Relaxation of Synchronization Constraints in Parallel Simulation of ATM Network

C.D. Pham S. Fdida

Laboratoire MASI
Université Pierre et Marie Curie
4 place Jussieu 75252 Paris Cedex 05
e-mail : {pham,fdida}@masi.ibp.fr

Abstract

Parallel simulation using conservative or optimistic approach does not exhibit good performance when applied to large ATM networks. We introduce a statistically correct approach based on the observation that some performance measurements can be obtained without the total time ordering established by conventional approaches. In our approach events can be processed in partial time order. Upon reception of an outdated message, a process does not attempt to cancel the bad computations it has done but uses local estimations instead. To demonstrate our approach a parallel simulation of an ATM network is performed applying this idea. We focus on the mean buffer length of each switch outputs and found out that the results are very close to those of a correct simulation. The statistically correct approach reduces the simulation cost while keeping a reasonable level of accuracy.

1 Introduction

ATM technology has been adopted as the transfer mode for future B-ISDN resulting in an intense activity to carefully evaluate the network components. The size of the network may exceed hundreds of nodes, making analytical methods unsuitable. Simulation does not suffer from the limitations of analytical methods which typically require simplifications of the real system and reduce the usefulness of the results. On the other hand, discrete-event simulations of large systems if processed sequentially are very time-consuming. Techniques for parallel simulation have been developed to reduce the execution time of the simulation.

In parallel simulation each physical process of a given system is simulated by a Logical Process (LP). Interactions between the physical processes are represented by timestamped messages exchanged between LPs. Parallel simulation must address the problem of process synchronization because the LPs may advance at different rate. A time error occurs when a message arrives at a receiving LP and is outdated, or old, according to the local virtual time. Existing mechanisms for parallel simulation must maintain correct synchronization between LPs in order to obtain the same results as if the simulation was processed sequentially. These synchronization mechanisms can broadly be classified in two categories: conservative and optimistic.

Conservative algorithms avoid all cases of time errors [Chandy81]. An LP is allowed to process an event only if it is determined with certainty that no time errors could occur anymore. If this is not the case the LP blocks. Null-messages are used to artificially advance the simulation time in order to avoid deadlock. In optimistic algorithms an LP can process all available input messages but implements rollback mechanisms to recover from time errors when they occur [Jefferson85].

Both conservative and optimistic approaches present many drawbacks. Conservative approaches need static configuration so the simulation of large systems is difficult. Furthermore they introduce a high number of null-messages in order to avoid deadlock when the number of real message in the system is small. This high null-message/real message ratio causes performance of the protocol to drop dramatically. Optimistic approaches, though not limited by the size, present high risk of cascaded rollbacks. The cause of these problems is that traditional approaches want to obtain correct simulations and so require that the messages be processed in increasing timestamp order. Based on the observation that some performance measurements can be obtained without the total time ordering established by conventional approaches, we introduce a *statistically correct approach* where some events can be processed in partial time order without rollbacks. For example, if one focuses on switch buffer occupancy then processing a reception event $<$RECV,$t_2 >$ before another reception event $<$RECV,$t_1 >$, with $t_1 < t_2$, will not dramatically affect the measurement of interest. On the other hand, execution time is reduced because some rollbacks have been avoided. If loss rate or blocked call probability are investigated the same remarks still hold.

Upon reception of an outdated message, a process does not attempt to cancel the bad computations it has performed since the arrival of this message but uses local estimations instead. These estimations are based on saved historical information in order to produce correct output messages. The results obtained by running the simulation are *statistically correct* when compared to those obtained by a conventional approach. The level of accuracy depends on the errors introduced by the bad computations that were not cancelled. To demonstrate our approach a parallel simulation of an ATM network is performed applying this idea. We focus on the mean buffer length of each switch outputs and found out that the results are very close to those of a correct simulation.

This paper is organized as follows : the next section details the new approach that introduces the idea of a statistically correct simulation; section 3 presents the test cases, the computation of local estimations and the preliminary results. Section 4 concludes the paper.

2 The Statistically Correct Approach

For purpose of defining the statistically correct approach, we give some basic definitions. The term simulation refers to distributed discrete event simulation and we assume that the reader is familiar with Logical Process (LP) and process local virtual time (LVT).

