Rollback Overhead Reduction Methods for Time Warp Distributed Simulation

M.S. Balsamo[•] and C. Manconi*

[•] Dipartimento di Matematica e Informatica, University of Udine
Vial delle Scienze 108, Udine, Italy, e-mail: balsamo@dimi.uniud.it

* Dipartimento di Informatica, University of Pisa
Corso Italia 40, Pisa, Italy, e-mail: manconi@di.unipi.it

Abstract

Parallel discrete event simulation is a useful technique to improve performance of sequential discrete event simulation. We consider the Time Warp algorithm for asynchronous distributed discrete event simulation. Time Warp is an optimistic synchronization mechanism for asynchronous distributed systems that allows a system to violate the synchronisation constraint and, in this case, make the system to rollback to a correct state. We focus on the kernel of the Time Warp algorithm, that is the rollback operation and we propose some techniques to reduce the overhead due to this operation. In particular we propose a method to reduce the overhead involved in state saving operation, two methods to reduce the overhead of a single rollback operation and a method to reduce the overall number of rollbacks. These methods have been implemented in a distributed simulation environment on a distributed memory system. Some experimental results show the effectiveness of the proposed techniques.

Keywords: distributed simulation, rollback overhead, simulation modelling.

---

1. Introduction

Parallel discrete event simulation is a useful technique to improve performance of sequential discrete event simulation [4,5,7,9]. Virtual time [6] is a paradigm for optimistic synchronization of asynchronous distributed systems which allows a system to violate the synchronisation constraint and, in this case, make the system to rollback to a correct state. Time Warp mechanism is a method to implement virtual time. We consider the Time Warp algorithm for asynchronous distributed discrete event simulation.

A Time Warp simulation [5,6,7] consists of a set of asynchronous processes, called logical processes. Each logical process simulates a submodel of the main simulation model. Specifically, a logical process simulates a portion of the distributed event list, manages its own simulation clock, called local virtual clock, and communicates with others processes by time-stamped messages. Each process manages the local event list called input queue. It gets events from its local event list, computes events and at each computation eventually produces some new events. The local virtual clock value is updated with the time-stamp of the computed event. New produced events can involve both the producer process and other processes. In the former case new events are stored in the input queue of the producer process, in the latter case they are sent to the involved process by a time-stamped message, called event message. New received event messages are stored by the receiving process in its own input queue.

The asynchronous behavior of the processes allows different values of the local virtual clocks of different processes. For this reason it is possible that a process receives an event message holding a time-stamp less than its local virtual clock called straggler message. In this case the process rollbacks to a simulation time less than the straggler's time-stamp, it cancels the computation performed at a time greater than that value and it computes forward again. Processes periodically save a copy of their own state into a queue called state queue, in order to restore a correct state when a rollback occurs.

Rollback operation, that is the kernel of Time Warp, consists of three fundamental steps: *restoration*, *cancellation* and *coasting-forward* phase [6,7].

In the restoration phase, the process considers the set of saved states with the local virtual clock value less than the time-stamp of the straggler, it selects the state with the maximum local virtual clock and updates the local virtual clock to this value.

In the cancellation phase the effect of both local and non local incorrect computations are cancelled. To this aim the saved states and the messages computed in the time interval between the time-stamp of the straggler and the local virtual clock before the arrival of the straggler are removed, in order to cancel the local incorrect computation. To remove non local effects of incorrect computations the rolled back process sends a copy of each message already sent with a time-stamp in that time interval. These copies are called antimesssages or negative messages, because they are marked by a negative sign. When a process receives a negative message it annihilates both positive and negative copies of that message.

As concerns the state saving operation we observe that since the state of the processes is not saved at each event computation, then the time restored in the restoration phase can be less than the time-stamp of the straggler message. After cancellation the coasting forward phase computes again each event in the time interval between the restored time and the time-stamp of the straggler, so obtaining the new state just before the straggler.

