Optimizing Conservative Parallel DE Simulation of WANs

B. Forgeau U. Killat
Department of Communication Networks
Technical University of Hamburg-Harburg - Germany
forgeau@tu-harburg.de killat@tu-harburg.de http://www.tu-harburg.de/et6

Abstract: A common problem encountered in the simulation of WANs is the huge amount of data that the simulator handles. Especially with a parallel discrete-events simulation scheme, this affects dramatically the performance because most of the data overloads the list of pending events. We propose a simple algorithm which increases the performance of parallel simulation by separating events according to their destination, and we verify the performance speedup on a simple parallel simulation model.

Keywords: Conservative PDES, Event-List, Communication Network, WAN, Long Delays

1 Introduction

Simulation is often the only realistic way to evaluate the performance of communication networks. Their growing size and complexity has increased the need for large-scale computer simulations, which use timestamped discrete-events (DE) as data entities. In such simulation systems, the physical system is divided into independent "Logical Processes" (LPs) that receive and produce data, and which are connected to each other via "links" to reproduce the mesh of physical components.

Despite the diversity of communication protocols, the different LPs typically used in WAN simulations, and the simulation models using these LPs, present similar characteristics. Unlike VLSI simulations, which consist of millions of simple components such as gates or buffers exchanging binary informations, communication networks models contain a relative small number of heavyweight LPs and huge amounts of complex data (called "packets") traveling over long links. These characteristics of long delays and high rates have a great impact on the performance of the simulation.

Another characteristic of communication networks is their inherent parallelism. This induces the idea of running large simulation scenarios on parallel computers, thus opening the issue of parallel discrete-events simulation. In order to solve the problem caused by the synchronization of processors running the same simulation, two different strategies have emerged: conservative and optimistic [1]. While the second scheme is a very active field of research, its requirements in terms of data storage makes it hardly practicable for the simulation of WANs. For this study, we assume that processors are synchronized according to a conservative algorithm similar to the ones found in [2, 3].

In this paper, we first analyze the specific issues of WANs simulation. Then we introduce the "Link Event-Lists" as an algorithm solving a typical problem of WAN simulations. Finally we verify its impact on a simple WAN model.

2 Problem Definition

In this first section, we show the influence of long-delayed links on the performance of sequential and parallel DE simulation.

2.1 Sequential Simulation

At first we need to describe the basics of DE scheduling.An event is defined as a tuple $e = (d, t, l)$, where d is the data carried by the event, t is the simulation time at which the event has to be processed (e.g. the timestamp), and l the input link of the LP that should process this event.

A DE simulator basically stores all the events in a globally stored "event-list", where they are sorted in increasing timestamp order. At each simulation step, the event e with the smallest timestamp is extracted from the list and put on the destination link l, which is an input link of the LP responsible for processing e. Then the activation function of the LP is called, which may produce new events that come back in the event-list via the output links.

We can consider the case of a "pure delay" LP within a whole simulation model. Such an LP has one input, one output, and its sole function is to increment by δ the timestamp of any event coming from its input link, before putting it on the output link. Each produced event must wait for processing in the event-list, while the events with smaller timestamps are processed. More precisely, the waiting time of an event is the difference between its creation time and its timestamp, which is δ. If we consider that links are monotonic, e.g. that events on a particular link l are produced in increasing timestamp order, links can be seen as FIFO queues. Thus we can apply Little's result to express the number of events with the same destination link l that are stored in the event-list: $n = \delta * \lambda$, where λ is the event-rate on l.

This calculation can be generalized to an output link of any LP producing events whose timestamps are bigger than their creation time. Such LPs are needed in WANs models, for example to simulate transmission delays over long wires or queuing delays. Because the simulator must sort the events, the per-

formance of the simulation depends on the size of the event-list, which in turn depends on the products "rate times delay" of every link.

2.2 Parallel Simulation

During the parallel execution of a DE simulation, each processor uses an own local event-list and runs an algorithm similar to the sequential one, with the difference that some of the events are exchanged with other processors as "messages" and that each processor handles only a part of the whole simulation mesh.

With a conservative algorithm, each processor needs, at least periodically, to recompute the value of the smallest timestamp with which a future event may be generated [2, 4]. Because this value depends on all the events stored in the event-list of a particular processor, the complexity of this recomputation, along with the parallel simulation performance, depends once again on the size of the event-list, what depends on rates and delays.

In this first section, we briefly explained how the performance of WANs simulation can be influenced by links with big rates and long delays. In the next section, we present a simple optimization which is particularly well adapted to parallel simulation.

3 Link Event-Lists

3.1 LELs algorithm

As we have seen, the number of events with the same destination link l, that are stored in the global event-list, is mainly influenced by the product of "rate times delay". But the first purpose of the global event-list in the DE algorithm is to find one event for the next LP to be scheduled. The remaining events with bigger timestamps are also stored in the global event-queue but not used for scheduling purpose.

The time spent for sorting the events in the event-list has a great impact on the performance of the simulation. Many schemes have been proposed to address this issue globally. These algorithms consider the event-list as a given set of independent prioritized items. Our approach is to reduce the size of the global event-list by distributing the events on Link Event-Lists (LELs).

