How can we study Large and Complex Systems

C.D. Pham H. Brunst S. Fdida

Laboratoire LIP6
Université Pierre et Marie Curie
4 place Jussieu 75252 Paris Cedex 05
e-mail : Cong-Duc.Pham@lip6.fr

Abstract

Simulation and performance evaluation have been tightly linked together over the past years. If traditional simulation techniques are now widespreaded and well-understood by a large number of users, this is not the case for distributed simulation used only by a few researchers. In this paper, we demonstrate that parallel and distributed simulation techniques can be used to simulate large and complex models of ATM networks for performance evaluation. The CSAM tool has been developed for that purpose. Our test model consists of a large ATM network with a load-balanced routing scheme that relies on link cost functions to distribute the load.

1. Introduction

Performance evaluation consists of the analysis and the design of systems. Two main methods are commonly used to perform this task: the analytical approach and the simulation approach. Analytical equations provide an elegant way to model and obtain stable results but they often require simplifications that reduce the usefulness of the study. Therefore, simulation and performance evaluation have been tightly linked together over the past years. This successful cooperation comes from the ability for simulations to handle complex systems while other methods fail. Each solution, analytical or simulation, however has its own disadvantages and the flexibility of simulation is paid by a large amount of computation time. In many cases sequential simulation has now reached its limits.

Distributed simulation techniques are very promising for bringing the power available in parallel machines to the simulation task to reduce the simulation time. However, due to the lack of flexible and powerful tools these techniques are still unknown to a large part of discrete-event simulation users. People are skeptical on the efficiency of parallel simulation and are also afraid to use it. This situation has partly been created by the parallel simulation community: over these last 2 decades, algorithms have been proposed, test cases have been studied, optimizations have been done, but yet few real tools or few real demonstrations that parallel simulation can be used now on operational models. Parallel simulation is still regarded as a strange tool used by people working on the subject without real problems to solve! It is true that parallel simulation is more complex than sequential simulation. For parallel simulation, the speedup depends on a lot of parameters like for example the synchronization method used. We must concede that each time somebody asks "*what speedup can I obtain for my model and what is the cost for making it working in parallel*", the answer is usually "*it depends on a lot of factors...*". This is one of the most important drawback of parallel simulation: the lack of generality. In most cases, people working on parallel simulation use quite simple test case models and focus mainly on simulation algorithms. On the other side, people doing performance evaluation do not really want to cope with parallel simulation and rather focus on the development of their models. However, things are changing rapidly: many projects are now designed with respect to parallel simulation techniques and people from both sides of the problem are working together.

Coming from the network community, we are more sensitive to performance evaluation of communication networks. As the demand for broadband communication is becoming larger and larger it is very urging to offer high-speed backbones to provide sufficient bandwidth for a various number of multimedia applications such as video-conferencing and TV distribution. Although simulations can be carried for small or specific parts of the network (e.g. switch element for switch architecture design), the challenge is now to simulate it in one piece. In an environment where both speed and bandwidth are increased, problems like congestion avoidance, traffic control and optimal routing algorithms should be addressed first to ensure the best behavior of the network. Early works by Earnshaw and Hind [4], Mouftah and Sturgeon [8] have already investigated the communication network distributed simulation problem but they are limited to the simulation of simple transfer of packet/cell between nodes. More recently, the joint TeleSim project between the university of Calgary and the university of Waikaito aims to develop high-fidelity ATM (Asynchronous Transfer Mode) network traffic models and a network simulation testbed called the ATM-TN [9]. At the Parallel Computing Laboratory of the University of California, Los Angeles, the DOMAINS project wants to demonstrate the effective use of parallel simulation techniques for the simulation of large-scale heterogeneous mobile systems. The PARSEC environment is used to perform this task. Also, the group at Georgia Tech works on the S3 project (Scalable Self-organizing Simulation) to enable the simulation of large-scale telecommunication with a focus on Personal Communication Systems [2].

We started our project 4 years ago and the goal was to study complex ATM network models with parallel simulation methods. One application is the simulation of the routing algorithm described in [6] with connection management and dynamic routing facilities. Tests are performed to study the recovery on failure properties of the routing algorithm under extreme conditions, e.g. where failures occur more likely than normaly assumed. The simulation of a routing algorithm implies the simulation of (i) the mechanism that handles the construction and update (on cost changes such as link failures) of the routing tables and (ii) the flow of cells transported by the network. Performing at the cell-level gives the possibility to inspect the smallest interactions between network components but is very time-consuming. We developed the CSAM tool (Conservative Simulator for ATM network Model) to reduce that cost.

