Parallel Simulation of a High Speed LAN

Rönnngren R., Rajaei H., Popescu A., Liljenstam M., Ismailov Y., and Ayani R.

Department of Teleinformatics
Royal Institute of Technology (KTH)
Stockholm, Sweden

ABSTRACT:

Simulation of communication networks is often time consuming. Parallel simulation may be used to reduce the execution time of such a simulator. However, efficient modeling and simulation of such complex applications is not a trivial task. In this paper, we discuss the results of simulating a multigigabit/s network using both an optimistic and a conservative scheme. Our experimental result on a shared memory multiprocessor indicates that the conservative approach is superior to the optimistic one for this application.

1.0 INTRODUCTION

With the growing complexity of communication networks it becomes increasingly difficult to conduct performance evaluation by strictly analytical means. It is often necessary to turn to simulation techniques to obtain adequate information on functionality and performance of communication networks and protocols.

The traditional sequential simulation techniques and tools would in many cases lead to excessive execution time (in the order of days and even weeks). Execution of such time consuming simulation programs on parallel architectures have been an active research topic in the recent years. Two main categories of parallel discrete event simulation (PDES), conservative and optimistic, schemes have been investigated by several researchers, e.g. see [1].

In this paper, we present the results of simulating a multigigabit local area network, called SUPERLAN, using both a conservative and an optimistic approach. Our experimental results on a shared memory multiprocessor show that the conservative approach outperforms the optimistic one for this specific application. This work illustrates a case where the conservative scheme is preferable. The special features of the application under study (as discussed in Section 2) and the state saving policy employed in our Time Warp implementation are among the main factors contributing to the reported results.

The rest of this paper is organized as follows. In section 2 we describe the SUPERLAN. In section 3 we develop a simulation model and briefly explain an implementation of the optimistic and the conservative parallel simulation scheme. Section 4 shows the simulation results of the two approaches. In Section 5, we analyze the obtained results and finally some conclusions are presented in Section 6.

2.0 SUPERLAN

SUPERLAN is an integrated high speed local area network based on Wavelength Division Multiplexing Architecture. It provides isochronous and nonisochronous services, both narrowband and broadband, with arbitrary bit rates [6]. The network has, in its first phase, a physical ring configuration with a master station (MS) and a number of up to 64 ordinary stations (OS). Up to 16 substations are connected to each ordinary station (Figure 1). The substations are divided into three traffic classes, each requiring a different bandwidth for data transmission.

2.1 Network Structure

SUPERLAN consists of eight logically separate subnetworks:

- One for distributing the clock.
- Two for handling isochronous data and control information respectively.
- Two for nonisochronous data and its control.
- Three for network management, error handling, and flow control in the case of nonisochronous traffic.

2.2 The Isochronous Subnetwork

In this study we restrict our investigations to performance of isochronous traffic handling. We simulate the Media Access Control mechanism to obtain statistics on *setup delay* and *blocking probability*.

The isochronous data subnetwork has a capacity of 9.6 Gbit/s. The data is transmitted in constant-length (125 μ s) TDM frames, with variable-capacity, fixed-time slots (so called universal time slots). Each slot has a length of 10 ns duration, and can provide a capacity ranging from 64 kbit/s to 768 kbit/s. Hence, the number of bits in each time slot is determined in terms of the bandwidth needed by the particular call allocated to the slot. The addresses are implicitly contained in the slot positions in the frame and determined by a call setup procedure.

The isochronous control subnetwork has a capacity of 100 Mbit/s, and supports resource management flows and connection management (signaling) information flows. This subnetwork runs in a time-synchronous fashion with the data subnetwork. A train of TDM frames (also 125 μ s) circulates continuously in the subnetwork, originating and ending at the master station. The first slot in each frame serves as a synchronization header. After the header follow

a number of slots (chosen according to the media access mechanism) providing transportation for control units (called cells). The last slot in each frame (tail), is reserved for auxiliary functions. Every station has free access to a certain number of time slots in a frame, that is, they can read and/or overwrite control data in different time slots every time the frame passes through the node.