It is easy to argue that the performance of the Time Warp distributed simulation algorithm depends on the efficiency of the rollback operation and the number of rollbacks. Therefore it is important to analyse the rollback operation to obtain an estimation of its effectiveness on the distributed simulation algorithm. This analysis leads to the identification and definition of some rollback overhead reduction methods. In this paper we present some techniques to reduce rollback overhead. We follow two main approaches at two different levels: an algorithmic approach, by considering some modification of the distributed simulation algorithm and an implementation approach.

In particular we propose a method to reduce the overhead involved in state saving operation, two methods to reduce the overhead of a single rollback operation and, finally, a method to reduce the overall number of rollbacks. These methods have been implemented in a distributed simulation environment on a distributed memory system. Some experimental results show the effectiveness of the proposed methods.

In the following, in the next section we present a simple model of the Time Warp simulation algorithm and the rollback operation. Section 3 introduces the main approaches that can be applied to reduce rollback overhead. In Section 4 we propose four methods to reduce the rollback overhead. Section 5 introduces the simulation environment and Section 6 presents some experimental results to test the proposed methods and their effectiveness on Time Warp simulation performance. Finally, Section 7 presents the conclusions.

2. A rollback model

During a Time Warp simulation rollback and forward computation phases alternate each one another. Then, in a simple model, a Time Warp simulation can be viewed as a sequence of computation cycles, each composed by a rollback phase and a normal forward execution phase, as shown in [8].

In this model let T_{TW} denote the time to perform Time Warp simulation, also called total execution time. Let $T_{C,i}$ denote the time to perform the i-th computation cycle and N the number of cycles performed during the simulation. Then the total execution time can be expressed as follows:

$$T_{TW} = \sum_{1 \le i \le N} T_{C,i}. \tag{1}$$

Consider the i-th computation cycle. It consists of a rollback phase and a normal forward execution time, as shown in Fig. 1. Let $t_{r,i}$ be the time of the i-th rollback phase and $t_{fwd,i}$ is the time of the i-th forward execution phase, $1 \le i \le N$. Then the time of cycle i can be expressed as follows:

$$T_{C,i} = t_{r,i} + t_{fwd,i} \tag{2}$$

which yields to

$$T_{TW} = \sum_{1 \le i \le N} t_{r,i} + \sum_{1 \le i \le N} t_{fwd,i}. \tag{3}$$

Now, let T_r denote the total rollback time, that is the summation of single times spent performing rollbacks, and T_{fwd} the total forward execution time, that is the summation of single times spent in forward executions, i.e.:

$$T_r = \sum_{1 \le i \le N} t_{r,i} \quad \text{and} \quad T_{fwd} = \sum_{1 \le i \le N} t_{fwd,i} \tag{4}$$

then the total time T_{TW} can be expressed as

$$T_{TW} = T_r + T_{fwd}. \tag{5}$$

4

Consider the i-th forward execution phase: the major operations during this phase are events' computation and state savings. Then the forward execution time in the i-th cycle can be expressed as follows: $t_{fwd,i} = t_{ec,i} + t_{ss,i}$, where $t_{ec,i}$ denotes the events' computation time in the cycle and $t_{ss,i}$ the state saving time in the cycle. Using these assumptions the total forward execution time is given by

$$T_{fwd} = \sum_{1 \le i \le N} (t_{ec,i} + t_{ss,i}) = \sum_{1 \le i \le N} t_{ec,i} + \sum_{1 \le i \le N} t_{ss,i} \qquad (6)$$

