Length-based Blocking and Local Estimations in Distributed Simulation: A Case Study

C.D. Pham S. Fdida

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

Abstract

This work investigates the use of a preventive length-based blocking strategy associated with local estimations. A logical process must block until the length of its input queue becomes greater than a threshold L_{min}. By blocking one can reduce the number of time errors in order to allow the local estimations to handle more efficiently the remaining time errors. In a first time, we seek for an empirical optimal trade-off between the initial blocking length, the speedup relative to a conservative simulation and the accuracy of the results. Then an automatic tuning is performed to adjust the initial blocking length to a desired time error ratio. We apply this method to simulate an ATM network and the results show that the length blocking strategy succeed in reducing the number of causality violations with a very small blocking overhead. Also both accuracy and speedup can be achieved.

1. Introduction

In most distributed simulations a real system is modeled by a collection of Logical Processes (LP). The interactions that exist in the real system are represented by timestamped messages exchanged between LPs. Since LPs are spread over several processors, synchronization mechanisms are required to ensure causality. These mechanisms traditionally fall in two categories, *conservative* and *optimistic*. On one side, the conservative approach [2] (CMB) requires that execution of an LP is halted until it is certain that processing an event is safe and would not result in further causality violations. Deadlocks that are unfortunately the counterpart of this blocking scheme are avoided by sending null-messages in the purpose of advancing artificially the simulation time. On the other side, optimistic approaches [9] do not avoid causality errors but provisions are made to roll back when they occur. Periodic checkpointing and anti-messages are then needed to permit the rollback mechanism.

During the last decade, a lot of work has been conducted for improving the initial conservative and optimistic scheme. A survey can be found in [5]. However, these new protocols do not call the causality constraints into question and therefore they still require some mechanisms to ensure causality. On the other hand, for some applications such as queuing networks, it is not mandatory to keep a tight synchronization between LPs to catch mean value statistics as demonstrated in [22]. When going a step farther, relaxing synchronization constraints can lead to an inaccurate behavior. In the terminology proposed by Reynolds [17], accuracy means that events must be ultimately processed in time order. Inaccurate protocols therefore allow events to be processed out of time order and do not try to roll them back. For instance, Time Warp is said to be accurate, and not inaccurate, because the rollback mechanism offers to the user the external view of events that have been processed at the simulation time at which they should be processed. Historically, inaccuracy was first used in SRAD [16] and then in [18] with the *Moving Time Window* protocol where *Side-Effecting* (SE) and *Non-Side-Effecting* (NSE) event are distinguished. A NSE event is supposed to always leave the process state unchanged. Alternatives to rollback using historical information are then available if a causality violation is produced by such an event. A causality violation caused by an SE event makes the process to roll back in a traditional fashion. In [13], a more general use of inaccuracy has been proposed. In this approach, local estimations, which are application dependent, are derived from saved historical information to serve as a substitute for the rollback mechanism. However it was found that accuracy may drop if the number of causality violations is too large.

Obviously, accuracy could be improved by reducing the number of causality violation. In a purely optimistic approach, one usually sets a brake on the "unlimited" optimism of the protocol. Many solutions have been proposed and two categories can be outlined: *window-based* and *blocking-based*. In this paper, we investigate the use of a queue length-based blocking strategy associated with local estimations. An LP must block until the length of its input queue becomes greater than a threshold L_{min}. The blocking strategy reduces the number of causality errors thus allowing the local estimations to handle more efficiently the remaining causality violations. We apply this method to simulate an ATM network and we seek for an empirical optimal trade-off between the initial blocking length, the speedup relative to a conservative simulation and the accuracy of the results. Then by setting a desired time error ratio, it is possible to automatically adjust the initial threshold to an optimal value when the environment changes.

Our work is organized as follows: the next section presents some related works on limiting optimism. Motivations for blocking strategies are presented in section 3. Our blocking strategy is detailed in section 4. Section 5 and 6 describe respectively the ATM network model and the local estimations. In Section 7 the preliminary results and a tuning algorithm are presented. The conclusion is given in section 8.

2. Background

Window-based methods to limitation of optimism have been used in previous protocols such as *Filtered Time Warp* (FTW) [11], *Moving Time Window* (MTW) [18], *Breathing Time Bucket* (BTB) [21] and *Adaptive Time-Ceiling* (ATC) [12]. These protocols limit the optimism by confining the events to be considered to a time window. The size of this window can be static as in FTW and MTW, even if MTW dynamically moves the time window over simulated time, or dynamically adjusted. For instance, BTB divides the simulation in *buckets* of variable size. A bucket is a kind of implicit dynamic time window which upper bound, called the *event-horizon*, is defined to be the smallest timestamp among those of generated events in that bucket. In the ATC protocol a *time-ceiling*, which is the upper bound of a time window, is dynamically changed by monitoring the simulator performance and the speculation error frequency.

