Effect of fan-out on the Performance of a Single-message cancellation scheme

Atul Prakash (Contact Author)
Gwo-baw Wu
Seema Jetli
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109-2122.
Email: aprakash@eecs.umich.edu, gwu@eecs.umich.edu
Phone: +1 313 763-1585

Abstract

For optimistic distributed simulations, in [7], a single-message cancellation scheme simulation was suggested. The scheme potentially requires fewer cancellation messages and does not require maintenance of output queues, thereby potentially also saving memory. Our performance results on a network of Suns indicate that scheme works well in low-fanout topologies. In large fan-out topologies, however, the scheme performs poorly; the scheme can actually end up sending more negative messages than the time warp protocol, many of them to destinations that would not have received negative messages if output queues were maintained. In this paper, we explore the issue whether maintaining a small amount of *hint* information about likely destinations, in place of the entire output queue, makes the scheme effective for large-fanout topologies. We report and analyze performance results on a network of Suns for time warp and three variations of the single-message cancellation scheme — no output queue, full output queue as in time warp, and hint information only. Our results indicate that hint information may be sufficient for good performance in both low-fanout and high-fanout topologies.

Index Terms: Distributed and parallel simulation, time-warp, discrete-event simulation, optimistic computations, distributed algorithms.

1 Introduction

Discrete-event simulations are frequently needed in analyzing and predicting performance of systems. However simulations, generally run sequentially, often take enormous amount of time. Parallelizing a simulation by partitioning it among multiple processors can potentially give significant speedups over sequential simulators and has been a subject of intense study for more than a decade. In this paper, we analyze performance of several optimistic parallel discrete-event simulation techniqu and attempt to address the question whether it is possible to reduce overheads in optimistic simulations in terms of communication, computation, and and memory.

Optimistic distributed simulation methods are based on the Virtual Time paradigm [4]. Each message between logical processes carries two timestamps, a virtual send time and a virtual receive time. To guarantee causality, each logical process attempts to process messages in the virtual receive time order. If a logical process receives a *straggler*, a message with lower virtual receive time than its virtual time (receive time the message it it last processed), it rollbacks and re-execute messages in the correct sequence. The process, during the rollback, may need to send *cancellation events* to other processes who may have seen erroneous messages from it.

An issue that affects performance of optimistic algorithms is the way cancellation events are send. Obviously, somehow sufficient state has to be maintained so that processes can determine which events to cancel when a straggler message arrives. In the Time Warp protocol [5], as well as its variations [9, 11], each process maintains an Output Queue, containing a copy of all messages that have been sent. Two of the fields in each message are *virtual send time* and *virtual receive time*. The virtual receive time is the virtual time at which the message is supposed to be received by the destination process. The virtual send time is the virtual time at which the message was generated and is always less than the virtual receive time. Upon a rollback to a virtual time t, a process goes through its Output Queue and sends *negative* messages corresponding to each message that has a virtual send time larger than t. Positive messages and negative messages are identical in all fields except for their sign. Each receiver maintains a queue, called *Input Queue*, containing received messages, both unprocessed and processed. If a receiver ever finds both a positive message and its corresponding negative message queued up, it *annihilates* both of them. If the positive message had already been processed, the process rolls back to the virtual receive time of the message, possibly sending additional negative messages.

An alternative protocol, SFilter[8], also based on the virtual time paradigm, uses a single cancellation message to cancel multiple messages to the same destination. The essential difference from the Time Warp protocol is that it is the receiving process that determines the messages to cancel. If the sender rolls back to a virtual time t, it sends a single negative message containing t to each potential receiver. Receivers go their Input Queues and annihilate all messages from the same sender with virtual send time ¿ t. To allow for potential reordering of messages during processing, both positive and cancellation messages also contains a monotonically increasing sequence number. Receivers annihilate positive messages from the same sender with lower sequence numbers.

Benefits of the SFilter protocol include:

1. Need for fewer negative messages, since only one negative message is required for each receiver;
2. Need for lower memory, since Output Queue is not needed to determine the contents of negative messages; and
3. Some saving in computation when sending a message because the message does not have to copied into an output queue and upon a rollback, no searching is required in an output queue (since output queue is not there).