2.1 Definitions

- *Definition 1* : A very-correct simulation is a simulation where events are processed by an LP in increased timestamp order with no time errors.
- *Definition 2* : A correct simulation is a simulation where some time errors have occurred but have all been corrected by a rollback mechanism.
- *Definition 3* : A statistically correct simulation denotes a simulation where some time errors have occurred but not all of them have been corrected.
- *Definition 4* : Let φ be a finite sequence of n events $< e_i, t_i >$ where t_i is the timestamp of event e_i. A sequence is said to be rollback-free if $\forall i, t_{i+1} \geq t_i$. In the same manner, a k-rollback sequence is a sequence where there are k t_i such as $t_i < t_{i-1}$. In extension, a rollback sequence is a k-rollback sequence where $k \geq 1$.

A conservative approach like the Chandy-Misra-Bryant algorithm typically produces very-correct simulations, whereas an optimistic one like Time Warp is more likely to produce correct simulations. Our statistically correct approach is on the optimistic side and can be classified as "very optimistic". Consequently we will compare its behavior to the Time Warp algorithm.

Now let TW_φ and ST_φ be the simulation of an LP using respectively the Time Warp algorithm and our statistically correct approach under the sequence φ. Simulation of an LP under a sequence φ means that the LP successively processes event i of φ, $i = 1..n$. We then define the operation $=$ between two simulations a and b as true if their results are identical. Finally the following statements can be written:

1. if φ is a rollback-free sequence then TW_φ and ST_φ are both very-correct simulations. We will have $TW_\varphi = ST_\varphi$.
2. if φ is a rollback sequence then TW_φ is a correct simulation. On the other hand, ST_φ is a statistically correct simulation. In most case, we will have $TW_\varphi \neq ST_\varphi$.

Since a rollback-free sequence does not generate time errors, simulations using any synchronization protocol, existing or to come, under such a sequence are all very-correct simulations. Statement 1 expresses the fact that under a rollback-free sequence our statistically correct approach provides results that are identical to those provided by a conventional Time Warp approach. Now if φ is a rollback sequence the Time Warp algorithm produces correct simulations by means of its rollback mechanism. On the other hand, we wanted our statistically correct approach to produce statistically correct simulations. This means that upon reception of a bad message the heavy rollback mechanism is not used to cancel the bad computations. In order to obtain coherent results, alternatives to rollback must be found. We propose utilization of local estimations and a different local virtual time evolution scheme as described in the next sections.

2.2 Local estimations

The life of a process consists of receiving and processing messages. Processing a message often changes the process state and may generate output messages to other processes. When a bad message m_{past} with timestamp t_{past} arrives at time t_{now}, a traditional optimistic approach must roll back. This means that the LP must restore the process state before t_{past}, jump back to time t_{past} and send anti-messages to cancel messages that it had produced. Then all events since t_{past} are re-executed. This is required because the processing of m_{past} may change the evolution of the process making all states and all produced messages since t_{past} erroneous.

There is no way but to roll back if one wants to cancel messages that have already been sent. On the other hand, if one accepts the errors introduced by not cancelling produced messages then there is no need to roll back. Local estimations can be used instead to avoid the overhead of rolling back. This is the driving idea expressed by our statistically correct approach. Estimations should be designed to produce correct output messages resulting from the processing of m_{past} and to re-evaluate the process states from t_{past} to t_{now}.

We propose local estimations based on historical information. Such historical information can be included within the process state and periodically be saved. When a past message m_{past} arrives, the process virtual time does not change and saved historical information before t_{past} are used to correctly process m_{past} and to generate output messages. Re-evaluation of saved states is also performed to take into account the changes introduced by m_{past}.

The way local estimations are computed is quite application dependent. We give in section 3 the estimation used for the simulation of an ATM network.

2.3 Local virtual time evolution

In the Time Warp algorithm, a time error causes the process virtual time to jump back to the time of the error in order to re-execute events in the right sequence. Now since local estimations are used instead, when a time error occurs the process virtual time needs not to jump back. The processing of a past message is done in what we call an *estimation environment*. Figure 1 shows the possible evolution of virtual time for (a) a conventional approach and (b) the statistically correct approach.
The first time error blocks the process virtual time at time t_{now} and initiates an esti-