On the other hand, the rollback operation is composed by three phases: restoration, cancellation and coasting-forward, as we discuss in the previous section. Let $t_{rs,i}$ denote the time required to perform restoration in the i-th cycle, $t_{cn,i}$ the time to perform cancellation and $t_{cf,i}$ the time to perform coasting-forward. Then the time spent in the i-th rollback operation $t_{r,i}$ is given by

$$t_{r,i} = t_{rs,i} + t_{cn,i} + t_{cf,i} \qquad (7)$$

and the total rollback time can be expressed as

$$T_r = \sum_{1 \le i \le N} (t_{rs,i} + t_{cn,i} + t_{cf,i}) = \sum_{1 \le i \le N} t_{rs,i} + \sum_{1 \le i \le N} t_{cn,i} + \sum_{1 \le i \le N} t_{cf,i}$$
$$(8)$$

Hence, by formulas (5) and (6) we obtain the following expression for the total execution time T_{TW}:

$$T_{TW} = T_r + \left(\sum_{1 \le i \le N} t_{ec,i} + \sum_{1 \le i \le N} t_{ss,i} \right) \qquad (9)$$

where T_r is given by formula (8). Note that the state saving is a portion of the forward execution phase, but it is performed to restore a correct state when a rollback occurs. Then we can consider the state saving time as a portion of the overall overhead due to rollback. With this assumption, finally, formula (9) can be rewritten as follows:

$$T_{TW} = \Omega + \sum_{1 \le i \le N} t_{ec,i} \qquad (10)$$

where

$$= T_r + \sum_{1 \leq i \leq N} t_{ss,i} \qquad (11)$$

is the overall rollback overhead.

In the next section we introduce the rollback overhead reduction methods based on the reduction of each single term in formula (11).

3. Rollback overhead reduction method

In order to reduce rollback overhead we can apply three algorithmic techniques as we can deduce by formula (11): methods to reduce the number of rollbacks N, named Rollback Number Reduction methods (RNR), methods to reduce the total rollback time T_r, called Rollback Cost Reduction methods (RCR) and those to reduce the total state saving time $\sum_{1 \leq i \leq N} t_{ss,i}$, called Rollback State-saving Reduction methods (RSR).

RNR methods reduce N, that is the number of cycles performed to complete simulation. Using these methods we can obtain both a reduction of the total number of computed events and a reduction of each single term in formulas (8), (10) and (11). On the other hand it is also possible that a reduction of the number of rollbacks leads to an increase of the width of a single rollback operation. Hence, in order to optimize the algorithm performance it is necessary to trade-off between the increase of rollbacks' width and the reduction of the overall number of rollbacks. An example of an RNR method that selects the appropriate trade-off between rollback width and rollback number is proposed in [2].

As we can argue by observing formula (8) to reduce the rollback cost, that is the time spent to perform rollback operations T_r, it is necessary to reduce at least one of the three terms: the total execution time spent in restoration, the total execution time spent in cancellation and the total execution time spent in coasting-forward. RCR methods attempt to reduce these terms. In particular they reduce the time spent for restoration, cancellation and coasting-forward in the i-th computation cycle, so reducing the overall time. An example of RCR method is direct cancellation technique proposed in [3].

RSR methods reduce the cost of a single state saving operation, so reducing the overall state saving time in formula (11).

In the next section we present some RNR, RCR and RSR methods that can be used to reduce the overall rollback overhead   and, as a consequence, to increase the efficiency of Time Warp simulation.

4. The proposed rollback overhead reduction methods

In this section we present four methods to reduce the overall rollback overhead and, specifically a RSR method, two RCR methods that reduce the overall cancellation time and the overall coasting-forward time, respectively, and a RNR method.

4.1 A state saving reduction method

The proposed RSR method exploits the characteristics of the simulation model to reduce the dimension of the saved states. This lead to a reduction of the time to save each state, so reducing the overall state saving time, $\sum_{1 \leq i \leq N} t_{ss,i}$. As an example, consider a queueing network model simulation where we can define the state of the logical processes only by the local virtual clock value and the seeds of the random number generators [6]. We obtain other information, such as the queue length and the job waiting service, from the queues of the logical process of the distributed simulator. In this way it is possible to implement the processes' states with a very small number of variables. For this example two variables are sufficient to define the process state. By reducing the state dimension we can increase the frequency of the state saving, so reducing the coasting-forward time and the overall number of rollbacks.