On the other hand, most blocking-based protocols add a conservative behavior by blocking the LPs by a certain amount of time. There have been a tremendous growth of such protocols that rely on blocking to adjust them-selves between the two end-points protocols represented by the traditional conservative and the fully optimistic approaches. Srinivasan and Reynolds [20] define a class of protocols called *Asynchronous Adaptive Waiting Protocols* (AAWP) that contains all the protocols described hereafter. A *penalty-based algorithm* has been proposed in [15] where an LP is not scheduled for a fixed amount of time for each anti-message it sends. Since sending anti-messages appears definitely as a bad behavior meaning that incorrect computations have been done, delaying execution is a way to avoid additional errors. Ball and Hyot propose the *Adaptive Time Warp* protocol [1] that suspends the execution of an LP if an abnormally high number of *time faults* has occurred in the recent past. An estimate of the optimal blocking window for each LP is computed by minimizing a wasted computation function. In [3], a cost model for moving across a continuum between optimistic processing and conservative blocking is constructed by recording the timestamp of incoming messages. The authors compute from this cost model an optimal CPU delay for every event before its execution. Hamnes and Tripathi propose a *Local Adaptive Protocol* that spans the continuum of conservative to optimistic simulation by associating a real-time blocking window (RTWB) with each channel [7]. When some channels are empty, the RTWB controls the event selection procedure, i.e. wait or block or go on. The optimal RTWB is periodically computed and should maximize the simulation time advancement rate. In [19], the authors introduce a new subclass of AAWP protocols, called *Near Perfect State Information* (NPSI) that typically rely on the acquisition of global information at very low cost to adjust their amount of blocking time to an optimal value. They also propose an NPSI protocol, namely *Elastic Time Algorithm* (ETA), that considers the gap between the LVT and the GVT, obtained with an NPSI method, to reduce its optimism.

3. Motivations for blocking

Even if blocking stops the progress of the simulation, the purpose of this strategy is simple. First, we must note that in many cases the probability of encountering a causality violation decreases when the number of events in the input queue increases. This is easily explained by the fact that the LP has more "choice" when picking an event for execution. Second, processing an input message as soon as it is available is very interesting if the message is the one expected (the message is on the critical path of the simulation) but can turn out to be very hazardous otherwise because this can lead to an excessive number of rollbacks. Therefore, blocking is a way to decrease the rollback probability by allowing more events to be placed in the input queue before any execution. Of course, this is only advantageous if the time wasted in the blocking state is smaller than the time spent in rollbacks. In [20], it was shown that bad decisions on the blocking time can conversely create more "false" rollbacks. Therefore a blocking scheme should be manipulated very carefully.

Unlike the penalty-based algorithm which blocking strategy can be classified as *reactive*, our length-based blocking, and all the previously cited protocols, should mainly be classified as *preventive*[1]. In a reactive strategy an LP has to block only on occurrence of a causality violation. This scheme is efficient if such causality errors are relatively infrequent or come in clusters. Unfortunately this assumption is untrue for models of system where sub-systems intensively interact each other. In this case a lot of messages are exchanged from an LP to another making causality violation quite frequent and occurring not in clusters but rather uniformly during the simulation. Reactive schemes are very difficult to tune because if the reaction condition or speed are badly parameterized this will lead to an unstable behavior that gives very poor performance. Conversely, a preventive strategy seems the most likely to succeed in reducing the number of causality violations even at the additional cost of always monitoring the simulation progress.

No much works have been conducted on length-based blocking though it seems more natural to put the blocking condition on the length of the input queue than on a waiting time because one can not know with certainty when messages will come. A time-based blocking has the advantage to allow a continuous behavior as the blocking time can be tuned very sharply. In addition, the load imbalance problem that tend to trigger out LPs from their blocking state prematurely if a length condition is used is lessened. On the other hand, with a length-based blocking strategy, it is possible to reduce the blocking overhead since the length condition can be satisfied immediately. Therefore, there are cases where a time-based blocking strategy has to block while a length-based one does not without additional risk. For example, consider the case where the optimal blocking time before every processing of event is found to be T_{opt} and the corresponding optimal length threshold to be L_{opt}. With a blocking time, one must delay execution of T_{opt} unit of time in order to be sure that the causality violation probability will be effectively reduced, even if it turns out that it was unnecessary. Now, if it was unnecessary to block because there are enough events in the pending queue to reduce the causality violation probability then an LP with a length-based blocking strategy can immediately jump over the blocking state. Therefore, we think that length-based blocking is of interest.

[1]The term *pro-active* can also be used when a kind of feedback control is implemented.