The master station handles network supervision and network operation control (resource allocation and management for isochronous traffic). The ordinary stations' task, on the other hand, is only to provide communication channels for the local traffic. Each of the substations connected to an ordinary station can provide only one kind of traffic.

2.3 Protocol Description

The network supports K types of traffics, traffic classes, each one having different requirements in terms of bandwidth. A fixed resource allocation policy is used in the master station for allocation of bandwidth resources (time slots) to different classes of traffics. Based on this protocol, each class has access to a certain number of time slots in each frame (bandwidth pool).

The multiaccess control for isochronous traffic is a mechanism acting at the call/session level. It emulates circuit switching for K traffic classes and it determines if a new requested call with a given bandwidth requirement in the isochronous data subnetwork should be accepted or denied service. This is a centralized control mechanism of type preventive which is implemented at the master station. It involves resource allocation through parameter negotiation between the caller station and the master station.

The call acceptance policy and the negotiated parameters are based on the traffic class (the required bandwidth), the bandwidth available for the given traffic class and the impact on the blocking probabilities. The protocol utilizes special control units, called *cells*, to be transported in the control channel.

3.0 The Simulation Model

Each of the ordinary stations is modelled as a logical process (LP) that also handles the generation of calls from the substations connected to it. The substations are thereby integrated into the ordinary stations in the model. The master station is modelled as two separate LPs (Figure 2b): the master station processor (MS_P) and the master station transmitter (MS_T). The structure of the simulation model is shown in Figure 2a. The logical processes communicate by message passing. Each message consists of either a single slot or tuples of slots.

The ordinary stations receive requests for call setup and requests for disengagement from the substations. These requests are placed in two queues according to type: a high priority queue for disengagement requests, and a low priority queue for setup requests (Figure 2c). Since the substations are incorporated into the ordinary stations, messages are actually only passed between the OSs. An OS receiving a tuple may extract some information from it (e.g. response to previous request) or insert new requests to the tuple before forwarding it to the next OS.

The master station processor receives slot tuples from the last OS on the ring. Slots containing some control information (a request for connection or disconnection) are processed and responses (to caller and called) are generated and passed to MS_T as a single cell. The transmitter queues the

response information, inserts it into the next available tuple and then sends it off.

3.1 The Time Warp Implementation

The Time Warp method, proposed by Jefferson and Sowizral [4], is the most known optimistic approach. Any process can proceed as long as it has pending events, but should a process receive a message with a time stamp smaller than its local simulated time (Local Virtual Time, LVT), it has to roll back in simulated time, undoing the events executed after the receive time of the message causing the rollback. Any messages sent after this time is cancelled by the sending of antimessages. This means that the states of the processes and all "newly" received messages must be saved periodically in order to make rollbacks possible. The Global Virtual Time (GVT) is calculated as the minimum of the LVTs for all processes and the send times of all messages sent but not processed. Processes only need to keep copies of the previous states and received messages back to and including the most recent state before GVT.

The cancellation policy described above is called aggressive cancellation and is the one used in this implementation. In addition to this, Fujimoto's direct cancellation mechanism [7] for shared memory architectures is also featured. It speeds up cancellation by eliminating the need for antimessages.

Two central ready queues for scheduling the logical processes (LPs) onto the physical processors are maintained: a high priority queue for processes that are to be rolled back, and a low priority queue for the others. The central scheduling that always selects the next pending event with the smallest timestamp should help prevent any process from running too far ahead of the others.

3.2 The Conservative Scheme

Several researchers have proposed window based conservative parallel simulation schemes (e.g., see [6, 7, 8]). The main idea behind all these schemes is to identify a window for each logical process such that events within different windows are independent and thus can be processed in parallel.

We have employed the following three-phase Conservative Time Window (CTW) scheme, described in [6], to identify independent windows.