4.2 Two rollback cost reduction methods

We consider the Time Warp simulation on distributed memory massively parallel processing systems. The first proposed RCR method, called *Lazy Sending* cancellation method, exploits the characteristics of most of the current distributed memory massively parallel processing systems, in which the message-passing environment guarantees the FIFO sending-arriving order for the messages. By using this property it is possible to reduce the rollback overhead by reducing the number of messages sent during the cancellation phase of the rollback. The basic idea is that

each process sends only one antimessage instead of a sequence of messages to each process involved in the rollback. In particular, we propose that each process sends only the antimessage related to the already sent message with the minimum time-stamp value. The receiver process removes all the received messages with a time-stamp greater than that of this when receives it. As a consequence, this reduces the cancellation time in the i-th computation cycle, $t_{cn,i}$.

Let t_{send} denote the time to send a negative message and let n_i be the number of negative messages that have to be sent in the i-th rollback phase using the standard cancellation method. Then $t_{send} \cdot n_i$ is the time required to perform cancellation. Under the same hypothesis, the Lazy Sending cancellation method only requires t_{send} time to perform cancellation. Then in the first case the overall cancellation time is given by

$$\sum_{1 \le i \le N} t_{send} \cdot n_i = t_{send} \cdot \sum_{1 \le i \le N} n_i \qquad (12)$$

and with Lazy Sending the cancellation time reduces to

$$\sum_{1 \le i \le N} t_{send} = t_{send} \cdot N \qquad (13)$$

Hence the proposed method drastically reduces the time to perform cancellation, so reducing the overall rollback overhead T_r.

The second proposed RCR method redefines the rollback mechanism by considering the causality relationships between events. In particular we define a new mechanism, called *back-forward*, that can substitute the conventional rollback operation when there is no causality relationship between the events involved in the computation. When a straggler message arrives at a logical process P_j, the local virtual clock of P_j *goes back* to the time of the last saved state with the local virtual clock less than the time-stamp of the straggler and removes all the saved states with to a local virtual clock greater than the restored one. This is the restoration phase. If the straggler message is negative, then the cancellation phase is performed by removing the positive copy of the straggler and by sending a copy of each message already sent when the local virtual clock was equal to the time-stamp of the straggler message. Then the local virtual clock *goes forward* to the value that the clock had before the arrival of the straggler. This corresponds to the coasting-forward phase.

8

The major cost of the *back-forward* operation is given by the restoration phase time $t_{rs,i}$ and by the cancellation phase time $t_{cn,i}$. However, we must observe that the cancellation phase is not ever performed. Then the cancellation time is null when the straggler message is positive and is equal to the time of sending a message when the straggler message is negative. The cost of the coasting-forward phase is given by the time of two memory accesses. Then the i-th rollback time reduces to $t_{rs,i}+t_{cn,i}+t_{cf,i}$, where $t_{cf,i}$ is negligible with respect to the rollback operation, $t_{cn,i}$ is either null or, in the worst case, except for the case in which a rollback operation is used, that is $t_{cn,i}, t_{cf,i} << t_{rs,i}$.