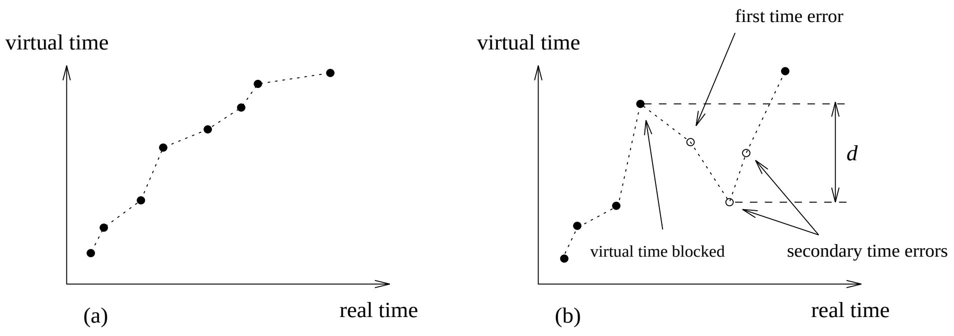

Figure 1: Evolution of local virtual time for (a) a traditional approach and (b) the statistically correct approach.

mation environment until a message with timestamp t greater than t_{now} is received. Since the virtual time is blocked, subsequent messages with timestamp lesser than t_{now} can also produce time errors. We call this last kind of error *secondary time error* as opposed to *first time error*. In figure 1 a white point indicates the processing of a past message, relatively to t_{now}, in an estimation environment.

3 Test cases and preliminary results

3.1 Test cases

Test 1 simulates an ATM switch with three traffic sources connected into it (figure 2). The switch has output buffers and the following switching strategy : in one slot time each cell that arrives on an input port is switched to the corresponding output. The choice of the output follows some fixed routing probabilities. Only one cell can arrive per slot time on an input but severals cells can be switched to one output, in which case they are buffered in the output queue. On the other hand, for all output the head cell in the queue is transmitted on the link in each slot time.

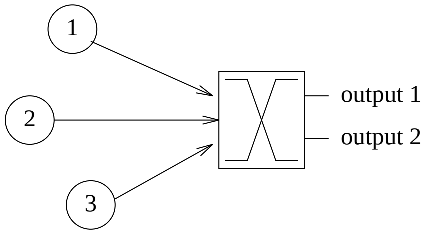

Figure 2: Network configuration for test case 1

In our model the switch has three inputs and two outputs. Traffic sources continually generate cell reception message. In order to generate at most one cell per slot time on a link, the cell inter-arrival time δ is computed as follows: $\delta = ceil(\tau)$ where *ceil()* is the rounding toward positive infinity function and τ an exponentially distributed variable. Upon reception of an <RECV,t > message, the switch schedules for itself a <SEND,t_d > event where t_d is the date of departure of the cell. A cell from a given input $i = 1..3$ is switched to an output $j = 1, 2$ with probability p_{ij} and sent out by scheduling at the receiver a <RECV,t' > message. We set for the simulation $p_{ij} = 0.5$ for all i and j.

In test 2 a network of five switches is simulated (figure 3). Only one traffic source per switch is considered that generates cell reception messages but only up to time $t_{endsource}$ lesser than the simulation time. Since there is no sink process, this limits the number of cells in the network. When a switch a receives a cell from another switch, it sends it to the output $j = 1, 2$ with probabilities p_{aj}. When traffic sources do not generate cells anymore, the number of cells in the network remains constant and switches keep sending existing cells to each others until the end of the simulation. We set for the simulation $p_{aj} = 0.5$ for all switches a and output j, and $t_{endsource} = 40$. Propagation delay between switches takes 2 slots time.
The state of a switch process is represented by a vector $S = (o_1, o_2, last_1, last_2, n_r, n_s)$. o_j and $last_j$ are respectively the number of cells waiting in the output queue j and

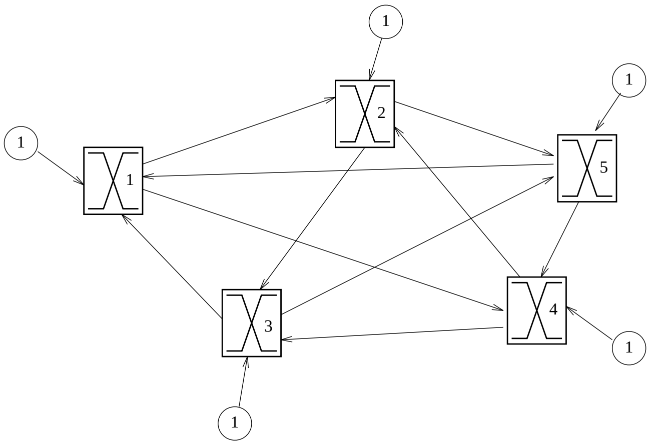

Figure 3: Network configuration for test case 2

the *last time* that a cell has been switched to output j. n_r and n_s are the number of received and sent cells respectively. State saving for local estimations is done after every processing of an <RECV,t> event. The processing of self-generated events such as <SEND,t> does not make the process to save its state. This choice keep the number of historical information to a minimum.