The paper is organized as follows: Section 2 presents some background information on distributed simulation techniques and our rationales for the CSAM tool. Section 3 describes our ATM network model test case. Performance results extracted from the model are presented in Section 4 with the speedup obtained on a Cray T3E. Conclusions are then given in Section 5.

2. Distributed simulation background

Parallel and distributed simulation can either be *synchronous* or *asynchronous*. Time-step approaches belong to the first category. In our case, they have two main disavantages. First, a synchronous simulator would constraint the time unit to the smallest time increment that exists in the whole system. When a large variety of time increment exists (i.e. not homogeneous transmission rates) this approach would lead to an overwhelming overhead. Second, a time-step approach is difficult to distribute because the definition of a global clock is hard. Therefore we focus mainly on the *discrete event* approach where changes in the system occur at discrete points in the simulated time with the model stepping from one point to another according to the progress of event processing. The distributed approach usually assumes that the system to be simulated can be spatially partitioned into disjoint sub-systems. Each sub-system is simulated by an associated *Logical Process* (LP) on a dedicated processor. To model the interactions that may exist in the real system messages are exchanged and time stamped by the sending LP according to its *Local Virtual Time* (LVT). At the receiving side, they are put in a local *Future Event List* (FEL) to be processed. Other methods to parallelize a simulation exist but their applications are limited [5].

Care must be taken when distributing virtual clocks and event lists among physical processors to ensure that the distributed simulation is still *correct* and produce the same output when compared to a sequential one. In particular, the *causality constraints* that represent the logical evolution of time must be maintained. Unfortunately since the different LPs may advance at different rates synchronization problems are likely to occur. A *causality error* happens when a message arrives at a receiving LP and is outdated, or old, according to its LVT. These problems arise the need of explicit synchronization mechanisms that are traditionally classified in two main categories: *conservative* and *optimistic*.

The conservative approach [3] only processes events that can definitely be considered safe, i.e. that processing the event would not result in further causality violations. Safety is guaranteed by forcing the LP to block if bad decisions could be taken. The advantage is to obtain a correct simulation at any time. Unfortunately, this scheme introduces deadlocks that must be avoided by sending *null-messages*. These messages rely on a *lookahead* ability which allows for an artificial propagation of the simulation time. On the other side, optimistic approaches [7] do not search for safety but provisions are made to *roll back* to an earlier coherent state when they occur. Periodic check-pointing and *anti-messages* to cancel bad computations are then needed as a counterpart of more freedom. In addition, a *Global Virtual Time* (GVT) is required to monitor the simulation progress and to reclaim memory used by obsolete information. In general, an optimistic approach is much more complex than a conservative one but can transparently exploit the parallelism in the system without the explicit determination of a lookahead property.

2.1. Rationales for the CSAM tool

There are many obstacles when developing a general purpose distributed simulation tool. For instance, experimental studies have shown that neither conservative nor optimistic approaches can be used for a large variety of applications. The wish to have a general tool usually requires the integration of a large variety of mechanisms to handle as much as possible cases. This solution dramatically increases the complexity of the tool. In addition, it is most likely that the user would still need to select the appropriate underlying synchronization mechanisms. To remove technical consideration for the end-user, we have to restrict the range of target applications to a given class of applications for which it is possible to predict which synchronization method would be more suitable, and focus on it. This is the approach we chose: we focus on communication network models and use the conservative synchronization rules in CSAM. This choice is dictated by the following observations. First, communication network models present a very high level of message synchronization because their basic function is to transfer packets from one point to another. This would produce a lot of time errors if an optimistic scheduling is used. And second, since the transfer function is quite simple, they also have a low computation granularity that would propagate bad computations very quickly.

A distributed simulation tool can be proposed in 3 different manners: (*i*) as a completly integrated tool, (*ii*) as a special language, or (*iii*) as a library for a general language. The first solution is a long term and costly solution but is highly convivial. The second one is practical and suitable for small models. However, as the cost for the model development increases as its size and complexity increases, it is preferable that the model could be later used as the real implementation with very few modifications. For example, the simulation of communication protocols is a field where reusability of the model is highly welcome. Therefore, the use of a language dedicated only to simulation can be a drawback. The last solution is a tradeoff between the integrated tool and the language. Using a well-known language can increase the reusability as well as decreasing the time needed to learn the language.

