JSDESLib: A Library for the Development of Discrete-Event Simulation Tools of Parallel Systems

Luís F. W. Góes, Christiane V. Pousa, Milene B. Carvalho, Luiz E. S. Ramos, Carlos A. P. S. Martins

Computational and Digital Systems Laboratory
Pontifical University Catholic of Minas Gerais
Av. Dom José Gaspar 500, 30535-610, Belo Horizonte, MG, Brazil
lfwgoes@yahoo.com.br, christ.bh@terra.com.br, milenebc@yahoo.com.br, luizedu@uai.com.br, capsm@pucminas.br

Abstract

In this paper, we present, verify and analyze the performance of JSDESLib (Java Simple Discrete-Event Simulation Library). It is a simplified and easy-to-use Java-based library for the development of discrete-event simulation tools of parallel systems. Our main objectives are: to propose, develop and implement JSDESLib: as a parallel event-oriented simulation library that simplifies the development of simulation tools; with a next earliest event mechanism and communication management between entities. Other main objectives are: to verify and analyze the performance of JSDESLib. A secondary objective is to present some simulation tools that were developed with that library. The main contributions of this work are: the proposal, implementation and performance analysis of JSDESLib; simplicity and ease usage of a parallel-optimized implementation of that library.

1. Introdução

In the face of activities that demand each time higher performance, parallel processing has become an interesting alternative to the limited power and speed of the existing sequential processors [9]. In the latest years, there were developed several parallel machines. The clusters of workstations are one of the most successful architectures both in academy and commerce. This is so mainly due to its high performance, low cost, good scalability etc. In order for that kind of computer to work optimized, it is important that the designer considers the entire hardware and software environment. That includes: the operating systems, compilers, applications network technologies, communication protocols, programming environments etc. All of them must be developed concerning the issues inherent to parallel processing. In order to analyze the results obtained from running parallel programs on clusters of workstations, we can use functional and/or performance analysis of algorithms (software) and architectures (hardware).

There are three well-known techniques for performance analysis: experimentation, analytical modeling and simulation [10] [11]. Simulation emerges as a lower cost alternative to experimentation, as well as a more precise option in comparison with analytical modeling [10] [11]. Simulations usually require the use of a computer for the numerical evaluation of a model. The collected data is used to estimate the actual features desired in the model. This provides a more detailed, flexible and controlled environment [10] [11]. Thus, analysts can compare his alternatives with several workloads. Besides, there are several types of software that aid in the development of simulations. Some examples are: simulation libraries (SimJava, JSIM, SimKit, ParaSol, SimJava, SPADES [12] [13] etc.), simulation languages (SIMSCRIPT [11], SLAM [11] [12]) and specific simulators (Gridsim [2], Simgrid [3], SRSim [1] [17], cache Simulator [15] [16], digital signal processor simulator [4], parallel task scheduling simulator [5], cluster simulator [8] [19] etc).

Discrete-event simulation addresses the modeling of systems that evolve over time. It is based on a representation in which the state variables change instantaneously at separate time points. These points represent the time instants in which the events occur. Each event is defined as an instantaneous occurrence that can change the state of the system [2] [10] [11].

In this paper, our main objectives are: to propose, develop and implement JSDESLib: as a parallel event-oriented simulation library that simplifies the development of simulation tools; with a next-time forwarding mechanism and communication...
management between entities. Other main objectives are: to verify and analyze the performance of JSDESLib. A secondary objective is to present some simulation tools that were developed with that library.

This paper is organized as follows: in section 2, we present the related works; section 3 contains the proposal of the architecture and the implementation of the JDESLib library; in section 4, we verify JDESLib; in section 5, we present some tools that were developed with JDESLib as well as some performance analysis; finally in section 6, we highlight the conclusions and future works.

2. Related Work

There are many libraries for simulation tools development. The most of them was built in Java, C++ or simulation languages [12] [13] [17]. Generally, the researches build simple libraries for the development of simple specific simulators and do not describe them or do not put available their font code.

Among the simulation libraries, that have their font code and documentation available, are JSIM, SimKit, ParaSol, SimJava, SPADEs etc [6] [12] [14] [17] [18]. The library that more resembles our proposal is [13] [17], that is also based in Java, allows discrete event simulation and object orientation. However, we will also analyze the library SPADEs [18].

The SimJava is a library based on Java. It has a discrete event simulation kernel and provides visual animation during the simulation with objects. Moreover, it simulates static nets of executed active entities in parallel, which communicate by means of ports. Each ports of an entity is connected to the ports of another entity, by where the events are passed. The simulation is managed by the Sim_system that keeps a queue of events ordered by time.

The entities use three basic methods to control its behavior in the simulation: schedule (), wait () and hold (t). The schedule () method sends an event for the entity connected to one determined ports. The wait () method makes an entity waits for an event.

Finally, the hold (t) method, blocks an entity for t units of simulation time. The simulation finishes when the queue of events becomes empty [13] [17]. Despite the SimJava has many of the desired characteristics, it has the following disadvantages: the use of ports limits the events to just one addressee for ports, or either, it does not support broadcast and nor multicast of an event; the method hold (t) is unnecessary in the majority of the implementations and the explicit use of the method wait () by the programmer can cause timing errors in the execution of the simulation.