Drawbacks of the SFilter protocol include

1. sending of unnecessary negative messages upon a rollback, since in the absence of an output queue, a negative message has to be sent to each potential recepient;
2. a slightly more complicated cancellation mechanism at the recepients since it is not apriori known how many messages will be cancelled by a negative message.

An obvious issue is whether the benefits outweigh the drawbacks. The drawbacks can be expected to become more serious as the number of potential receivers for each process grows, since upon a rollback, the SFilter protocol would require sending a negative message to every potential receiver whether it actually got an erroneous message or not. In the worst case, if nothing is known about the topology, the SFilter protocol would require sending of a negative message to every process in the system. In the results given in [8], benefits did seem to outweigh the drawbacks. However, in the example used, each LP had only small number of potential destinations, so the drawback did not appear to be serious.

To explore this issue further, in this paper we will look at three variations of SFilter, with output queue, without output queue, and without output queue but with some *hint* information that helps cut down sending of unnecessary negative messages. We will compare the schemes and our implementation of the time warp protocol on a network of Suns for topologies with increasing fanouts.

Another scheme for replacing multiple cancellation messages with a single negative message is using *message bundling* [2]. That scheme however requires Output Queues to be maintained since it transmits a count in each negative message that indicates how many messages it is supposed to cancel.

2 Variations of SFilter Used

Besides the basic version of SFilter that does not use output queues, we evaluated two other versions:

2.1 SFilter with Output Queue

If a process maintains an Output Queue, unnecessary negative messages can be eliminated in SFilter, but at the expense of more memory use and some extra computation when sending a message. The idea is to examine, upon a rollback, the Output Queue to determine which receivers have seen erroneous messages and send exactly one cancellation message to each of the receivers. The time-stamp in the cancellation message is the virtual time to which the process rolled back [1]

2.2 SFilter with hint information about destinations

In the case, instead of saving all the output messages, a process only saves some simple information for each of its receivers. The information saved is the virtual send time of the*latest* message a sender sent out to a receiver. Upon rollback to virtual time t, a process examines the hint information and sends a cancellation message to only those receivers whose latest send time is $\geq t$.

Since complete output queue is not maintained in this method, some unnecessary negative messages could still be sent. However, the method should overall use fewer cancellation messages than using no information about receivers at all. An advantage of the method is that it would usually require less memory than maintaining a complete Output Queue.

[1]It is also possible to set the time-stamp of a cancellation message sent to a receiver to be equal to the virtual send time of the first erroneous message sent to that receiver. This can help slightly with cutting down the time to find corresponding positive messages at the receiver if the input queue at the receiver is ordered by time.

The hint information does not required to be updated after a GVT computation. However, after a rollback, it may need to be updated. If a process rollbacks to time r, all time-stamp values in the hint information that are greater than r are set to r. The effect of this is similar to discarding all messages with virtual send times larger than r from from the Output Queue in Time Warp.

3 Performance

We have implemented the time-warp and SFilter programs on a network of SUNs. Several variations of SFilter have been implemented to be compared with the time-warp algorithm. The following will describe the expermient we made and the performace result.

3.1 Testbed

Logical processes are grouped in clusters such that the inter-cluster communication is minimal. Each cluster of logical processes will be allocated to one processor. There are some clustering algorithms can be found in [12][3][10][6].

3.2 The Experiment

Standard benchmarks have not yet been formulated for parallel simulations. We chose to model a ring of processes to measure the relative performance between Time-Warp and SFilter algorithms. Figure 1 shows a ring of processes with two outgoing channels for each process.

In our experiment, 64 processes are simulated. Adjacent processes are grouped together to form a cluster. The number of processes in a cluster depends on the number of processors used. For example, 16-process clusters are formed if 4 processors are used in the simulation. Upon receiving a message, each process does some internal computation which is simulated as a for loop for 20,000 times(about 3 ms). Total simulation time is about 10 minutes for the sequential run.