Our primary goal was to develop an environment with practical relevance for simulating ATM network models at the cell-level. CSAM is an environment for developing ATM network models that can be run on a parallel architecture using the conservative method (in addition to sequential execution). Our approach lies somewhere between the integrated tool and the library. CSAM is written in C++ and provides communication primitives as well as base class models for traffic sources, switches and links. For the end-user the definition of new model consists of defining and implementing a set of classes in a hierarchical manner to maximize the reuse of sub-models. The complexity of distributed simulation techniques is completely hidden to the user. Even though the simulator's kernel is theoretically capable of simulating all types of applications, several modules in CSAM are designed for communication network to reduce the cost of new model developments. To support a different class of applications, it is necessary to adapt the environment.

2.2. Amount of parallelism and speedup

The goal of distributed simulation is to reduce the simulation time when compared to a sequential execution. It is obvious that speedup can only be obtained if a sufficient amount of parallelism exists in the real system. Therefore pathological cases where the behavior of the system is extremely sequential would not be parallelized successfully. Fortunately, the replicated nature and the spatial distribution of communication networks provide a high potential of parallelism. In addition, increasing the size also increases the number of events at the same simulated time and therefore increases the parallelism as well. We think that the lack of parallelism in ATM networks will not be often raised because pathological cases are either unrealistic or not really interesting to study.

However, the degree of parallelism that exists in the real system does not imply the same degree of parallelism in the simulation of the model. This phenomenon has been previously shown in [10] for queuing network models. In realistic network model for instance, it is most likely that some part of the network would be more loaded than other parts creating bottlenecks in the model. Under the general assumption that all physical processors have the same computational power this load imbalance problem limits the speedup of the simulation. Therefore *model parallelism* should be distinguished from real parallelism.

Achieving good speedup is a difficult task. It is bounded by (*i*) the amount of inherent model parallelism and (*ii*) by the overhead introduced by synchronization algorithms. Improving the former is possible by mapping the system into LPs in an optimal fashion. For instance, it may be possible to combine several network nodes together (relatively easy) or to split a congestioned node into several LPs (much more difficult). For the second, optimization and new mechanisms for synchronization algorithms can provide significant improvement in the performance of the simulator. This is still a very active area of research.

2.3. Lookahead

Lookahead refers to the ability of predicting the future—what can happen or what can not happen—based on application specific knowledge. If an LP knows all events that can occur up to time t and if it can predict all events that will be generated up to time $t + L$, then it is said to have a lookahead of L. Good lookahead is essential for conservative methods to perform well, especially when a deadlock avoidance mechanism is used, whereas an optimistic method does not really need lookahead.

Several delays can be taken for lookahead purpose in an ATM network model. For instance, the commutation time t_c to switch a cell from an input port to an output port, the waiting time t_w of a cell at an output port, the transmission time t_t and the propagation time t_p of cells on a link can be used. t_c, t_t are not significant when compared to t_p since the former are typically expressed in hundreds of μs whereas the latter is usually expressed in ms. When the network is lightly loaded, t_w is less significant than t_p and the lookahead mainly relies on t_p. When the load increases, t_w can be larger than t_p in some extreme cases (congestion). Trying to use t_w usually requires FCFS scheduling that would seriously limit the range of applications. In network models, most of congestion avoidance mechanisms and rate control algorithms are based on more complex scheduling schemes. We choose to use only the lookahead provided by the link propagation time for more freedom on the scheduling policy.

3. Description of the ATM network model

The ATM technology is intended to provide large throughput and high-speed transfer between nodes by using small cells, instead of packets, and switching components linked by optical fiber. Nowadays, ATM can be found in Local Area Networks (LANs) and starts to appear in Metropolitan Area Networks. The next step is to develop it for large scale configurations. We propose to study the fictive topology illustrated by Fig. 1. This 78-switch network model has been obtained by linking three 26-switch networks (bold links in figure 1) used in the ATM Forum specification on the PNNI (Private Network-to-Network Interface) routing algorithm [1]. We added 10 sources (S_0 to S_6 and S_4^1 to S_6^1) and 7 sinks (D_0 to D_6) to model the traffic of cells at the network interfaces in each sub-network. Each source sets up a permanent Constant Bit Rate (CBR) connection to the corresponding sink (subscript number) until the simulation ends. We also added 3 inter sub-network traffic sources called A, B and C. These sources are multi-destination sources that repeatedly choose one destination among the 21 destinations for a temporary connection. The sources/sinks represent ATM end-systems that are normally ATM multiplexers/demultiplexers of traffic from/to LANs or single host. Very generally, we assume that cells are generated by traffic sources and sent to the corresponding sinks by intermediate ATM switches. Signaling is also modeled: traffic sources execute a `setup` procedure with their corresponding destination prior to any data transfer[1]. All along the path of the connection, a new entry is added to the connection table of each intermediate ATM switch to register the connection identifier and the output link for this new connection. During the setup a switch can decide to reject the call if the resources available are not sufficient (`release` message). Upon reception of the `setup` message, the final destination can either send back a `connect` message or a `release` message. At the moment, the admission control works as follows: each CBR connection indicates a *Peak Cell Rate* (PCR) that is reserved in each output link along the way to the sink. If the sum of all PCRs allocated at a given link has exhausted the link capacity an additional call can not be routed via this link. A call is