### Table 1 – Simulation Libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Implementation</th>
<th>Synchronization</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimJava</td>
<td>Event</td>
<td>Sequential</td>
<td>Java</td>
</tr>
<tr>
<td>SPADEs</td>
<td>Process</td>
<td>Sequential</td>
<td>C++/Java</td>
</tr>
<tr>
<td>JSIM</td>
<td>Process</td>
<td>Distributed</td>
<td>Java</td>
</tr>
<tr>
<td>SimKit</td>
<td>Process or Event</td>
<td>Sequential</td>
<td>Java</td>
</tr>
</tbody>
</table>

The SPADEs (Structured Parallel Discrete-Event Simulation) is a simulation library developed in two versions: one in C++ e the other in Java. It has a discrete event simulation kernel and does not provide visual animation during the simulation. Moreover, it provides a framework for simulations development. In this library, two classes define the user: process e resources that model the entities job e server. [18].

Despite of the SPADEs has many of the desirable characteristics, it has the following disadvantages: the programmer is required to use many methods and is guided by processes and not for the events generated during the simulation.

In relation of synchronization, like the two libraries showed in this section, the JDESLib allows sequential multithread synchronization. It has the necessary methods for the development of discrete event simulation tools and does not have the wait () e hold () methods that can be unnecessary or can generate synchronization errors. Moreover, the JDESLib support broadcast and multicast of an event.

The table one shows the most important characteristics of the implementation of the libraries cited in this section.

3. JDESLib

The parallel simulation library JDESLib is based on discrete event simulation. Its main object is to simplify the construction of discrete event simulation tools of parallel systems. It implements a next earliest event mechanism, in which the simulation timer is
advanced for the time of the event more eminent in the queue of events. It also manages the communication between the simulation entities. Each entity has a body() method, in which the actions for the event treatment are implemented (Fig. 1). The simSchedule() method is responsible for set the new event.

The main characteristics of the library are:

- The library has a standard method for simulation development.
- The body() method is composed of a selection structure, in which the programmer just use the simSchedule() method for build the entities behavior (Fig. 1). For this reason, the JSDESLib is a library of easy use.
- Wait calls (wait()) are not necessary, because the body() method of the entities receives automatically the events from the simulator, until the end of the simulation. This eliminates the development errors doing by the users.
- The queue of events are ordered for the clock time, so, the entities that do not interact with other entities, can schedule all its events at the same moment.
- An event can have more than one destination (multicast or broadcast).

3.1. Architecture

The JSDESLib architecture can be divided in two mainly components: simulator e entity. In the simulator there is a queue of events that are generated and received by the entities. In Fig. 2, we present the JSDESLib architecture, in which the entities create and receive the events throw the simulator. Moreover, the simulator controls the synchronization between the entities, by means of barrier. The simulator implements the events recycling for a best utilization of the memory. The entities are executed simultaneously by means of a parallel mechanism.

3.2. Implementation

The JSDESLib is composed by five classes: Simulator, SimEntity, SimBarrier, SimTags e SimEvent. In Figure 3, we present the JSDESLib class hierarchy. All these classes inherit the characteristics from Java Object Class. Furthermore, the Class SimEntity extends the Java Thread class.

The class Simulator manages the events queue (instances from SimEvent class) and, consequently, the interaction between the simulation entities. Each event represents an important action in a determined instant of time that changes the system state, being able to generate new events. An event must contents an identifier of the sender entity, the time instant in each the event will occur from the actual instant, an event identifier, the destination entities and data.

The entities must communicate with one another only by events, therefore, data and events are transmitted together. The events queue is sorted by time, in that the simulation clock is always forwarded to the time of the eminent event. The simulation ends when there are no events in queue or when a time of simulation, defined by the user, is reached. The class Simulator also realizes the events recycling, because the JVM garbage collector [13] tends to allocate more memory for the virtual machine instead of collect the non-utilized objects. Thus, many times, the simulation causes a memory overflow that interrupts the execution. The event recycling reuses objects from SimEvent class, which are more utilized, without the necessity of garbage collector intervention.

The SimEntity class represents a simulation entity. As an example of entity, we can cite a user, cluster or node. In the JSDESLib, each entity is an independent thread, making possible the use of parallelism. The SimBarrier class is responsible for the synchronism between the entities and the simulator. When the simulator executes, it removes the first event from the queue and unblock the destination entities. After that,
the simulator is blocked until that all the entities have been executed their event actions. The SimTags class defines the constants that are used on simulation, as the events tags.

4. Verification of JSDESLib

After the implementation of JSDESLib, it was tested and verified by object-oriented project techniques and default situations. As example situations, we simulated the event exchange between n entities using some communication patterns.

Figure 4 – Communication patterns: (a) One to all point to point; (b) One to all broadcast; (c) All to one; (d) All to all

As shown in Table 2, the distribution of the events was equal to the desired ones, considering each communication pattern and the events that were destined to an entity arrived in the right chronological order. It is important to see that the number of the event there is not relation with the arrival time. Beyond the described tests, the library was exhaustingly tested, in diverse situations, through the development of simulation tools.