4. Length-based blocking

The preventive blocking strategy we propose is based on the length of the input queue. An LP remains in a blocking state until the length of its input queue becomes greater than a threshold L_{min}.

Unlike blocking strategies that are based on a blocking time, one can not set the threshold L_{min} to an arbitrary initial value L_{init} without facing a deadlock configuration. This is because the population of messages in the system may be finite and because messages are often linked each other. Therefore, if no messages are processed, because of the length condition, then no messages are produced. So, if L_{min} is greater than the number of messages that can arrive in the input queue, a deadlock occurs because every LP is waiting after the other LPs. To overcome this deadlock problem, the length condition should be relaxed after a given time. This is done by decrementing the value of L_{min} after each D_t unit of time (ut). In doing so, the length condition is relaxed step by step. A very aggressive behavior is obtained when $L_{min} = 1$ because the LP is ready to execute the first incoming message. Figure 1 depicts the algorithm.

Setting L_{init} and D_t

There are two alternatives for the choice of L_{init} and D_t: either set L_{init} large and D_t small, or set L_{init} small and D_t large. In the first solution, if messages arrive rapidly then the length condition will be rapidly satisfied. Now if messages arrive slowly then the threshold L_{min} will however decrease rapidly to avoid useless wait. In the second solution, if messages arrive rapidly then the length condition will be rapidly satisfied like previously but, if it is not the case the process may waste a lot of time in blocking until the length condition will be relaxed. We prefer the first solution that is, in a sense, more adaptive. The underlying hope of a length-based blocking strategy is that messages arrive rapidly at an LP avoiding in most cases useless wait. Now, the initial threshold L_{init} should not

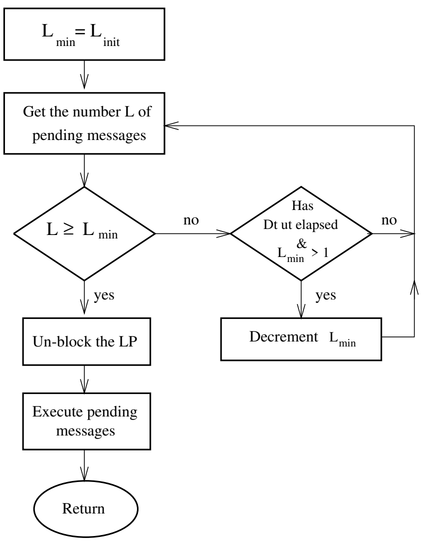

Figure 1. Flow chart of the blocking algorithm

be set too high unless the length condition would be too hard to satisfy. It may be very attractive to set this threshold quite high in order to lower the time error probability to a very minimum. The extreme point where L_{init} is so large that the blocking LP has in its input queue at least one message from each input process is theoretically equivalent to a conservative approach if deadlocks can be avoided. This is not the goal of this blocking strategy. Some risk is necessary in order to achieve good speedup when compare to a conservative behavior.

5. Modeling the ATM network

Future B-ISDN networks will be based on the Asynchronous Transfer Mode (ATM) to perform the high-speed transfer of data cells between end-points of the network by means of switches. At the cell-level two events are considered: a receive event $< recv, t_r >$ that indicates a cell reception event at time t_r and a send event $< send, t_d >$ that indicates a cell departure at time t_d. We now describe the switch model and the network model.

The switch model

The switch model we consider is non-blocking and has output buffers (see [8] for more details) with the following switching strategy: in one time slot each cell that arrives on an input port is switched to the corresponding output queue. The choice of the output follows some fixed routing probabilities. Only one cell can arrive per time slot on an input but severals cells can be switched to the same output in the same time. In this case they are buffered in the output queue. On the other hand, for all outputs the head cell in the queue is transmitted on the link in each time slot (FCFS server). Upon reception of a $< recv, t_r >$ message, the switch schedules for itself a $< send, t_d >$ event where t_d is the time of departure of the cell at the output queue. A cell from a given input i is switched to an output j with probability p_{ij} and sent out by scheduling at the receiver a $< recv, t_a >$ message where $t_a = t_d + link\_delay$. Figure 2 shows the switch model of the study.

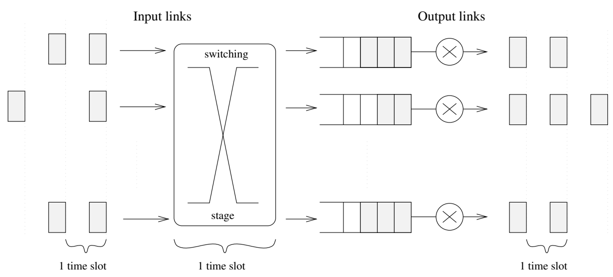

Figure 2. Switch model.