[1] We assume only one connection at a time per traffic source.

rejected at a switch if no output link with sufficient resources can be found.

Even if the topology is fictive, it is general enough to represent most of real topologies. The internal delays of switch and link models can be instantiated to reproduce the average behavior of real systems. For this study, the switch model is non-blocking with buffers located at the output ports. The average delay for switching a cell from an input port to an output port depends on the throughput of the switch. Each output port contains 2 queues for high and low priority traffic. Scheduling is done in a FIFO manner within each queue. Links are characterized by a throughput and a propagation delay. In the network model, permanent CBR connections model the long term load in the network while temporary connections model the cross-traffic.

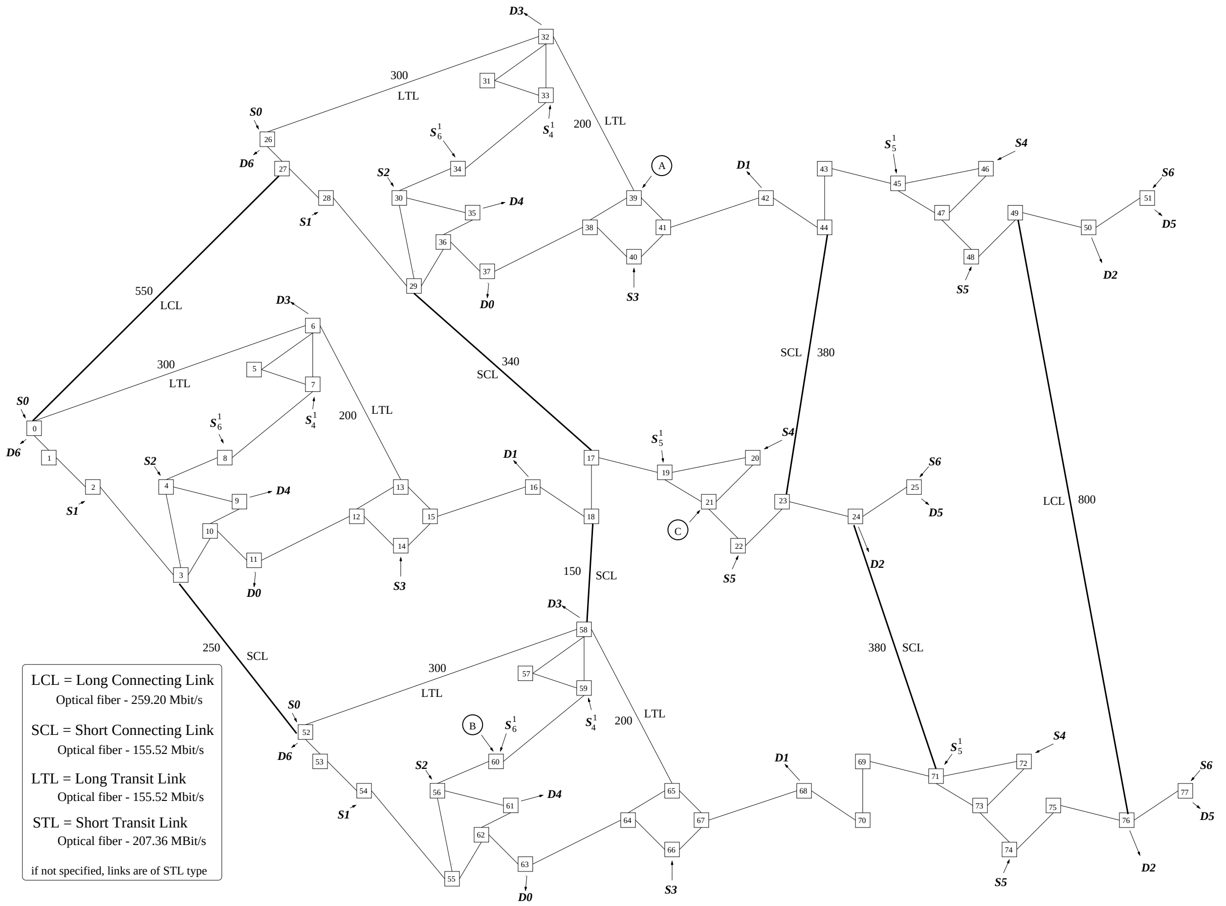

Figure 1. Network model with 78 switches.