3.2 Computation of local estimations

Let L_s be the list of saved states and $S_i = (o_1^i, o_2^i, last_1^i, last_2^i, n_r^i, n_s^i)$ be the ith saved state in L_s. A saved state S_i has a timestamp T_i. Index i increases as T_i increases. Upon reception of an outdated cell reception event <RECV,t>, $t < LVT$, we use the following estimation algorithm at the switch process:

1. Gets the first state before time t from L_s. The retrieved state is S_i with $T_i \leq t < T_{i+1}$.

2. Determines the output j, derives the date of departure t_d of this new cell by computing its waiting time and schedules an $< SEND, t_d >$ event for itself.

3. Inserts the new state in the list just before the retrieved state S_i (so at index $i+1$; all state indexes are incremented).

4. Re-evaluates all states S_k with $k > i+1$.

Since historical information are saved after every <RECV,t> event, the retrieved state is exactly the state the first past message would have find if it was received in sequence. For subsequent bad messages, the retrieved state can be a re-evaluated one. We explain step 2 to 4 in what follows.

Let t_w be the waiting time of the cell before the beginning of its processing at the output j. The date of departure t_d of the cell on output j is computed as follows (time is expressed in slot time) :

$$\blacksquare \text{ Step 2} \begin{cases} \text{if } t = last^i_j \text{ then } t_w = max\{o^i_j - 1, 1\} & (eq.\,1) \\ \text{else } t_w = max\{o^i_j - 1 - (t - T_i), 0\} & (eq.\,2) \\ t_d = t + t_w + 1 + 1 & (eq.\,3) \end{cases}$$

Note that if $t = last^i_j$ we have $t = T_i$ and $o^i_j \geq 1$. On the other hand if $t \neq last^i_j$ then we can have $t = T_i$ but this is not mandatory. Since we assume that several cells can be switched from different inputs to an output queue in one slot time, relation $t = T_i$ expresses the fact that there is more than one cell switched to an output in one slot time. More precisely relation $t = last^i_j$ indicates that there has been at least one cell previously switched to output j in the same slot time. So if $t = last^i_j$ and $o^i_j > 1$ (eq. 1) the cell must wait for all the cells in front of it in output j at time t, minus the one that was already sent when the cell will arrive at the output queue. This because transmission of the head cell of an output also requires one slot time. Now if $o^i_j = 1$ (eq. 1) then the cell must only wait for the unique cell in front of it in output j because both have been switched in the same slot time. If $t \neq last^i_j$ (eq. 2) then the cell must wait for all the cells in front of it in output j at time T_i, minus those that were already sent during the interval $t - T_i$, and minus the one that was already sent when the cell arrives at the output queue. Of course if the output queue is empty then t_w is nul. The date of departure t_d is obtained by adding to t the switching time (one slot time), the transmission time (one slot time) and the waiting time (eq. 3).

The new process state S_{i+1} after the processing of the <RECV,t > message is saved at index $i + 1$ and we have :

$$\blacksquare \text{ Step 3} \begin{cases} o^{i+1}_j = max\{o^i_j - (t - T_i), 0\} + 1 \\ last^{i+1}_j = t \\ n^{i+1}_r = n^i_r + 1 \\ n^{i+1}_s = n^i_s + min\{o^i_j, t - T_i\} \end{cases}$$

If $o^i_j < t - T_i$ then all cells waiting in output queue j have been sent during the interval $t - T_i$. In which case $o^{i+1}_j = 1$, e.g. the new cell waits alone in the output queue j. On the other hand if $o^i_j > t - T_i$ there are some cells left and the new one waits with the other $o^i_j - (t - T_i)$ remaining cells. In the same way, the number of sent cells is incremented by the number of cells that have been sent between t and T_i.

