTW are presented in Table 4. Two lengths of the window in MTW (1 time unit and 10 time units) were taken into account. The application of CMB and TW synchronization protocols brought much worse results—the computation time increased seriously.

![Figure 5](image)

**Fig. 5.** Example 2: computer network.

Similarly to the previous example the best results were obtained for hybrid protocol.

4. Conclusions

Simulation plays an important role in the computer-aided analysis and design of large computer networks. The ASimJava software framework is suitable to solve many small and large scale problems, based on simulation. The package is flexible and can be easily extended by software modules, which are specific to a chosen application.

Appendix

Synchronization protocols

A.1. CMB protocols

In the case of the CMB scheme [7] each process executes events only if it is certain that no event with the earlier time stamp can arrive. At current time \( t \) the \( i \)th logical process \( LP_i \) computes the minimum time \( LV T_i = \min_{j=1,...,p} t_{ij} \), where \( t_{ij} \) denotes the time stamp of the last message received from process \( LP_j \) and \( p \) is a number of processes transmitting data to \( LP_i \). Next, every \( LP_i \) simulates all events with time stamps less than its \( LV T_i \). The CMB scheme in its basic form may lead to deadlock. Several approaches were proposed to resolution of deadlock:

- **Null messages.** The additional messages (null messages) [7], are sent by each process to the others, after completing the iteration. They are used to announce absence of messages. The information about the earliest possible time of the next event execution may be appended to the null messages. The evident drawback is the high volume of null messages, especially when we consider the high cost of message-passing in distributed memory machines.

- **Carrier null messages.** The additional messages for processes synchronization propagated through a system carry more information: the list of visited logical processes and pending event time [9].

A.2. Window protocols

Some of conservative algorithms assume constrain concurrent simulation activity to be within a window of global synchronization [1, 8, 9]. A minimum time window is identified for all logical processes, which calculations may be carried out concurrently. Typically, this calculation involves lookahead of some kind. At current time \( t \) each logical process \( LP_i \) is asked to compute the time \( T_i(t) \) of the next message it will send, based on its event list. The global time window \([t, T(t)]\) is defined; where \( T(t) = \min_{i=1,...,p} T_i(t) \), and \( p \) is a number of logical processes. Every process can simulate all events with time stamps within this window. Next, \( t = T \) and new window is calculated. The evident constraint on this scheme is that only the events within the same window can be executed concurrently. The events performed at the different windows are executed sequentially. The efficiency of this algorithm strongly depends on the calculations decomposition and allocation to the processors.

A.3. Time Warp protocols

The optimistic scheme—Time Warp protocol [3, 4] allows to each \( LP_i \) keeps calculations in its local simulated time \((LV T_i)\) under the assumption that message communication between processors arrives at proper time. In the case when causality error occurs, e.g., because \( LP_i \) receives a message with a time stamp smaller than \( LV T_i \), the calculations are rolled back. The state of \( LP_i \) is restored to that, which existed prior to processing the event. Additional messages, i.e., antagonists are sent to cancel the previously sent messages. After receiving the antagonist the receiver is rolled back, possibly generating additional antagonists. In order to allow rollback, all results of previous calculations has to be recorded, which is connected with requirements of extra resources for states of all processes storing during calculation.
A.4. Moving Time Window protocol

The principal advantage of Time Warp over presented conservative schemes is that it offers the potential for greater exploitation of parallelism to the programmer. Despite this advantage Time Warp has some drawbacks: an excessive amount of wasted, rolled back computations and inefficient use of memory. The danger is greatest when interaction between processors is light and processors loads are uneven. It may result cascading rollbacks. Because of that, the issue of combining conservative and optimistic approaches in a hybrid protocols has received considerable attention in recent years. MTW belongs to the group of protocols, which reduce degree of available parallelism. It limits execution of Time Warp to the defined time window $[t, t + \Delta t]$ [12]. Events with time stamps greater than or equal to $t + \Delta t$ are not executed. All processes are synchronized at time $t + \Delta t$ and a new window is simulated. The rollbacks may occur only within the current time window. The problem is to calculate adequate $\Delta t$.

References