3.1. The routing algorithm, link cost and routing functions

A routing algorithm consists of calculating the best path for the packets/cells from a source to a sink. This calculation is done by considering either implicit criteria such as the number of hops or explicit ones given by the user. In this paper, we are interested in the routing algorithm proposed by Jaffe and Moss in [6]. This algorithm belongs to the *vector-distance* category and by the authors is said to have a quick recovery from cost changes property (this is the main reason for the choice of this algorithm). Transient loops are avoided by a synchronized exchange of cost increase information and acknowledgments (nodes may either be in a *freeze* state or in a *normal* state).

Routing functions are used to determine in real time the cost of a given link based on its type and instantaneous load. The type of a link varies with its maximum throughput, its length, its function in the network and its access medium (e.g. twisted pair, coaxial, optical fiber, radio...). In our model we will assume that optical fibers are available everywhere in the network. Four types are then defined in the model: Short Transit Link (STL) and Long Transit Link (LTL) for the interconnection of switches in the same sub-network, Short Connecting Link (SCL) and Long Connecting Link (LCL) for inter sub-network connections. A link is said to be long if its length is greater than 150km for a connecting link and 50km for a transit link. For each type of link, a routing function is defined to calculate the cost of the link. These functions work with 4 thresholds and have a generic shape as illustrated by Fig. 2.

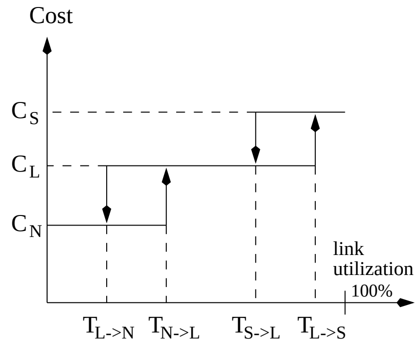

Figure 2. The generic routing function.

C_S, C_L and C_N are respectively the cost for saturated, loaded and normal state. $T_{L\rightarrow N}, T_{N\rightarrow L}$, $T_{S\rightarrow L}$ and $T_{L\rightarrow S}$ are respectively the thresholds, in percentage of the effective throughput (ET), for going from loaded state to normal state, from normal state to loaded state, from saturated state to loaded state and from loaded state to saturated state. The values for these parameters are summarized in table 1.

	STL	LTL	SCL	LCL
C_S	100	95	97	97
C_L	65	65	55	55
C_N	15	20	25	35
$T_{L\rightarrow N}$	22%	22%	55%	55%
$T_{N\rightarrow L}$	39%	50%	72%	72%
$T_{S\rightarrow L}$	57%	60%	80%	80%
$T_{L\rightarrow S}$	95%	95%	95%	95%
Throughput	207.36	155.52	155.52	259.20
ET	168.38	126.28	126.28	210.47

Table 1. Routing function parameters.

Connecting links are triggered to a loaded state at a higher percentage of utilization than transit links. Changing the thresholds gives the possibility to load balance the connections on a larger number of links. For instance, to avoid links with no connections, the link's threshold from normal to loaded state can be set very low so that its cost is increased as soon as one connection is established. Therefore, next connections would have to take free links.

The effective throughput is obtained by taking 81.2% of the maximum throughput of the link. This percentage represents (i) the losses due to the header of the cells (-10%), (ii) -5% in order to not saturate the FIFO buffers and (iii) -5% reserved for the connection establishment and reconfiguration. Hence we have $0.9 * 0.95 * 0.95 = 0.812$. At the moment, the calculation of the link cost is based on the PCR given during the connection phase. Each time a new connection is accepted and routed on a given link, the cost (and the state) of the link may change due to the thresholds of the routing function. In the case of a change, the switch sends cost update information to its neighbors according to the Jaffe and Moss algorithm. The routing table may then change but new cheapest links will only be taken into account for new connections. Multi-destination sources in the network model will cause a lot of link cost changes resulting in very frequent updates of routing tables.

