Analysis of the Performance of a Distributed Simulator

Jacques Labetoulle, Didier Loisel and Alaa Dakroub

EURECOM
2229 Route des Crêtes
BP 193
06904 Sophia Antipolis Cedex
France
{labetoul, loisel, dakroub}@eurecom.fr

Abstract

In a global project devoted to the realization of a distributed network simulator, we have investigated several ways to optimize the performance of our tool. To allow the comparison of different policies of management of the roll-back process implied by the distribution of the simulation, we have developed a specific simulation tool (a simulator of the simulator). We analyse here the structure of this simulator and how we can exploit it to optimize the real simulator.

The Network Simulator

The proposed study is part of a larger project in which we intend to build a simulator to analyze the behavior of a general corporate network [1]. Such a tool may be useful for many purposes :
- Besides the classical network management process, it is of high importance that a network operator may have the possibility to analyze the behavior of his network under various scenarios. As examples of these scenarios we can mention, predicted future load, evolution of the structure and dimensioning of the network and analysis of abnormal situations. This kind of analysis can only be made in a systematic way by using a dedicated simulation tool.
- In an academic environment, teachers would like to give their students a practical vision of corporate networks. Classical laboratory environments cannot provide such facilities. Only a simulated tool can allow the observation and demonstration of a global heterogeneous network.
- Network management in itself represents a new challenge for network conceptors. The linking of a network management technology with a simulated tool may be of a great interest for the conception, design help of new function development for network management tools. In fact, our project is larger than the realization of a simulated tool. It is planned to interconnect the simulation tool with a network management environment (with an agent and a manager level) with the aim to provide such facilities, but that is not the objective of this paper and will be described elsewhere.

This last aspect represents a difficult constraint for the development of our simulation tool since it implies that the simulator may run at a speed that are not compatible with the network management environment, thus near to the real time. Everybody can imagine that it will be quite difficult to satisfy such a constraint. So we have decided to build the simulator in a distributed environment to accelerate the process. Specialized models has been studied and written, each of which being dedicated to a given kind of network element (voice, X25, Ethernet, token ring, ATM, ...), to provide the best possible performance.

The simulator is composed of a kernel and a set of simulation modules. The kernel has been designed to synchronize the different modules while running the simulation. Before launching the simulation, instances of the specialized models are loaded in the modules, so that they will represent the topology of the network to be simulated. The affectation of models to modules is made in such a way that the flows between modules will be minimized. This process is very critical for the ability of the simulator to reach its objectives in term of simulation time. In a first step, this process will be done manually (by selection through a man-machine interface); an automatic tool minimizing the flows between modules will be studied later.

The Distributed Simulation

In parallel simulation, the processes of the model interact by exchanging messages. When designing a distributed simulation program there are two main approaches for the sequencing of these messages : a conservative and an optimistic one [2].
- In the conservative approach, represented by Chandy and Misra protocol [3] time progresses in each module in a controlled way, such that an event may never arrive out of sequence (i.e., the current time of the module is larger than the date of the event). This protocol suffers of dead locks that may occur during the simulation. To resolve this problem, null messages must be initiated in order to advance the clock of the blocked modules. However these null messages increase the communication overhead.
- In the optimistic approach , represented by Time Warp protocol [4] some risks are taken. Here, an event may arrive out of sequence in a module. To solve the problem it is necessary to abort the simulation and to restart it at a given time point, where we are sure to have a "correct" simulation. In order to do this roll-back we must have saved all the events, at least from this correct time point. The CPU time necessary to "re-simulate" a sequence of events affects the performance of the simulator. Some other overheads are introduced by the process of determing the correct time points, by the transmission delays they imply and by the fact some modules may stop running their simulation while the other have reached the synchrnization points. The benefits that we acquire by calculating these times are to discard all the data before this time in order to free the memory space allocated and decreasing the CPU overhead since we will have to "re-simulate" shortest sequence. The challenge is to design these synchronization points and the policy to apply in an optimized manner.

The analysis of the literature in the area of distributed simulation has shown that many solutions has been proposed to optimize the global performance of a distributed simulator. It's accepted that Time Warp outperforms Chandy-Misra in must cases [2]. Many studies have been done to enhance Time Warp performance. Most of the time, only some improvements of previous methods are given, and the comparison of performance is limited to the two implied methods. For the purpose of optimizing our own simulator, we have designed a simulation program, in which the policy for time management (and more explicitly the way to solve the problem arising when a desynchronized event occurs) can be parametrized. This tool can be exploited to analyze different scenario, allowing the optimization of the network simulator.

Description of the Simulation Tool

The program is composed, as the simulator itself, of a kernel and a set of modules. The kernel is in charge of implementing different possible policies as mentionned in the previous section. A scheme of the simulation model of a module is shown in figure 1. The program is written using the Modline [5] package, in its last version allowing object oriented programming. Events in the simulation are represented by objects, containing all necessary information (date of creation, execution times, routing, ...) to facilitate the restart of the simulation at any predefined synchronization point.

The main parameters used in the simulation tool are the following:
- simulation time of an event (that may be different for each module, depending on the type of network being simulated),
- routing probabilities of events,
- network delays (time to send an event to a module),
- delays introduced at the kernel level (time to treat the policy procedure in case of roll-back, time spend at the end of a simulation period, ...)
- simulation period (modules are requested to simulate until a due date, so that they will be synchronized at that date),
- the number of modules

The selector chooses the soonest event to be served amongst 4 sets :
- the next generated new event,
- the set of "event replay", filled after having restarted the simulation,
- the set of events being arrived from other modules (external arrivals)
- the target date (used to synchronize all modules).