The CTW scheme has the following properties:

- Events within a window are processed sequentially, but events within different windows are independent and can be processed in parallel.
- The size of each window is dynamically determined. Different windows, even those belonging to the same iteration, may have different sizes.

Initialization

REPEAT

1. Nomination phase

nominate a window W_i for each LP

$W_i = [t(e_i), \text{MAX_SIM_TIME}]$

BARRIER

2. Adjustment phase

identify length of the windows such that
they are independent

BARRIER

3. Evaluation phase

Process the events within these windows

BARRIER

UNTIL (end_of_simulation)

- Each of the phases of the algorithm may be executed by several processes in parallel. However, synchronization is required between any two consecutive phases.

4.0 Experiments and Results

The main objective of the simulation was to estimate the setup delay and blocking probability of the control subnetwork as defined below:

- *Setup delay* is the time taken from the instant a substation (user) generates a request for call setup to the instant when the substation receives the answer from the master station.
- *Blocking probability* is the ratio between connection requests denied and the total number requests

A satisfactory performance evaluation of SUPERLAN would require to observe its behavior for about one hour. Considering the characteristics of the network outlined in Section 2, a sequential simulator run on a workstation would need weeks of execution time. This problem was resolved in two ways:

- (a) We developed a sequential simulator and the network was simulated for one second of real time. The corresponding execution time for the sequential simulator was around two hours. The number of calls generated and the total number of simulated events for each run of the simulator corresponding to one second of real time (assuming 60 ordinary stations) are about 5000 and 10 millions respectively. How-

ever, one second is a short time and the obtained result was difficult to validate. Thus, this approach reduces the execution time of the simulator at the price of harder test conditions for SUPERLAN.

(b) We developed two parallel simulation testbeds on our Sequent Symmetry S81 shared memory multiprocessor: one is based on a Conservative Time Windows (CTW) scheme and the other one is a Time Warp implementation. We conducted several experiments on the parallel testbeds and compared the results with the sequential simulator. The results of the parallel simulators were identical to the results obtained from the sequential one. Our experimental results show that the execution time of the conservative simulator (Figures 3 and 4) was much shorter than the corresponding sequential one. To validate the parallel simulators, we also compared our results with an analytical model developed in [4].

The following assumptions were made in running the simulators:

- The session time of a connection (call) is generated as an exponentially distributed random time value.
- One simulation time unit is set equal to the duration of a slot in the transmission frame which is 680 ns.
- The distance between all ordinary stations is set equal. Thus, an integer number of slots is under transmission between two stations at any given time. The reason for this is that, to speed up the simulation, a tuple of slots rather than a single slot is passed as one message between processes in the model. This is to keep the number of messages down, as most of the simulation time would be spent on passing slots around the ring, rather than handling calls from substations since they are less frequent. The number of slots in one tuple is determined as the number of slots that can be accommodated (in terms of propagation delay) on the connection line between two stations.
- The time distance between stations was set to 1360 meters (corresponding to 10 simulation time units), and the Master Station processing time was 10 simulation time units.

We conducted several experiments with various number of ordinary stations, connection requests, and session times. More than 150 hours (CPU time) were spent on running these experiments. Some of the results are illustrated in Figures 3-5, where OS_K means that there are K ordinary stations in the network.

Figure 3a shows the speedup of the CTW scheme compared with one-processor execution of the same scheme. Figure 3b illustrates the speedup compared to a sequential simulator. The corresponding results for Time Warp are shown in

Figures 4a and 4b. As can be seen, the conservative scheme produces almost a linear speedup (about 6 running on 8 processors) for large networks. The obtained speedup for time warp using 8 processors is less than 2.

Figure 3a: Speedup of the CTW scheme compared to its one-processor execution

Figure 3b: Speedup of the CTW scheme compared to the sequential simulator

Figure 4a: Speedup of Time Warp compared to its one-processor execution

Figure 5: Number of committed events/second using Time Warp.