3.2. Why is it difficult to obtain speedup?

From the distributed simulation perspective, this model has several challenging properties. First, the simulation of a shortest-path routing algorithm on a general topology creates disparities between links and many of them may not be often used (links depicted in the figure are bi-directional even if for sake of simplicity a single link has been drawn). Therefore the conservative simulator has to generate a lot of null-messages to propagate the simulation time on these channels. We think that this situation is a highly realistic one. The second property is a consequence of the first: bottlenecks can appear in the simulation by concentrating a large part of the messages through a given processor. Such behavior also appears in the real system but can be overcome by upgrading some network components. For the distributed simulation, in most cases, it will be impossible to increase the speed of the processor that simulates the overloaded part of the network. Third, in addition to permanent connections set up by static sources, dynamic sources with random destinations introduce frequent changes to the link's load resulting in complex load-balance patterns. Finally, simulating high-speed networks at the cell-level produces a very low granularity. One second of the real system may require the simulation of millions of events. In distributed simulation, depending on the number of processors used, these events must be sent over the inter-communication network with a lot of communication overhead.

4. Experimentations

The CSAM simulator offers a unified framework for parallel and sequential implementations: the same source code can either be run sequentially or in a distributed manner. This method has the advantage to allow fair speedup measures. The simulations are run on a Cray T3E provided by the *Institut du Développement et des Ressources en Informatique Scientifique* (IDRIS) at Orsay, France. Interprocess communication uses the native SHMEM library. The Cray T3E consists of 256 processing nodes (PNs) running in dedicated mode (one job) and interconnected by a very low latency 3D torus network. Traffic source and sinks connected to a switch are never separated and reside on the same LP.

4.1. Model's parameters and test cases

The simulation time of the simulator is expressed in time units. The corresponding real time value can be obtained by defining a time unit as follows: it is the time needed to switch a cell from an input port to an output port given a base throughput T_b. For the network model depicted in Fig. 1, the base throughput T_b is set to the throughput of the most rapid link (259.20 Mbit/s) in order to avoid time increments less than 1 time unit. In this case, 1 time unit represents $\frac{8*53}{259.2*1024*1024} = 1.56\mu s$ of real time (8 bits per byte and 53 bytes per ATM cells). All switches process cells at the rate of 155.52 Mbit/s per input link. STL links have a throughput of 207.36 Mbit/s, LTL and SCL links have a throughput of 155.52 Mbit/s and finally, LCL links have a throughput of 259.20 Mbit/s. The propagation delay for links is set to 125 time units for all SCL links. For the other links a different value is indicated in figure 1. If 1 time unit represents 1.56 μs, then 100 time units represents approximately 32km of optical fiber when 2/3 of light-speed is assumed for the propagation. The initial load for all links is set to zero resulting to an initial cost equal to the normal cost C_N indicated in table 1. CBR sources from 0 to 3 in each sub-network set up 40Mbit/s connections. The remaining permanent CBR sources send cells at the rate of 48Mbit/s. Temporary connections are characterized by a connection time of 25,000 time units, an idle time between 2 connections of 500 time units, and a rate of 50Mbit/s. With these parameters and the routing parameters of table 1, an STL link can only carry 2 connections before switching to a loaded state. 2 more connections and the STL link becomes saturated.

We simulate the model for 200,000 time units that represent approximately 0.31s of the real system. More that 18 millions of events have been simulated for approximately 6 millions of cell exchanges.

4.2. Performance issues of the model

In this section, we present some performance criterion that can be extracted from the model. They are not intended to represent any deep performance investigation but rather examples of what can be done. Figure 3 shows the time needed to find optimal paths to 50%, 80% and 100% of the possible destinations.

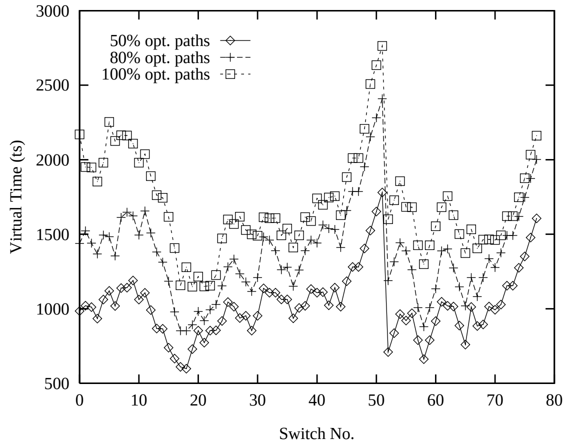

Figure 3. Time to find optimal paths.