The network model

The network model consists of an ATM network of N_s switches fully connected. One traffic source per switch is considered that generates up to n_c cell reception messages. The links between switches are bi-directional, therefore each switch is considered to have N_s inputs and $N_s - 1$ outputs. The traffic sources generate cell reception messages with inter-arrival time that follows a given distribution. When all traffic sources do not generate cells anymore, the number of cells in the network remains constant and all switches keep sending existing cells to each other until the simulation ends. We set for the simulation $p_{aj} = 1/(N_s - 1)$ for all switches a and output j. Propagation delay between two switches takes 2 time slots. Figure 3 shows the network model for $N_s = 5$.

Our ATM network model is quite similar to the PHOLD model introduced by Fujimoto [6]. Using the associated terminology, our simulation model has a locality, expressed in fraction of nodes, of $\frac{N_s-1}{N_s}$ that is near maximum. In addition, the computation grain is kept small because no extra delay in event processing is inserted. The message density is n_c since each switch is considered to receive n_c cells from its connected source. The amount of lookahead is directly derived from delays associated with ATM processing (switching time, link delay) and, when expressed in term of *lookahead*

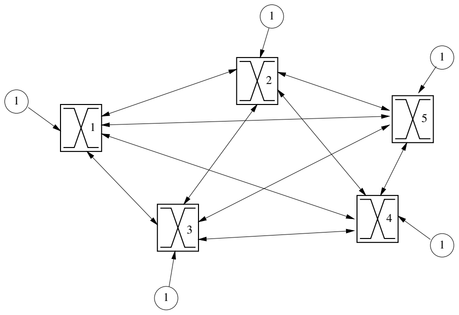

Figure 3. Network configuration

ratio (LAR) as defined in [4], the model exhibits a high degree of lookahead since $LAR = 1.0$.

High connectivity associated with a small computation grain make the network model a challenging optimistic simulation problem even if the size is small. Based on the results presented in [6], we can expect that a Time Warp system with *aggressive cancellation* would yield significant number of rollbacks which may be very costly especially in a distributed environment (i.e. not shared memory). Conversely, the high degree of lookahead will tend to advantage a conservative approach.

6. Local estimations

The life of a process consists of receiving, processing incoming messages and generating output messages. Processing a message often changes the state of the process and may cause the sending of output messages to other processes.

There is no way but to roll back if one wants to correct the bad computations propagated by bad produced messages. Now if one accepts the errors introduced by not cancelling produced messages then there is no need to roll back. We propose local estimations based on historical information as a substitute to rollbacks. Such historical information can be included within the process state and periodically saved in a Time Warp-like fashion. When a past message m_{past} arrives, the process virtual time does not change and saved historical information before t_{past} are used to process m_{past}. Re-evaluation of saved states is also performed to take into account the changes introduced by m_{past}. There is no cancellation of bad computations and in some cases this approach may be similar to the *lazy-cancellation* mechanism [10].

Estimations work because in the ATM network the semantic of a cell reception event is very limited. Therefore, the processing of some events in partial order does not dramatically affect the simulation result.

Two additional reasons justify the utilization of local estimations in simulation of ATM networks. First, time errors that may occur involve a lot of LPs resulting in long rollback chains as a rollback may cause additional costly "remote" rollbacks in a cascaded manner. Second, since LPs in our network model synchronize very often because of the large number of remote messages, this tends to produce a lot of causality violations of small distance[2]. If message synchronization is reduced, this would result in more occasional causality violations of larger distance and a rollback mechanism will be preferable but in our case, it is not.

Computation of local estimations

Let L_s be the list of saved states and $S_i = (o_1^i, .., o_j^i, last_1^i, .., last_j^i, n_r^i, n_s^i)$ be the ith saved state. o_j^i and $last_j^i$ are respectively the number of cells waiting in the output queue j and the *last time* that a cell has been switched to output j. n_r^i and n_s^i are respectively the number of received and sent cells. State saving for local estimations purpose is done after every processing of an $< recv, t_r >$ event. The timestamp of S_i is T_i. Index i increases as T_i increases. Upon reception of an outdated cell reception event $< recv, t_r >$, $t_r < LVT$, 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 $T_i \le t < T_{i+1}$.