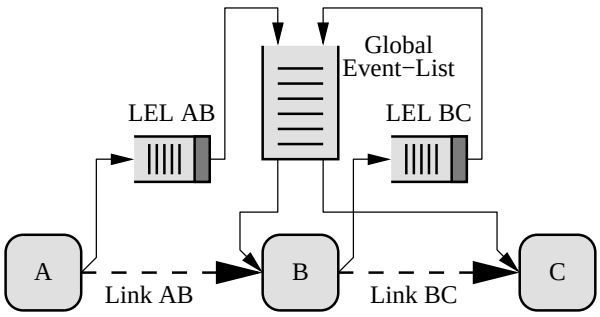

Figure 1: Link Event-Lists

Each link has an event-list comparable to the global one, and a variable *size* initialized to 0. Figure 1 shows a representation of the

LELs for two links. The events with the same destination link l are stored on the LEL corresponding to l, except the one with the smallest timestamp, which must be stored in the global event-list of the processor, in order to be used for scheduling. The *Enqueue* subfunction is given in C++-like pseudocode:

```
void Enqueue(Event e) {
  if(link(e).size==0)
    globalQueue.enqueue(e);
  else
    link(e).queue.enqueue(e);
  link(e).size++;
}
```

It is supposed that the links are monotonic, then the $link(e).queue.enqueue$ function has a complexity of $o(1)$, for the events just need to be put at the end of the LEL without sorting. The complexity of the *enqueue* function is then the complexity of $globalQueue.enqueue$.

The monotonicity hypothesis is a needed restriction for conservative PDES but does not limit the design possibilities of most LPs. LELs may be also used when the links are not monotonic, but then the *Enqueue* function does need a real sort on the LEL. The *Dequeue* function is given as:

```
Event Dequeue() {
  Event e = globalQueue.dequeue();
  if(linke(e).size>1)
    globalQueue.enqueue(
       link(e).queue.dequeue());
  link(e).size--;
  return e;
}
```

Because a *dequeue* function on an event-list has a complexity of $o(1)$, the complexity of *Dequeue* is the same as the one of $globalQueue.enqueue$. But with this algorithm, at most one event per link is stored in the global event-list. Thus the *enqueue* function on the global event-list must do at most n_{links} operations, where n_{links} is the number of links in the simulation model. As a result, the complexity of *Dequeue* is also $o(1)$, if we take the total number of events in the simulated system as parameter.

The obvious advantage of the LELs is that the maximal number of events contained in the global event-list is limited by the number of links in the simulation model. More precisely, for each link l with a product rate times delay bigger than 1, what means that the LEL contains events, the only event with destination l in the global event-list is the one with the smallest timestamp.

3.2 Simulation Results

We verify the correctness and the performance of the LELs on a typical WAN model with long-delayed, intra-processor and inter-processors links. The model is designed in such a way that the rates do not depend on the delays. We simulate the model over a wide range of delays with and without LELs.

At first we can observe on Figure sim:size the size of a local event-list on a particular processor. As expected, the size increases as the delay increases without LELs ("linear"), and is limited with LELs.

We then observe on Figure sim:execpar the execution time of the whole simulation. We

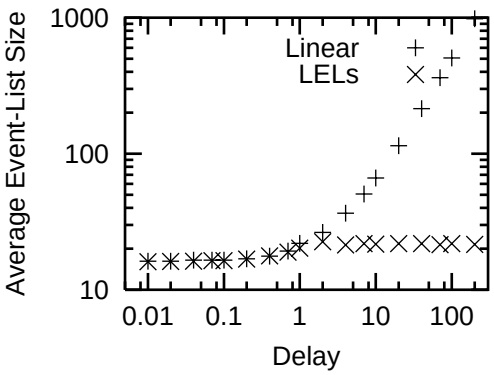

Figure 2: Size of the events-list

see a difference of behavior for long and short delays. On the left side, we can observe the effect of null-messages retransmissions [2]. On the right side, we see that the LELs make the performance stable for long delays, what is not the case with the normal scheduling algorithm.

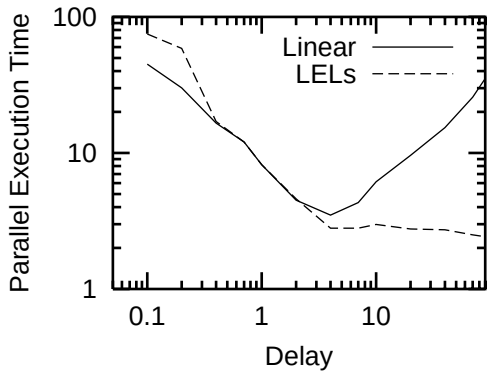

Figure 3: Execution time for various delays

In the context of WANs simulation, where the event-list is typically overloaded with events coming from long-delayed links, the LELs, which can be implemented apart from any other optimization scheme, provide a significant speedup.

4 Conclusion

This paper deals with parallel DE simulation of communication networks. We first introduced the influence of the product rate times delay on the performance of parallel and sequential simulation. We then introduced the Link Event-Lists as an optimized scheduling algorithm and verified its performance in a parallel simulation.

References

[1] David M. Nicol and Richard M. Fujimoto. Parallel simulation today. *Annals of Operations Research*, (53):249–285, 1994.

[2] J. Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1):39–65, 1986.

[3] Kenneth R. Wood and Stephen J. Turner. A generalized carrier-null method for conservative parallel simulation. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, 1994.

[4] E. Naroska and Uwe Schwiegelshohn. Conservative parallel simulation of a large number of processes. *Simulation*, (72):150–162, 1999.