The figure shows that the time is proportional to the switch position relatively to the others. A switch in the centre of the network, with the shortest distance to its neighbors, finds the optimal paths much quicker than a switch at the "edge" of the network (e.g. switch 52). This result was not astonishing as the algorithm complexity was reported to be proportional to the distance (in our case the link delay) of the longest possible route between network nodes. For the recovery after a link failure we obtained similar results.

During intensive tests of the routing algorithm we sometimes encountered the situation that a network node forever stayed in freeze state for a certain destination. After long debugging sessions in which we clarified that our implementation corresponded exactly to the specification by Jaffe and Moss, we found out that a special condition was simply forgotten by the authors. In fact, a problem occurs when a network node becomes completely isolated (i.e. looses connectivity to all neighbors). Since the neighboring network nodes will not send any acknowledge after the links have recovered, the destination remains frozen forever. Our solution to this problem is a simple check which un-freezes all destinations in the case the node becomes isolated.

Figure 4 shows the link utilization (impulses) and the link cost of the STL link from switch 17 to switch 18. We can see from the figure that even if the link utilization decreases to 30% for some moment, the link cost remains at 65 (for a maximum of 100). This result shows that the thresholds defined in table 1 are somehow too pessimistic. Deeper investigations can find optimal threshold values for a better link utilization.

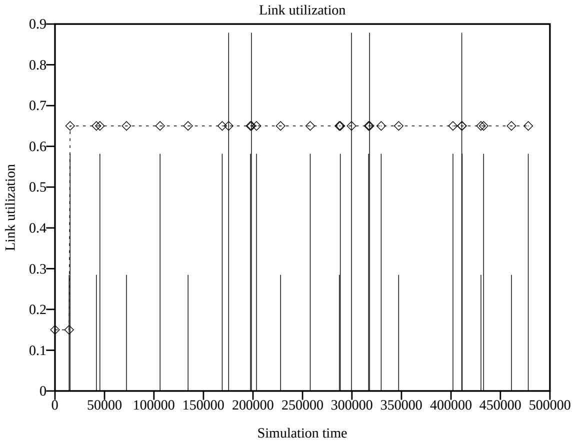

Figure 4. Evolution of the link's cost

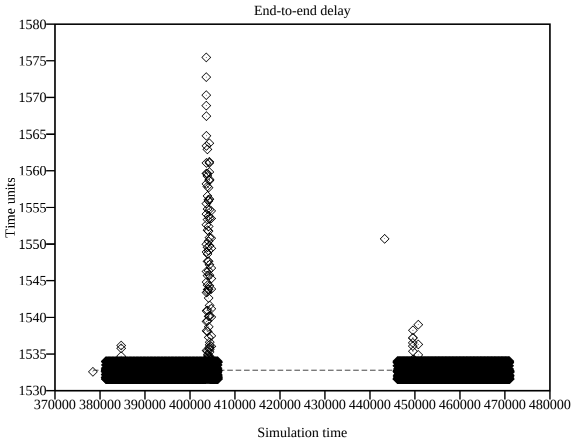

Figure 5. End-to-end delay.

Figure 5 shows the end-to-end delays of cells from source C to sink D_6 of switch 0. The mean end-to-end delay is about 1532 time units while some cells encounter a larger delay due the cross-traffic generated by another source. Working at the cell-level provides the ability to investigate the smallest interactions between the network components.

4.3. Efficiency of the parallel simulator

The speed of a parallel program compared to a sequential implementation is somewhat difficult to measure, although simple at first glance. In principle, a simple execution of the parallel and sequential implementation with a subsequent comparison of the execution times should be fine. This approach is correct and fair as long as both versions have been optimized as much as possible. For our tests we used a sequential version of the simulator with all communication overhead removed. The object oriented approach we used made it very easy to generate such a sequential version. The parallel version of the simulator in fact inherits all classes from this sequential version and adds some synchronization facilities. Figure 6 shows the execution time for the sequential version and for 6 mappings of the distributed simulation. Table 2 presents the speedup and the efficiency (speedup/nproc) as the number of processors *nproc* is varied.