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

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

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

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

$$\blacksquare\ (2) \begin{cases} \text{if } t_r = last_j^i \text{ then } t_w = max\{o_j^i - 1, 1\} \\ \text{else } t_w = max\{o_j^i - 1 - (t_r - T_i), 0\} \\ t_d = t_r + t_w + 2 \end{cases}$$

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

[2]The distance of an error is similar to the rollback length except that in our case no rollback is performed.

$$\blacksquare\ (3) \begin{cases} o_j^{i+1} = max\{o_j^i - (t_r - T_i), 0\} + 1 \\ last_j^{i+1} = t_r \\ n_r^{i+1} = n_r^i + 1 \\ n_s^{i+1} = n_s^i + min\{o_j^i, t_r - T_i\} \end{cases}$$

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

$$\blacksquare\ (4) \begin{cases} \text{if } T_k - t_r < o_j^{i+1} \text{ then } o_j^k = o_j^k + 1 \\ \quad \text{else } o_j^k \text{ is unchanged} \\ last_j^k \text{ is unchanged} \\ n_r^k = n_r^k + 1 \\ \text{if } t_d < T_k \text{ then } n_s^k = n_s^k + 1 \\ \quad \text{else } n_s^k \text{ is unchanged} \end{cases}$$

7. Experimental results

The implementation is done in C using the PVM package [14] for inter-processes messages passing. The simulation tests are carried out on a single monoprocessor SparcStation 20 with 64MB of memory (configuration 1) and then on a Connection-Machine (CM5) with 32 processors (configuration 2) of the *Centre National de Calcul Parallèle en Sciences de la Terre (CNCPST)*. In this case there is only one LP per processor and CM/PVM that is an optimized version of PVM is used. Care was also taken to ensure that the simulation was the sole user load on the CM5.

The network model of Figure 3 with $N_s = 5$ is simulated. We focus on the mean buffer length (MBL) of the ATM switches expressed in number of cells. Only the results for the output 1 of SW1 are reported since the other results are quite similar. With configuration 1, the number of cells in the network is set to 100 ($n_c = 20$) and we experimented with 10 values for L_{init}: 1, 2, 4, 5, 6, 8, 10, 11, 13 and 15. D_t is set to 15ms for all simulations. To test the accuracy of the algorithm, the results are compared to those obtained by a conservative approach. Simulations stop when each switch has simulated 20,000 cell receptions. With configuration 2, an automatic tuning of L_{init} is performed under a different traffic load ($n_c = 10$).

The performance metrics used are the speedup relative to a conservative simulation and the *Time Error Ratio* (TER) metric defined as follows:

$$TER = \frac{\text{Number of causality violations}}{\text{Number of incoming events processed}}$$

The numerator represents the number of causality violations, caused by outdated incoming events, that are corrected by means of local estimations. Therefore the TER represents the proportion of events that should trigger the rollback mechanism in a traditional optimistic simulation.

7.1. Empirical optimal value for L_{init}

Figure 4 shows the speedup as the initial value L_{init} is varied. The horizontal axis indicates the different values of L_{init}. Values for the speedup obtained with the corresponding value of L_{init} are represented on the y-axis. It is interesting to see that the interpolated

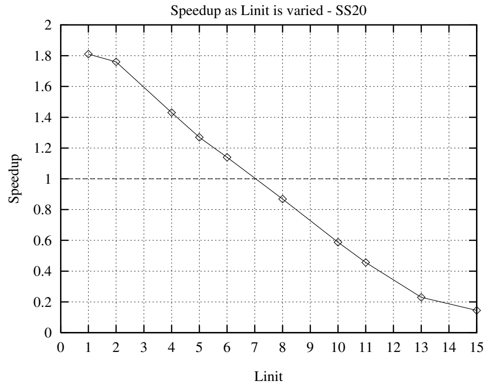

Figure 4. Speedup as L_{init} is varied

curve is almost linear in function of L_{init}. When the initial threshold L_{init} is set too high, the speedup dramatically drops because the LP is wasting too much time in blocking. Interesting values of L_{init} should, of course, give a speedup value greater than 1.

The TER is shown in Figure 5 with the different values for L_{init} represented on the x-axis. This convex curve presents an interesting minimum for $L_{init} = 10$. As L_{init} increases from 1 to 10, the blocking strategy successes in reducing the TER by allowing more events to be received in the pending queue. However, for values greater than 10, the TER starts to increase again. This strange phenomenon that we have not predicted can be explained as follows. Since L_{init} is large, the length condition is very hard to satisfy. Therefore every LP blocks and waits for messages from other LPs. Unfortunately, messages in our network model are tightly linked to each other, so all LPs are inter-blocked because no messages are produced. When L_{min} becomes small, some LPs are brutally triggered out of their waiting state and start to process their messages again, producing therefore output messages. As a consequence, the other LPs can receive more messages and, since their threshold L_{min} is now very small, they immediately process the incoming messages in a very aggressive manner resulting in more time errors. This side

L_{init}	MBL	% of error	TER
15	3.48	+5.51%	0.123
13	3.40	+3.03%	0.091
11	3.45	+4.54%	0.060
CMB	3.30	0%	0
10	3.20	-2.75%	0.044
8	3.19	-3.10%	0.080
6	3.15	-4.55%	0.196
5	2.93	-10.97%	0.270
4	2.85	-13.52%	0.330
2	2.65	-19.70%	0.500
1	2.52	-23.40%	0.580

Table 1. % of error and TER as L_{init} is varied

effect of blocking too large is similar to the problem of blocking to long as described in [20].