This selection takes place as far as the target date (time of synchronization) is not reached. If it is the case, the module stops its simulation, informs the kernel and waits until a new target time is sent by the kernel. The selector then send the selected event to the "server" where it will be processed. The time sprent in the server is determined once for each event, at the generation of the event, and is memorized in the event itself as an attribute.

At the end of the process in the server, the event is systematically copied in the "event memory queue", for eventually being re-processed if necessary. An option in the programm allows to define intermediate time points, so that the "event memory queue" will be composed of a set of queues, each one dealing with a given time sub-interval. When using this possibility, the total number of events to replay in case of a roll-back procedure can be smaller (limited to the time periods where the problem took place).

Depending upon its own routing information the event, after update of its current date, can be sent to one another module. When arriving in this new module, its date is compared to the current date of the new module. If the event arrives too late from the point of view of the new module, a special procedure is launched (contained in the Kernel process), in which is defined the policy of the roll-back process. This is the only procedure to be rewritten for testing a new policy. In that case, a message is sent to all the modules asking them to stop the simulation.

The roll-back procedure will take events from the "event memory queue" and fill the "event replay" queue before restarting the simulation at the right time. The event that caused the problem is ememorized in the "External Event" queue of the new module (in advance in that case) so that it will be treated at the right moment the second time.

Each module is authorized to process its simulation until a predefined target date. When all the modules have reached this date (eventually after one or several roll-back processes) the kernel calculates the statistics, empties all the "event replay" and "event memory" queues, gives the new target date and restarts all the modules.

Due to the fact that we are simulating a simulator, we are obliged to deal with two time scales : that of the real simulator (the time it takes to treat the events) and that of the events themselves. That means that each event contains two time attributes, one for each scale. When the event is sent to a module, the time comparison refers to date of the event being processed in the new module compared to its own date.

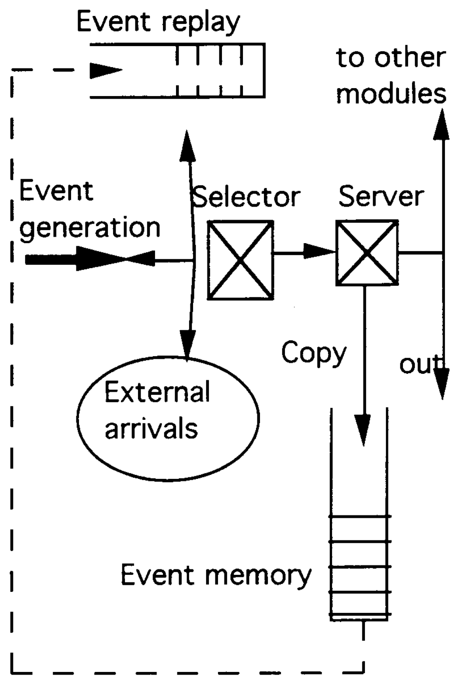

Figure 1

Result Exploitation

We have intensively exploited the simulation tool to get a better idea on the ideal way to implement our network simulator. Numerical results will be given during the conference, since they require a large place. It appears that the optimization process is quite sensible to the real parameters of the network simulator : time to treat an event, length of the period (determination of the target dates), routing probabilities, ... It has been verified that even with a very basic procedure (just restarting all modules at the previous synchronization point when a de-synchonized event is detected), the advantage of distributing the simulation is obvious. Depending on the optimization of the time period (the sensitive parameter for a fixed routing probability) we can obtain an acceleration factor greater than 50 % for 3 modules in parallel. It is also shown that the performance is improved only if the routing probability of events is small (about 10 to 15% of events transit in more than one module). The real benefit obtained by distributing the simulation can be determined by using realistic numerical values

It seems clear that the distribution of the simulation accelerates by an interesting factor the performance of the simulation. The results show that the roll-back policy is very sensitive and can improve significantly the global performance of the simulator. A better policy implies a better performance, but it is difficult to evaluate (at the point we are now) the overhead introduced by the realization of this policy that may affect this result. So we must be careful by identifying the parameters of the network simulator. The direction we have chosen is the following :
- we will carefully measure the performance of the simulation modules (all the parameters introduced in the simulation tool) and of the kernel,
- we will identify the characteristics of the real simulation (the number of events in each module, routing probabilities, ...),
- we will then use the simulation tool, with the measured parameters to deduce the optimization of the network simulator,
- we will then experiment the network simulator with this optimization to verify the validity of the approach.

This process cannot be experimented until the network simulator is operational (not before some months from now).

Conclusion

With the realization of a simulator of our target network simulator, we have shown that the performance of a distributed simulation can be significantly improved, by a careful selection of parameters of the roll-back policy. Some numerical results will be given during the presentation; further results are required to really optimize the network simulator, after having measuring it and exploiting the results.

References

[1] A. Dakroub, J. Labetoulle, D. Loisel: An Architecture for a Network Management Simulator, 4th Open workshop on High Speed networks, Brest, September 1994.

[2]Yi-Bing-Li, Understanding the Limits of Optimistic and conservative Parallel Simulation, Technical Report 90-08-02 University of Washington 1990.

[3] Chandy, K.M. and Misra, J. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. IEEE Transactions on Software Engineering, SE-5(5):440-452, September 1979.

[4] Jefferson, D., Beckman, B. Wieland, F., Blume, L., Di Loreto, M., Hontalas, P., Laroche, P., Strudevant, K., Tupman, J., Warren, V., Vedel, J., Younger, H., and Bellenot, S. Distributed Simulation and the Time Warp Operating System. Proc. 11th ACM Synposium On Operating Systems Principles, pages 77-93, November 1987.

[5] MODLINE V 1.2, User's Guide, Simulog S.A. 1994.