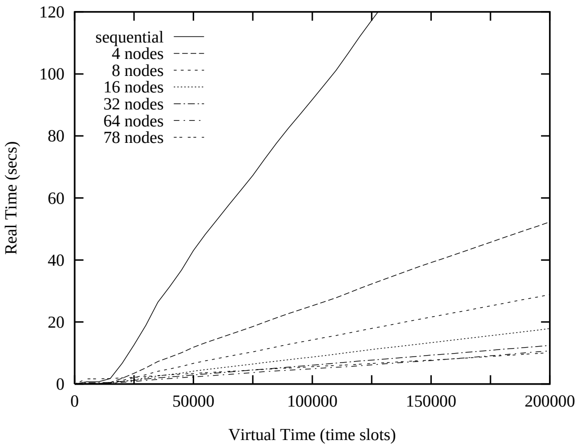

Figure 6. Execution times.

nproc	*ex. time*	*speedup*	*efficiency*
1	189	–	1.0
4	52	3.63	**0.91**
8	29	6.52	0.81
16	18	10.5	0.66
32	12	15.75	0.49
64	11	17.18	0.26
78	11	17.18	0.22

Table 2. Exec. times, speedup and efficiency.

The speedups are almost linear with 4 and 8 processors. With more than 8 processors, the speedups still continue to increase, but asymptotically. A maximum speedup of 17.18 can be achieved with 64 and 78 processors, the latter case representing a simple one switch to one processor mapping. Using so much processors is an extreme case and no significant improvements have been obtained with more than 32 processors. With so many processors, the proportion of messages that go off the local processor dramatically increases and synchronization overhead limits the speedup. The fact that the speedup does not decrease either must be related to the very low cost of the Cray's communication network. Furthermore the SHMEM message exchange functions provide a 4 time lower latency than high level packages like PVM.

The most difficult problem we had to face was the load-balancing issue. For large models, it is important to map the LPs on the processors in an optimal manner. At the moment, the initial partitioning uses a neighbor aggregation method. Practically, as systems with a large number of processors and very low communication cost are not accessible to the majority of researchers nowadays, parallel simulations are likely to be executed on a smaller number of processors with higher communication costs. In this case, it is certainly better to use a small number of processors with optimal partitioning and load-balancing features.

5. Conclusion

The distributed simulation research at the LIP6 laboratory aims to enable the performance evaluation of large-scale and highly complex network models. Focusing on ATM network models, the CSAM environment allows the user to develop network models for parallel execution with no particular concern about parallel simulation.

The application studied in this paper consisted of a 78-switch network with thresholded routing functions based on link costs. Connection management and admission control facilities are also modeled. The model has been developed by using base classes for traffic sources, switches and links provided by the CSAM environment. No particular considerations have been made for parallel execution on a Cray T3E. We obtained speedup ranging from 3.63 to 17.18 depending on how many processors were used. For 4 and 8 processors, the speedups are nearly linear and do not improve significantly with more than 32 processors. The only problem we had to face was for the load-balancing when a large number of processors are used. We plan to add load-balancing features into the CSAM kernel.

Providing an environment for a given class of applications favorises the use of distributed simulation to solve problems that traditionally would have been studied with sequential tools in a much longer time. Furthermore, distributed simulation can also simply enable the study of very large configurations that would have been impossible to handle in sequential.

References

[1] ATM Forum, PNNI v1.0 Specification

[2] C. D. Carothers et al., "Distributed Simulation of Large Scale PCS Networks", Proceedings of the MASCOT'94, 2-6.

[3] Chandy, K. M., Misra, J. : Distribution Simulation: A Case Study in Design and Verification of Distributed Programs. Trans. on Soft. Eng., **5**(5), 440–452.

[4] Earnshaw, R. W., Hind, A. : A Parallel Simulator for Performance Modeling of Broadband Telecommunication Networks. Proc. of the WCS'92, 1365–1373.

[5] Ferscha, A., Tripathi, S. K. : Parallel and Distributed Simulation of Discrete Event Systems. CR-TR-3366, Dept. of Comp. Science, Univ. of Maryland.

[6] Jaffe, J. M., Moss, F. H. : A Responsive Distributed Routing Algorithm for Computer Networks. IEEE Trans. on Comm., **30**(7), 1758–1762.

[7] Jefferson, D. R. : Virtual Time. ACM Trans. on Prog. Lang. and Sys., **7**(3) (July 1985) 405–425.

[8] Mouftah, T., Sturgeon, R. T. : Distributed Discrete Event Simulation for Communications Networks. IEEE JSAC, **8**(9), 1723–1734.

[9] B. Unger et al., "A High Fidelity ATM Traffic and Network Simulation", Proc. of the WSC'95, 996–1003.

[10] D. B. Wagner and E. D. Lazowska, "Parallel Simulation of Queuing Networks: Limitations and Potentials", Proc. of the ACM Sigmetrics 1989, 146–155.