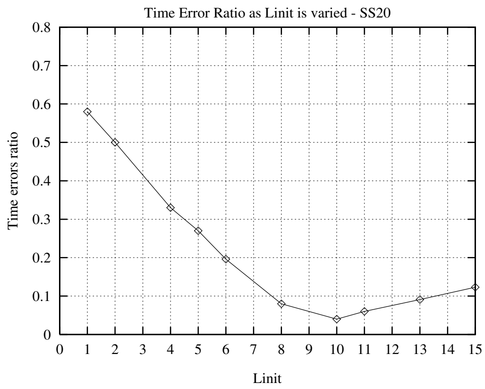

Figure 5. TER as L_{init} is varied

Table 1 compares the mean buffer length obtained with a conservative method and the one obtained by using local estimations and the blocking strategy. The associated TER is also indicated. With $L_{init} = 1, 2$, the results obtained are quite below those obtained by a correct approach with an error greater than 20%. As L_{init} increases, the accuracy increases since the number of time errors is greatly reduced (cf. Figure 5). With $L_{init} = 4, 5$ the percentage of errors does not exceed 15%. With $L_{init} \geq 6$ the error becomes less than 5%.

Finally, in the light of the results obtained, the best trade-off between accuracy and speedup is realized with $L_{init} = 4, 5$. Nevertheless, these values depends on parameters such as the computer speed and the delay introduced by the communication system. If the simulation is carried out on a different machine, the optimal values for L_{init} will unfortunately change. For instance, on a real multiprocessor architecture, the optimal values for L_{init} will certainly be smaller due to the increase of processing power.

7.2. Automatic tuning of L_{init}

Changing the value of D_t is not really interesting since it will simply shift the optimal value for L_{init}. The choice of the initial value for L_{min} is now essential because it affect directly the correctness of the simulation. Ideally, one should give the accuracy desired, i.e. error less than 15% for example, and let the simulation engine determines the associated value for L_{init}.

The idea is to specify a level of error by using the TER as a metric. When looking at table 1, one can theoretically associate a TER_{ref} value with a level of error even if the environment change because the accuracy should only depend on the number of time errors that are handle by local estimations. Finding the associate value for L_{init} can therefore be done at runtime by monitoring the simulation progress and compute a TER_{comp}. The initial value for L_{min} that produces the theoretically desired level of accuracy should produce a TER_{comp} equal or less than the TER_{ref}.

Practically, one way to find the optimal value for L_{init} is to begin with an initial value of 10 for example, and compute a TER_{comp} for every N processed events. The TER_{comp} obtained can then be compared to the specified TER_{ref} in order to adjust (increase or decrease) L_{init} until TER_{comp} becomes equal or lesser than TER_{ref}. The tuning algorithm is described below in a C-like syntax.

```
TER_tmp=INFINITY;
found=FALSE;
stop_if_lesser=stop_if_greater=FALSE;
L_init = 10;

do {

Simul(N);
if (not found)
{
    TER_comp=ComputeTER();
    if (TER_comp < TER_ref) {
        if (stop_if_lesser) found=TRUE;
        else { L_init = L_init - 1;
               stop_if_greater=TRUE; } }
    else {
            if (stop_if_greater) {
                L_init = L_init + 1;
                found=TRUE; }
            else {
                    if (TER_comp > TER_tmp) {
                        L_init = L_init - 1;
                        TER_tmp = 0;
                        stop_if_lesser=FALSE; }
                    else {
                            TER_tmp = TER_comp;
                            L_init = L_init + 1;
                            stop_if_lesser=TRUE; }
                    }
            }
} } while(not end_of_simul);
```

TER_{ref}	TER_{comp}	MBL	% of error	L_{init}
0.044	0.044	1.27	+1.56%	6
CMB	0	1.25	0%	—
0.080	0.061	1.22	-1.69%	5
0.196	0.121	1.20	-3.72%	4
0.270	0.268	1.08	-13.6%	3
0.330	0.270	1.07	-13.6%	3
0.500	0.498	0.93	-25.6%	2
0.580	0.496	0.93	-25.6%	2
1.0	0.740	0.77	-33.88%	1

Table 2. L_{init} as TER_{ref} is varied

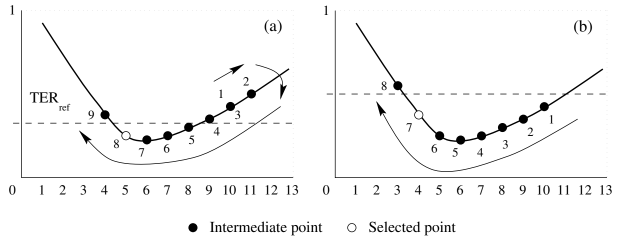

Figure 6. Initial point on the bad branch.