In our model, there is a source process injecting messages into the ring. On receiving a message, the process does some internal computation, then sends a new message to one of the immediate output neighbors through one outgoing channels. We used a uniform random number generator to ensure that each output channel was equally likely to be selected. When a predefined cutoff virtual time is reached for every process, the simulation terminates. At the end, some statistic information for each process is

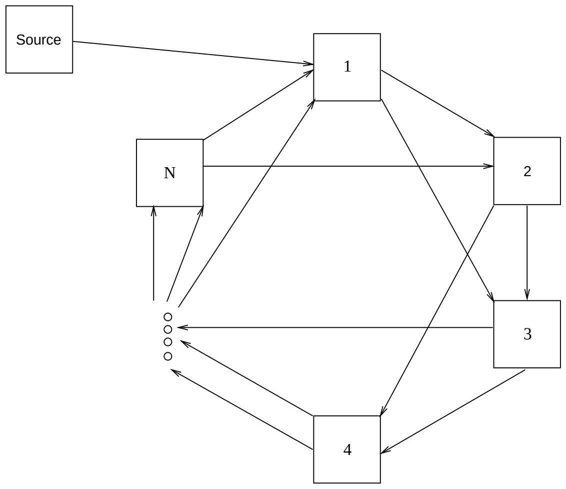

Figure 1: 2-fanout ring network. In general, in an m-fanout network, each process can send messages to the next m processes in the network

displayed for the programmers to check the correctness of the simulation. Total elapse time is also displayed for each processor. We believe that the above type of configuration is useful in testing parallel simulations, because it contains a reasonable amount of inherent parallelism, it is homogeneous and symmetric, and a good mapping from processes to processors can be found.

Currently, 32 messages are injected to the ring by the source process. These 32 messages carry time-stamps 8, 16, 24, ..., and 256. Messages will be delivered when the time-stamps of the messages are smaller than or equal to the local virtual clock of the destination. The time of servicing a message is based on an exponential distribution with rate 0.2. Internal computation is around 2.8ms for every event simulated. After servicing a message m, the process will schedule the delivery time of a new message at one of its neighbors based on its local virtual clock and the service time of m.

We varied the fanout of processes in the ring to see how each algorithm will perform in each case.

3.3 Parameters Measured

Speedup: The speedup is the time required to run a simulation on a number of processors compared with the time required to run the same simulation on a single processor. The result is the curve usually goes up and then drops.

Number of cancellation messages processed: The higher number of negative messages to be processed in each process involves higher overhead to traverse the queue and to cancel erroneous messages sent by other processes.

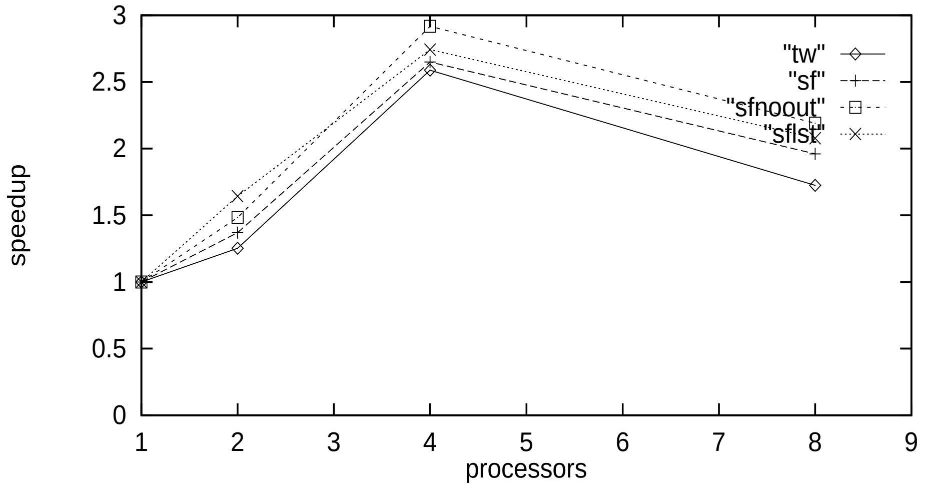

Figure 2: Speedup vs. number of processors for a 2-fanout ring network

Number of messages cancelled: We can measure the effectiveness of the cancellation messages by looking at the number of cancellation messages processed and the number of messages were cancelled.

3.4 Results

In the following graphs, *tw* stands for time-warp, *sf* stands for SFilter with output queue, *sfnoout* stands for SFilter without output queue, and *sflst* stands for SFilter with hint information of send time of last message sent to each destination.

Figures 2 to 4 show the performance of each algorithm in different fanout topologies. We stopped further simulation if the speedup curve for an algorithm went below 1.