Figure 4b: Speedup of Time Warp compared to the sequential simulator

5.0 Discussion

Comparing the sequential simulator with the Time Warp executed on a single processor (Figure 4a) shows that the sequential simulator is almost two times faster. In this section, we will try to give some explanations.

- The increased overhead in the Time Warp simulation stems mainly from the state saving and message copying. The parameters of an event message is copied before the user defined function to process the event message is called. A pointer to the area holding the copied event is passed to this function. The copying of the parameters is needed because the user defined function operates directly on these data and hence may change these values. In the case that a rollback occurs there must be ways of undoing such changes, i.e. by retrieving a copy of the original values. In the sequential and conservative scheme, an event message consists of a pointer to a linked list, in Time Warp, however, the event message consists of a copy of the whole list corresponding to the tuple mentioned in subsection 2.4. Consequently, an event message is about 10bytes in the conservative and sequential cases, but about 2kbytes in the Time Warp simulation.

- Message passing is implemented using shared memory on our multiprocessor, where the required memory for copying messages is allocated dynamically. The memory allocation is performed by the operating system using an atomic operation. Our experiences indicate that memory allocation is a rather expensive operation.
- In this application, the state vector is rather large and the states are saved very frequently (after executing each event). Hence, the address space of each processor executing Time warp becomes large and may cause many page faults and low cache hit-ratio. To reduce the address space, the GVT calculation and fossil collection is performed rather frequently (once a second). Our investigation indicates that this is almost optimal and no additional gains can be obtained by doing these operations more frequently.
- Experiments on a 64 nodes fully connected network with constant message population, where the probability of forwarding a message to any other LP is uniformly distributed, indicates that increasing the state and message sizes from 32 bytes to 2kbytes results in an increase of the simulation time by a factor of 2 to 3.
- The application under investigation has a *good lookahead* and this is one of the main factors behind good performance of the conservative scheme.

6.0 Conclusions

we presented the results of simulating a multigigabit local area network using both a conservative and an optimistic approach. Our experimental results on a shared memory multiprocessor show that the conservative approach outperforms the optimistic one for this specific application. Our result contradicts most of the reported results in the literature, e.g see [5].

We believe that the special features of the application, e.g. good lookahead, and the used simulation model were mostly beneficial to the conservative approach. It is possible to use the same simulation model and the same data structures for both the sequential and the conservative scheme. However, moving from a sequential simulator to Time Warp requires remodeling the application. Otherwise, event copy

ing and state saving may kill all benefits that could be obtained using Time Warp.

Acknowledgement

This research is part of a project on Advanced Simulation Techniques and Tools financed by the Swedish National Board for Industrial and Technical Development (NUTEK) under contract number 93-3319.

References

1. Fujimoto, Richard M., "Parallel Discrete Event Simulation", in Communications of the ACM, Vol. 33, No. 10, October 1990
2. Adrian Popescu, Yuri Ismailov, Hassan Rajaei and Rasul Ayani, in MASCOT -93
3. Jefferson, David R., "Virtual Time", in ACM Trans. on Programming Languages and Systems, Vol. 7, No. 3, July 1986
4. Popescu, A. and Singh R.P. "An Alternative Solution to the Electro-Optic and Service Bottleneck Problems in Integrated Multi-Gbit/s LAN: the SUPERLAN Architecture.", in Computer Networks and ISDN Systems 25(May 1993): 1089-1105..
5. Fujimoto, Richard M., "Time Warp on a Shared Memory Multiprocessor", Intl. Conf. on Parallel Processing, August 1989.
6. Ayani, R. and Rajaei, H., "Parallel simulation based on conservative time windows: a performance study", In Cncurrency: Practice and Experience, vol. 6(2), 119 - 142 (April 1994).
7. Boris Lubachevsky, "Efficient distributed event-driven simulations of multiple-loop networks", Communications of the ACM, 32(1):111-123, Jan. 1989.
8. David. M. Nicol, "Performance bounds on parallel self-initiating discrete-event simulations. ACM Transactions on Modeling and Computer Simulation, 1(1):24-50, Jan. 1991