4.3 A rollback number reduction method

Finally, the proposed RNR method exploits the Artificial Forward Synchronization (AFS) protocol for memory management, proposed in [3], to reduce the rollback trashing behaviour. Indeed it is well known that the number of rollbacks in Time Warp simulation is prone to trashing behaviour if some processes are allowed to go too far in the future [2]. AFS protocol is a memory management protocol designed to solve the memory overflow problem for Time Warp simulation, based on a memory overflow avoidance approach. A fundamental feature of AFS protocol is that it limits the capacity of processes' input queues, so that only a limited number of messages can be stored in the input queues. In particular, the algorithm chooses queues' capacity so that the summation of the capacities of all the queues is less than the available memory in the system, so avoiding that memory overflow occurs. We use this characteristic of AFS protocol to limit the Time Warp optimism and to reduce the total number of rollbacks N, avoiding rollback trashing behaviour. Note that this approach leads to a reduction of the total number of events, hence a reduction of $\sum_{1 \le i \le N} (t_{fwd,i} + t_{r,i})$, but it can increase the rollback width. Then we have to consider the trade-off between rollback width and the number of rollbacks N. From experimental results we observe that the proposed method leads to reduction of the overall rollback overhead.

5. The simulation environment

To perform experimental studies to test the effectiveness of the proposed rollback reduction methods on the Time Warp simulation performance we used the *Perseus*

distributed simulator [1]. This distributed simulation environment has been developed at the Department of Computer Science of the University of Pisa, running on the Meiko Computing Surface, a distributed memory system based on Transputer T800 with 52 processing nodes and 1MB of local memory for each node.

We consider the number of Time Warp processes varying between ten and one hundred and with a number of processes per processor varying between one and four. We consider the class of queueing network models. We performed experiments by varying the model parameters including the number of service centers, service and arrival rates, network topology, routing probability and the number of customers of the network. We consider the class of queueuing networks with fork and join nodes as example of models where there is no causality relationship between some of the events involved in computation, so that we apply the back-forward mechanism of the second proposed RCR method.

The goal of the results that we present is to show the effectiveness of the four proposed methods to reduce the overall overhead due to rollback in Time Warp simulation. To show the effectiveness of the proposed RSR methods we studied how the state dimension affects the state saving time. We observed that when the state dimension is very small, it is easy to determine the optimal state saving frequency, which in most of the cases is very low. On the contrary, when the state dimension is high, it is more difficult to determine the optimal state saving frequency. These studies have been performed for various models by varying from the best case, which means that no rollback occurs, to the worst case, when the number of rollbacks is of the same order of the computed messages' number.

As concern the Lazy Sending cancellation method, we performed experiments studying the overall cancellation time in both cases, i.e., with Lazy Sending cancellation and the conventional cancellation method. To evaluate the effectiveness of the back-forward operation with respect to the rollback operation we studied the effect of introducing processes using back-forward operation on simulation completion time.

Concerning the evaluation of the AFS memory management protocol to reduce the rollback number, we studied how the memory available for processes affects the number of rollbacks.

To evaluate the performance of the simulation algorithm and to study the effectiveness of the proposed methods we consider the execution time of the simulation, the cancellation time, the state saving time and its frequency. We perform experiments by varying the dimension of the input queues of the logical processes to study the impact of AFS protocol on the algorithm performance.

6. Experimental results

The effectiveness of the four proposed methods is confirmed by some experimental results.

6.1 Evaluation of the proposed RSR method

In order to evaluate the performance of the proposed RSR method we present some experimental results obtained by a set of simulation runs involving up to 50 logical processes allocated on 25 processors. For each logical process we defined a state formed by six floating point variables. Figure 2 shows the execution time and the states saving time as a function of the state dimension, i.e., the number of floating point variables that compose each state. We observe that the execution time grows as the state saving time grows, but with a different rate. This behaviour is due to the cancellation within the rollback operation, which is affected by the modification of the saved states.

6.2 Evaluation of the Lazy Sending method

In order to evaluate the performance of the Lazy Sending cancellation method, by the experimental results we can derive some observations. The results shown in Fig. 3 refer to a queueing network model with central server topology. The simulation model has 10 logical processes allocated on 5 processors. We perform two sets of simulation experiments by using in the logical processes in one case the Lazy Sending cancellation method and in the other one the conventional cancellation method. Fig. 3 shows that there is a reduction of both execution and state saving time when we apply the Lazy Sending cancellation method. In the experiments we considered different values of the state saving frequency that does not affect the results on the performance of the proposed RCR method.