The results show the following:

- SFilter without output queue performed the best in low fanout case, but performed poorly in medium and high fanout with number of processor greater than 2.
- Time-warp performed reasonably well and was stable but was usually outperformed by SFilter algorithms.
- SFilter with output queue performed well in most of the cases.
- SFilter with hint information performed almost as well as the best scheme in all situations.

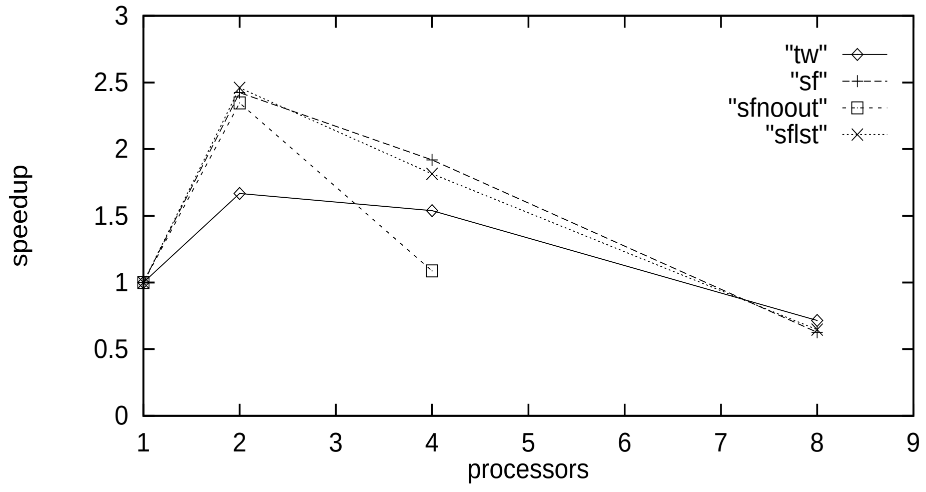

Figure 3: Speedup vs. number of processors for a 8-fanout ring network

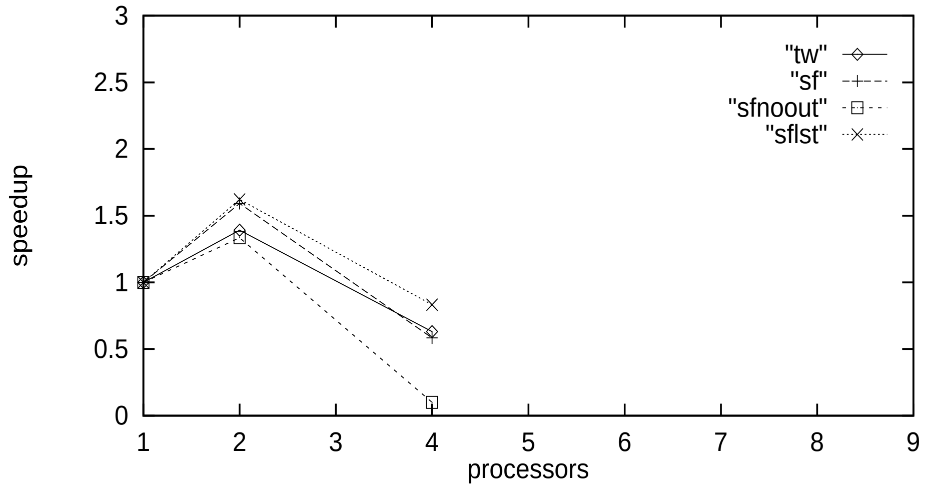

Figure 4: Speedup vs. number of processors for a 32-fanout ring network

	algorithm	2 processor	4 processor	8 processor
fanout 2	sf	10228	14027	47134
	sfnoout	14680	21870	61008
	sflst	6799	14533	37502
	tw	16117	19885	66502
fanout 8	sf	6723	37994	183061
	sfnoout	48424	292360	—
	sflst	6844	41193	177113
	tw	8737	43213	151300
fanout 32	sf	6625	144668	—
	sfnoout	250781	—	—
	sflst	7003	134940	—
	tw	12495	121132	—