Re-evaluation of all states S_k with $k > i + 1$ is done as follows:

$$\blacksquare \text{ Step 4} \begin{cases} o^k_j = o^k_j + 1 \text{ if } T_k - t < o^{i+1}_j \text{ else } o^k_j \text{ is unchanged} \\ last^k_j \text{ is unchanged} \\ n^k_r = n^k_r + 1 \\ n^k_s = n^k_s + 1 \text{ if } t_d < T_k \text{ else } n^k_s \text{ is unchanged} \end{cases}$$

If $T_k - t < o^{i+1}_j$, it means that the new cell has not yet been sent at time T_k so o^k_j should be incremented. On the other hand if $T_k - t \geq o^{i+1}_j$ then the new cell has been sent and o^k_j remains unchanged. In the same manner, if $t_d \leq T_k$ then the new cell has been sent before or at time T_k so n^k_s should be incremented. Note that if $t_d \leq T_k$ then we have $T_k - t \geq o^{i+1}_j$ because both conditions are equivalent.

3.3 Preliminary results and discussion

Simulations are carried out on a network of SparcStations. Implementation is done in C using the PVM package [Pvm93] for inter-processes messages passing. The results show the number of first and secondary time errors that are detected, and the buffer occupancy for the switch outputs. The simulation time is fixed to 4000 slots time. In order to easily compare the accuracy of the results provided by our statistically correct approach, deterministic sources are used. This means that from one simulation to another, the sources generate the same sequence of cell reception messages $<\mathrm{RECV}, t>$. However, the randomness of inter-network message passing delay causes the number of time errors from one simulation to another to vary.

Tables 1 and 2 show the buffer occupancy for test 1 and 2 obtained with a correct simulation. Table 3 and table 4 show the results obtained using the statistically correct approach.

Test case 1	SW1
Buffer 1	0.812
Buffer 2	0.846

Table 1: Results with a correct simulation of test 1

Test case 2	SW1	SW2	SW3	SW4	SW5
Buffer 1	2.584	2.588	2.581	2.572	2.578
Buffer 2	2.578	2.574	2.561	2.558	2.558

Table 2: Results with a correct simulation of test 2

Test case 1	SW1
First time error	5
Secondary time error	22
Buffer 1	0.809
Buffer 2	0.841

Table 3: Results with a statistically correct simulation of test 1

Test case 2	SW1	SW2	SW3	SW4	SW5
First time error	93	109	113	98	112
Secondary time error	135	157	151	137	148
Buffer 1	2.425	2.374	2.421	2.385	2.463
Buffer 2	2.383	2.436	2.401	2.434	2.395

▪ **Table** 4: Statistically correct simulation of test 2

Only one simulation result is shown for test 1 but in all simulations that we have performed for this case we found that the results are very close to the target values obtained by a correct simulation. We can notice that the number of first time errors is small. This is easily explained by the fact that the system is very simple and transfer of cells is only unidirectional from sources to the switch. This lowers the degree of interactions between the network components and then reduces the number of time errors. Good accuracy is then obtained.

For test 2 things are very different since interactions between network components are much more complex. In this scenario a switch a can receive from another switch b a cell that it has originally sent. If meanwhile switch a has processed several cell reception events from other switches then the risk of a time error is high. When compared to test 1, we do find that the number of first time errors is much greater. However, and it is encouraging, the results obtained have not been dramatically altered.

For both test cases the value of the buffer occupancy provided by our statistically correct algorithm are always lesser than the value obtained by a correct simulation. This phenomenon was predictable since the error introduced by not rolling back consists of neglecting the possible amount of extra waiting time of previously sent cells. In the light of the obtained results this error is kept relatively low.

4 Conclusion and perspectives

A new approach called *statistically correct approach* is proposed for parallel simulation where total correctness is not always required. This approach reduces the simulation cost while keeping a reasonable level of accuracy. Early tests with use of local estimations give favorable results.

The way estimations are obtained is very application dependent. We are aware that this scheme gives better results when generated messages and process state do not change very much after the processing of the missing events. This is typically the case for ATM networks. Future works would address the use of this approach to larger networks with hundreds of nodes. We believe that in order to make such large scale simulations possible in a reasonable amount of time, one should relax the synchronization constraints and accept statistically correct results.

We are in the process of adding a control that decides whether to use local estimations or the ultimate solution of rolling back. With such a control, the statistically correct approach offers the possibility to span the continuum of correctness from correct to statistically correct. In addition, further studies should address the performance and the suitability of a the statistically correct approach under more sporadic traffic sources.

References

[Chandy81] K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations", *Communication of the ACM, Vol. 24(11), April 1981, pp198-206.*

[Jefferson85] D. R. Jefferson, "Virtual Time", *ACM Transaction on Programming Languages and Systems, Vol. 7(3), July 1985, pp405-425.*

[Pvm93] A. Geist and al., "PVM 3 User's Guide and Reference Manual", *May 1993.*