Moreover, we observed that the proposed technique drastically reduces the number of messages sent during the rollback so increasing the simulation speed-up. Figure 4 shows the comparison between the Lazy Sending cancellation method and conventional one in terms of number of messages; these results concern a queuing network model with tandem topology with feedback and 50 logical processes allocated on 5 processors. We considered various values of the input queue

cardinality to evaluate the impact on the algorithm performance. Figure 5 shows the speed-up obtained for the Lazy Sending method on this queueing model.

6.3 Evaluation of the back-forward method

To evaluate the performance of the back-forward method we performed a set of simulation experiments applied to a class of queueing network models that include fork and join nodes, which represent cuncurrency and synchronisation constraints. It is easy to verify that in these nodes there is no causality relationship between the events representing the completion of service of independent customers and hence in some events involved in the computation. Then we can apply the back-forward mechanism instead of the rollback operation. From experimental results we can observe that the back-forward method performs better than the rollback. Figure 6 shows the results for a fork and join queueing network with 10 logical processes allocated on 5 processors. The figure illustrates the execution time of Time Warp simulation using the two methods, by varying the dimension of the input queue.

6.4 Evaluation of the AFS protocol

As concerns the proposed RNR method we observed that the AFS protocol can contribute to reduce the number of rollbacks so increasing the simulation speed-up. Figure 7 shows the results for a queueing network with tandem topology and 25 logical processes allocated on 7, 13 and 26 processors, respectively. The figure shows the speed-up obtained by varying the dimension of the input queue of the logical processes.

7. Conclusions

Time Warp distributed simulation is a useful algorithm to enhance the performance of sequential discrete event simulation. In order to efficiently apply the Time Warp simulation it is important to reduce the overhead due to the rollback operations. In this paper we have presented four methods to reduce rollback overhead. In particular we have proposed a method to reduce the overhead involved in state saving operations (RSR method), two methods to reduce the overhead of a single rollback

(two RCR methods) and a method to reduce the overall number of rollbacks (RNR method).

From experimental results we studied the impact on simulation performance and the effectiveness of the four proposed methods. Specifically, we observe that the execution grows with the state saving time and that the lazy-sending method (the first RCR method) improves simulation speed-up. Moreover, when it can be applied, the back-forward technique (the second RCR method) performs better than rollback and produces a simulation performance improvement. Finally, we observe that the AFS protocol (the RNR method) can increase the simulation speed-up.

References

[1]   S.Balsamo and C.Manconi, Parallel Discere Event Simulation in a Distributed Memory System, in: Proceedings of European Simulation Multiconference, ESM'94, Barcelona, Spain (1994) 160-164.

[2]   S.R.Das, R.M.Fujimoto, An Adaptive Memory Management Protocol for Time Warp Simulation, in: Proceedings of ACM Sigmetrics 1994, Nashville, Tennessee (1994) 201-210.

[3]   R.M. Fujimoto, Time Warp on a Shared Memory Multiprocessor, in: Proceedings Int. Conference on Parallel Processing (1989) 242-249.

[4]   R.M. Fujimoto, Parallel Discrete Event Simulation, Communication of the ACM , 33 (10) (1990) 30-53.

[5]   R.M. Fujimoto, Parallel Discrete Event Simulation, in: Proceedings of Winter Simulation Conference (1989) 19-28.

[6]   D.R.Jefferson, Virtual Time, ACM Transaction on Programming Languages and Systems, 7 (3) (1985) 404-425.

[7]   D. Jefferson, H. Sowizral, Fast Concurrent Simulation Using the Time Warp Mechanism, in: Proceedings of SCS Multiconference on Distributed Simulation (1985) 63-69.