At the first iteration, N can be chosen large in order to reach a steady state whereas subsequent iterations can be realized with a smaller value for N. What complicates the algorithm is the possibility when specifying a TER_{ref} to be on the bad branch of the TER curve (graph a and b). In this case, incrementing L_{init} does not lower the TER but do increase it (graph a). So, we must detect this increase of the TER in order to adjust L_{init} in the right way. Generally, the number of iterations is not large since the range of possible values for L_{init} is limited. In table 2 a TER_{ref} is pre-specified and the TER_{comp} obtained on a CM5 with the corresponding L_{init} value found is returned. The error compared to a conservative result is also shown.

Figure 7 shows the speedup obtained on a CM5. This curve is no more linear and presents a drop of the speedup for small value of L_{init}. One of the possible explication is that on a CM5, the computation overhead introduced by the local estimations can become greater than the communication overhead between LPs when a lot of time errors occur. The previous curve obtained on a SS20 is almost linear because in this case the communication overhead associated with the context switching overhead is always greater than the computational one making the latter less significant. With $L_{init} = 4$, the simulation speed is 1.8 times greater than the conservative one for a corresponding $TER_{comp} = 0.121$ and an error of -3.72%. This is certainly the best trade-off between accuracy and speedup. Figure 8 compared the TER obtained on an SS20 (previously plotted) with the one obtained on a

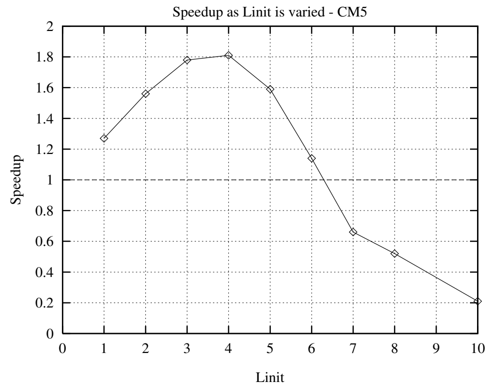

Figure 7. Speedup as L_{init} is varied — CM5

CM5 as L_{init} is varied. The Blocking Ratio (BR) on the CM5, defined as the ratio between the time spent in blocking and the total simulation time, is also shown. We observe a more rapid decrease of the TER on the

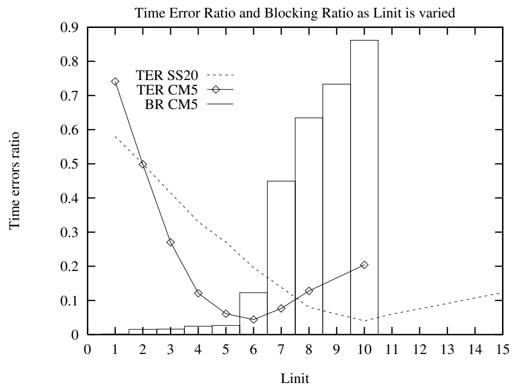

Figure 8. TER and BR as L_{init} is varied

CM5 and a shift of the L_{init} value that minimizes the TER curve (the minimum TER remains the same for both environment). Both phenomenon are explained by the small communication latency and the increase of processing power on the CM5 making messages to be passed more rapidly between LPs. As previously, the TER starts to increase again when L_{init} is set too large. In addition, when $L_{init} < 6$ the BR is lesser than 3% keeping the blocking overhead very small. For instance, with $L_{init} = 4$ (optimal value) we have a BR of only 2.45% whereas the observed decrease of the TER from $L_{init} = 1$ (no blocking) is of 83%.

7.3. Varying the size and the message density

Only preliminary results have been conducted in increasing the size of the network (N_s) and the message density (n_c). However we can assert some behaviors of the blocking algorithm. First, increasing N_s does not increase linearly the optimal value of L_{init} even if the number of input LPs increases. Second, it appears that increasing N_s makes the algorithm to reduce more rapidly the number of time errors. The counterpart is that accurate tuning is difficult or impossible since L_{init} can only take integer values.

Now if the message density is larger, performance of the conservative method is likely to be improved because the null-message traffic is reduced. But if the size of the network is also large, then the speedup can be larger than what we obtained. Extensive studies is however required to measure more precisely the effect of increasing the network size and the message density.