5. Results

In this section, we present some simulation tools that were and are been developed with the JSDESLib. Moreover, we present the performance analysis of simple simulation executions of the JSDESLib.

5.1. Developed Tools

In this section, we present three tools developed by our group using the JSDESLib: ClusterSim, DSMSim and MPSim. The ClusterSim (Cluster Simulation Tool) [6][7][8] is a Java-based parallel discrete-event simulation tool for cluster computing. It supports visual modeling and simulation of clusters and their workloads for performance analysis, using JSDESLib.

As a cluster is composed of single or multi-processed nodes, parallel job schedulers, network topologies and technologies. A workload is represented by users that submit jobs composed of tasks described by probability distributions and their internal structure.

The architecture of the ClusterSim is divided in three layers: graphical environment, entities and core (Fig. 5). The first layer allows the modeling and simulation of clusters and their workloads by means of a graphical environment. Moreover, it provides statistical and performance data about each executed simulation. The second layer is composed of three entities: user, cluster and node. Those entities communicate by means of event exchange supported by the JSDESLib [8], a discrete-event simulation library, which is the simulation core of the ClusterSim.
The Fig. 5 shows the events exchange diagram of the ClusterSim, detailing the interaction among the user, cluster and node entities. To simplify the diagram, some classes were omitted. The ClusterSim was used to obtain the results on parallel job scheduling published in [7]. In despite of this tool, other tools are in development: DSMSim and MPSim.

The DSMSim (Distributed Shared Memory Simulator) is a distributed shared memory simulator that simulates: lazy and strong consistency models, coherence and access protocols. This simulation tool has 4 events: end of remote instruction, node message arrival, shared data updating and remote data sending.

The MPSimulator (Message Passing Simulator) is a message-passing simulator in SAN or LAN networks [9]. It is composed of two models: a network model containing communication patterns and network characteristics (ex: latency, bandwidth, etc.); and another model that simulates MPI point-to-point communication functions and other collective functions. The MPSimulator supports crossbar, bus and other customizable topologies. It has 3 events that represent the sending, arrival or routing of a message.

5.2. Performance Analysis of JSDESLib

To analyze the performance of JSDESLib, we made some tests varying the number of entities and events. Though these tests, we verified the advantage of JSDESLib multicast mechanism. Moreover, we observed the impact caused by the events recycling mechanism in the events sending and receiving.

As we see in Fig. 6, the events sending, using the multicast mechanism of JSDESLib, reduced the response time proportionally to the increasing of the number of entities.

Up to 32 entities, the sending of events, using the two mechanisms, is equivalent (they spent few milliseconds). In the case of point-to-point sending, for each entity, an event is generated. In the multicast, just an event is created and directed for all the entities. In the case of multicast sending, all of the receiving entities need to receive the event, generating competition in the reading of the event. But the overhead is very smaller compared to the generation and reading of several events.

In the Fig. 7, we observed the throughput (events/ms) for each number of events exchanged between two entities. We noticed that when reaching 512 events, the throughput reaches its maximum value, around 16 events per millisecond. For more than 512 events, the throughput decreases; therefore the saturation of the memory cache happens. When an entity sends the events, they are stored in the cache, but when that number of events increases a lot, the first events are substituted by the last events in the cache. Then, in the reception of the events for the receiving entity, the first events have to be fetched again in the memory. For this reason, when using a big number of events, the throughput decreases, tending to less than an event per millisecond.

6. Conclusions

According to our objectives, the JSDESLib library was proposed, developed, implemented and verified. Moreover, we verified its usage in the development of parallel systems simulators and we analyzed its performance.

JSDESLib is a library that is limited to provide basic mechanisms of discrete-event simulation and some features: simplicity and use-of-use; and a parallel and optimized implementation. The programmer needs to learn just the method simSchedule() to develop a simulation, because JSDESLib automates the entities synchronization and the events distribution. Besides the use of use, the implementation of JSDESLib is optimized, supporting parallelism between entities and
recycling of events, as shown in the results. Last, it has a mechanism of events multicasting that increases the performance for a big number of events.

The collector of garbage of JVM tends to allocate more memory for the virtual machine instead of collecting objects no more used. With that, a lot of times, the simulation reaches the limit of memory, interrupting the simulation. The recycling of events reuses objects of the class SimEvent, that are the more used, without the need of the garbage collector intervention.

As counterbalance to the simplicity and easy of use, it does not have several functionalities found in the related works as: visual animation of objects, distributed implementation, hierarchical simulation, etc. But it is important to emphasize, that some of those functionalities could be implemented without harming the easy of use of JSDESLib.

As future works, we will improve JSDESLib in many aspects: distributed implementation using sockets or Java RMI; usage of lazy consistency in the events execution and hierarchy simulation, allowing the utilization of multiple levels or event queues.

We would like to thanks PUMG, Informatics Institute, PPGEE, PropPG, PIBIC, CAPES and LSDC.

7. References