[8]   Yi-Bing Lin, B.R.Preiss, W.M.Loucks, E.D.Lazowska, Selecting the Checkpoint interval in Time warp Simulation, in: Proceedings of the 7th Workshop on Parallel and Distributed Simulation, PADS'93, San Diego, CA (1993) 3-10.

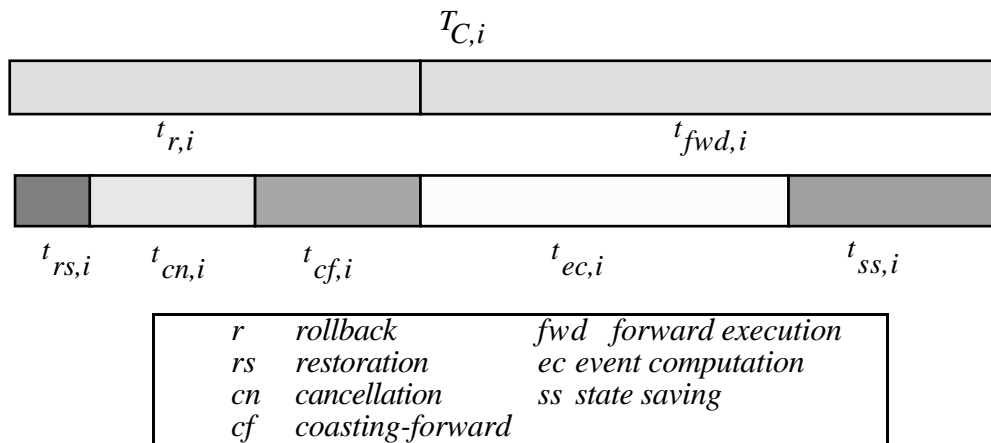[9]   J. Misra, Distributed Discrete Event Simulation, ACM Computing Surveys, 18 (1) (1986) 39-65.

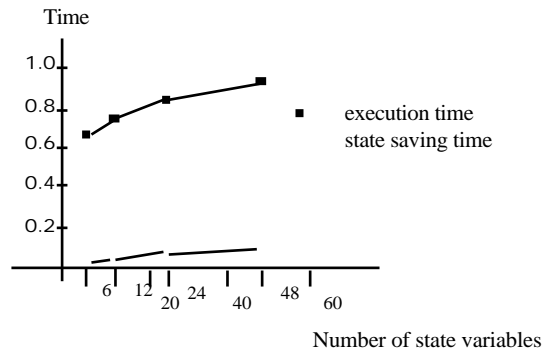Fig.1. The i-th computation cycle and its components.

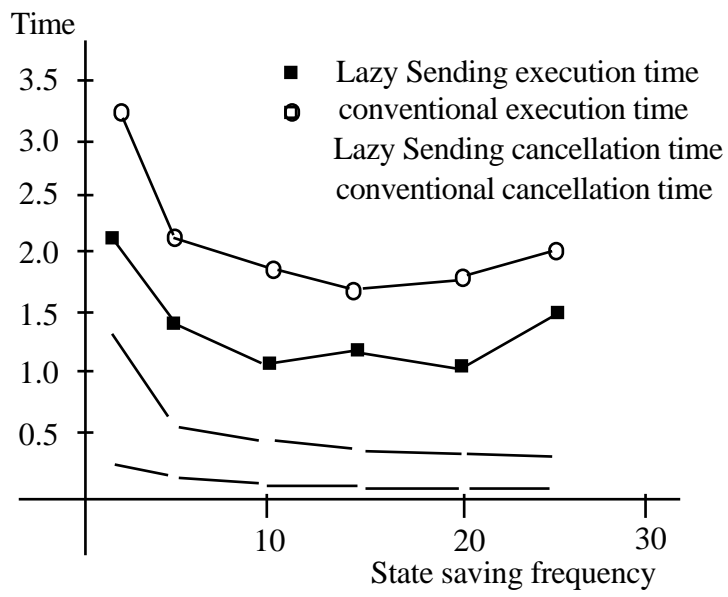Fig. 2. Execution and cancellation time versus state dimension.