8. Conclusion

A simple preventive blocking strategy based on the length of the input queue is proposed. This blocking strategy aims at reducing the number of time errors. As the number of time errors decreases, local estimations work better to give more accurate results. Simulations of an ATM network with ad-hoc local estimations give favorable results and error less than 5% can be achieved for a speedup relative to a conservative simulation of 1.8.

We found that the initial threshold L_{init} should not be chosen too large because the speedup dramatically drops for no significant improvement in the accuracy of the results. Moreover, large values for L_{init} have the side effect to increase the number of time errors by introducing bad blocking decisions. Now, when correctly tuned, this simple length-based blocking strategy succeeds in reducing the number of causality violations. Experimental results show that the length-based blocking strategy performs better when messages arrive rapidly. In this case, the optimal value for L_{init} is relatively small and incrementing or decrementing L_{init} by 1 causes substantial changes in the value of the TER. This is very advantageous since one can greatly reduce the TER with a very small blocking overhead.

A tuning algorithm for finding the optimal value for L_{init} when giving a TER_{ref} corresponding to a level of accuracy has also been proposed. However, we found that the desired level of accuracy could only be met approximately.

References

[1] D. Ball and S. Hoyt, "The Adaptive Time-Warp Concurrency Control Algorithm", *Proc. of the 1990 SCS Multiconference on Distributed Simulation, pp174-177.*

[2] K. M. Chandy and J. Misra,"Distribution Simulation: A Case Study in Design and Verification of Distributed Programs", *Transactions on Software Engineering, 5(5), pp440-452.*

[3] A. Ferscha and J. Lüthi, "Estimating Rollback Overhead for Optimism Control in Time Warp", *Proc. of the 28th Annual Simulation Symposium, pp2-12.*

[4] R. M. Fujimoto, "Lookahead in Parallel Discrete Event Simulation", *Proc. of the 1988 Int. Conf. on Parallel Processing, pp34-41.*

[5] R. M. Fujimoto, "Parallel Discrete Event Simulation", *Communication of the ACM, 33(10), pp31-53.*

[6] R. M. Fujimoto, "Performance of Time Warp under synthetic work-loads", *Proc. of the 1990 SCS Multiconference on Distributed Simulation, pp23-28.*

[7] D. O. Hamnes and A. Tripathi, "Investigations in Adaptive Distributed Simulation", *Proc. of 8th Workshop on Parallel and Distributed Simulation, pp20-23.*

[8] R. Händel, M. N. Nuber and S. Schröder, "ATM Networks: Concepts, Protocols, Applications", *Addison-Wesley, 1994.*

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

[10] Y. Lin, "Estimating the Likelihood of Success of Lazy Cancellation in Time Warp Simulations", *To appear in the Int. Journal in Computer Simulation*

[11] B. Lubachevsky, A. Shwartz and A. Weiss, "Rollback Sometimes Works ... If Filtered", *Proc. of the 1989 Winter Simulation Conference, pp630-639.*

[12] Y. Matsumoto and K. Taki, "Adaptive Time-Ceiling for Efficient Parallel Discrete Event Simulation", *Proc. of the 1993 Object-Oriented Simulation Conference, pp101-106.*

[13] C. D. Pham and S. Fdida, "The Statistically Correct Approach to Distributed Simulation of ATM Network", *Proc. of the 6th IFIP Conference on Performance of Computer Networks, pp352-366*

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

[15] P. L. Reiher and D. Jefferson, "Limitation of Optimism in the Time Warp Operating System", *Proc. of the 1989 Winter Simulation Conference, pp765-769.*

[16] P. F. Reynolds, "A Shared Resource Algorithm for Distributed Simulation", *Proc. of the 9th Int. Computer Architecture Conference, pp259-266*

[17] P. F. Reynolds, "A Spectrum of Options for Parallel Simulation", *Proc. of the 1988 Winter Simulation Conference, pp325-332.*

[18] L. M. Sokol, B. K. Stucky and V. S. Hwang, "MTW : A Control Mechanism for Parallel Discrete Simulation", *Proc. of the 1989 Int. Conference on Parallel Processing, pp250-254.*

[19] S. Srinivasan and P. F. Reynolds Jr.,"NPSI Adaptive Synchronization Algorithms for PDES", *To appear in Proc. of the 1995 Winter Simulation Conference.*

[20] S. Srinivasan and P. F. Reynolds Jr.,"Adaptive Algorithms vs. Time Warp: An Analytical Comparison", *To appear in Proc. of the 1995 Winter Simulation Conference.*

[21] J. S. Steinman, "Interactive SPEEDES", *Proc. of the 24th Annual Simulation Symposium, pp149-158.*

[22] M. Theofanos, "Distributed Simulation of Queuing Networks", *Master's Thesis, The University of Virginia, January 1984.*