Table 1: Cancellation messages processed

	algorithm	2 processor	4 processor	8 processor
fanout 2	sf	13200	17941	69381
	sfnoout	11099	14848	52524
	sflst	8145	15175	52270
fanout 8	sf	6949	40782	206675
	sfnoout	8054	64176	—
	sflst	6772	38683	156427
fanout 32	sf	6663	148515	—
	sfnoout	7115	—	—
	sflst	6808	111529	—

Table 2: total messages cancelled

To further analyze the results, we also measure the number of cancellation messages processed and number of messages cancelled by cancellation messages in the system. Table 1 shows the total number of cancellation messages received and processed in the simulation for each algorithm in different fanout topologies. Table 2 shows the total number of messages that got cancelled in input event queue in the simulation for each case except for time-warp, in which the number of messages cancelled will be the same as the number of cancellation messages processed.

Several facts can be summarized as follows:

- In SFilter with output queue, for every cancellation message, approximately 1 to 1.3 messages get cancelled.
- In SFilter without output queue, the number of cancellation messages is always higher than the number of messages to be cancelled. The situation becomes worse when fanout increases.

- In SFilter with hint information, each cancellation message cancels 1 to 1.4 messages for low fanout situations. For medium and high fanout situation, each cancellation message cancels approximate 0.8 to 1 messages.

The results show that SFilter with output queue and SFilter with hint information have low overhead in terms of cancellation messages.

4 Conclusion

In the paper, we evaluated three variations of SFilter algorithms and time-warp on a local network of workstations. SFilter without output queue eliminates the cost of maintaining and traversing the output queue, but the benefit only shows up in the low fanout topology which low number of cancellation messages is produced for each rollback. In high fanout topology, SFilter with hint information corrects the problem of the need to send multiple cancellation messages by keeping the information of send time of last message sent to each potential output neighbor. SFilter performs as well as the best scheme in each fanout topology.

In near future, we are planning to look at other simulations, such as the colliding pucks simulation [1], that have much higher fanouts. We want to see if for extremely high fanouts, maintaining summary hint information continues to be sufficient. It is concievable that for very high fanouts, output queues become necessary and Time Warp and SFilter with Output Queues may outperform other algorithms.

We are also looking to evaluate various strategies on other machine architectures such as the KSR which may present different behavior for the various algorithms discussed in this paper.

References

[1] B. Beckman et al. Distributed simulation and time warp part 1: design of colliding pucks. In *Proc. of the Society for Computer Simulation Western Multiconference on Distributed Simulation*, 1989.

[2] J. Butler and V. Wallentine. Message bundling in time warp. In *Simulation Work and Progress, 1991 Western Simulation Multiconference*, January 1991.

[3] K. Efe. Heuristic models of task assignment scheduling in distributed systems. *Computer*, 15(6):50–56, June 1982.

[4] D.R. Jefferson. Virtual time. *ACM Trans. on Programming Languages and Systems*, 7(3):404–425, July 1985.

[5] D.R. Jefferson et al. Distributed simulation and the time warp operating system. *ACM Operating System Review*, 1987.

[6] V.M. Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Trans.*, C37:1384–1397, May 1988.

[7] A. Prakash and R. Subramanian. Conditional knowledge approach to optimistic distributed simulations. Technical Report CSE-TR-86-91, Department of EECS, U. of Michigan, Ann Arbor, 1991.

[8] A. Prakash and R. Subramanian. An efficient optimistic distributed simulation scheme based on conditional knowledge. In *6th Workshop on Parallel and Distribued Simulation (PADS 92), Proc. of the 1992 SCS Western Simulation Multiconference on Parallel and Discrete Event Simulation*, pages 85–94, Newport Beach, California, January 1992.

[9] P. Reiher, S. Bellenot, and D. Jefferson. Temporal decomposition of simulations under the time warp operating system. In *Proc. of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 47–54, January 1991.

[10] A.K. Sarje and G. Sagar. Heuristic model for task allocation in distributed computer systems. *IEE Proceedings-E*, 138(5):313–317, Sept 1991.

[11] L.M. Sokol, D.P. Briscoe, and A.P. Wieland. MTW: A strategy for scheduling discrete simulation events for concurrent execution. In *Proc. of the SCS Multiconference on Distributed Simulation*, pages 34–42, July 1988.

[12] H.S. Stone. Multipleprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Software Eng.*, SE-3(1):97–106, Jan. 1977.