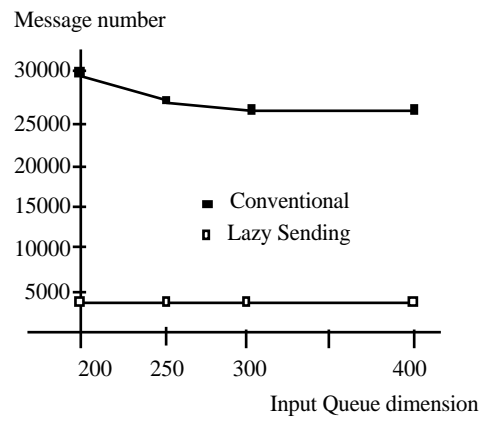Fig. 3. Execution and cancellation time in Lazy Sending and conventional cancellation.

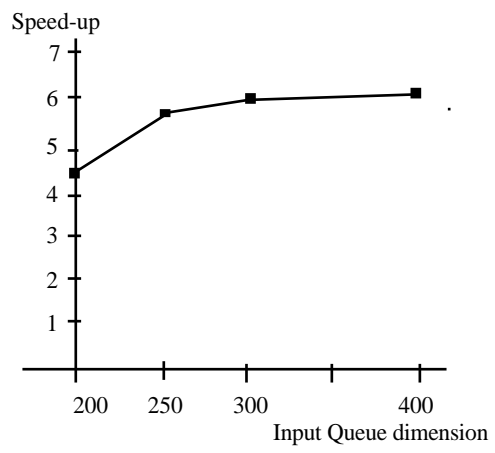Fig. 4. Number of messages in Lazy Sending and conventional cancellation.
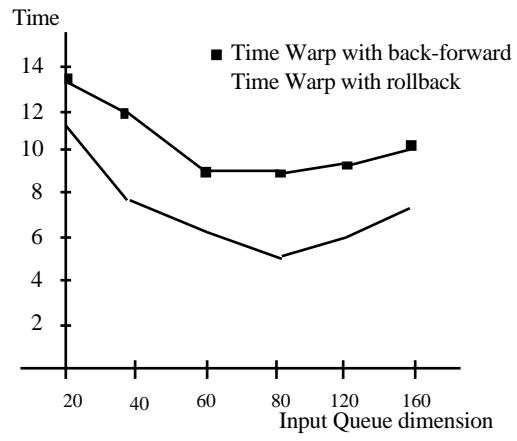
Fig. 5. Speed-up of Lazy Sending cancellation.
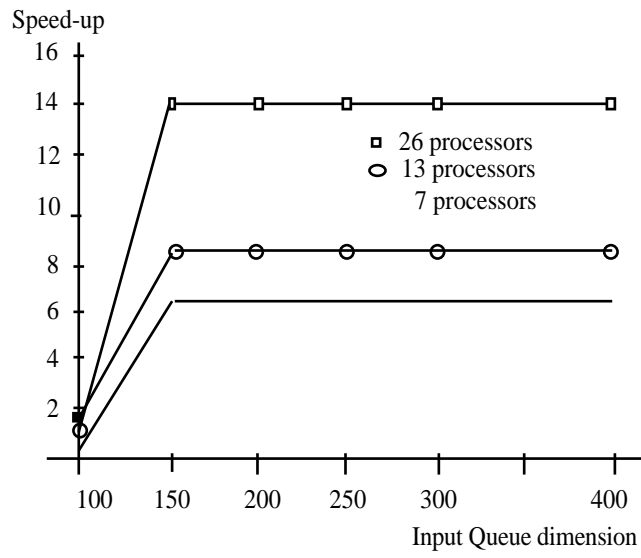
Fig. 6. Execution time with back-forward and with rollback.

Fig. 7. Speed-up of RNR method by varying processor number.