Performance of a Riskfree Time Warp Operating System

Steven Bellenot
Mathematics Department
Florida State University
Tallahassee, FL 32306
bellenot@math.fsu.edu

ABSTRACT

Optimistic methods of synchronizing parallel discrete event simulations can be risky by sending (positive) messages (events) before they have been committed. Risky methods often use anti-messages (negative messages) to cancel incorrectly sent positive messages. Riskfree methods are more conservative, they do not send messages until they are known to be correct. The Time Warp Operating System (TWOS) uses anti-messages. Riskfree TWOS is implemented and tested on the standard TWOS benchmarks. Performance of the riskfree TWOS is dependent on the amount of lookahead in the simulation. Good lookahead was required for even reasonable performance. Tracker, a simulation of the riskfree simulation, is used to give idealized best case riskfree performance. BeRisky is an example simulation which has a speedup of n for Time Warp, but only a speedup of 2 for riskfree methods.

Introduction

We refer the reader to [F90] for an introduction to parallel discrete event simulation, and to where undefined terms can be found. The Time Warp Operating System (TWOS) is the operating system implementation of Time Warp [JBW87] done at the Jet Propulsion Laboratory (JPL). Time Warp is a method of optimistically synchronizing parallel discrete event simulations.

An optimistic mechanism can also be *risky* in the sense of [R88]. Execution of event can generate events (messages) for other objects. Risky simulation methods immediately send these messages. A risky method must provide a mechanism, often anti-messages or negative messages, to unsend an incorrect (positive) message. A optimistic method can be *riskfree* just by not sending messages until it is certain they are correct. TWOS is risky and SPEEDES ([S91] and [S92]) is a riskfree optimistic method. (The name SPEEDES is changed to breathing time buckets in [S92], but we will use the old name in this paper.) Steinman in [S92] adds an aggressive cancellation version of Time Warp to the riskfree SPEEDES and compares their performance. Here we do a similar exercise with TWOS. We implement a riskfree version of TWOS and compare it to the regular risky TWOS. Riskfree TWOS is not synchronous like the riskfree method in SPEEDES.

Generally, the performance of the riskfree TWOS was poor. We show that some simulations like BeRisky below will not run well with any riskfree method. We find that good riskfree performance for the JPL benchmarks requires good lookahead in the simulation. The *lookahead* of an object is the $min(r(m) - s(m))$ as m ranges over the messages it schedules. ($r(m)$ = virtual receive time, and $s(m)$ = virtual send time.) A simulation has good lookahead if the lookahead value is large compared to the usual event spacing. Riskfree TWOS ran best when the list of uncommitted events was long (in terms of CPU time) and new events were scheduled at times which put them near the end of these uncommitted events. (Or in the language below, successive event horizons must contain lots of simulation work.) This particular implementation of a riskfree TWOS is slowed by using the usual TWOS code to compute GVT. To factor out this limitation, Tracker, a simulation of riskfree simulations was created.

The Event Horizon

The event horizon is a term from SPEEDES, but it has meaning in a sequential simulation. At a given simulation time T, the *event horizon* is the virtual time of the earliest event generated by an executing an event at virtual time T or greater. The main loop of SPEEDES executes object until the next event horizon is known. Once all nodes know the new event horizon, all output messages with send time less than the new event horizon are sent. Due to the nature of its sending mechanism, all nodes will know when all messages have been safely received. The object execution resumes to determine the next event horizon. In SPEEDES, the sequence of event horizons is repeatable and can be determined from a sequential run. The calculating the event horizon has a lot in common with calculating global virtual time (GVT) in TWOS. But the next event horizon calculation requires the all events with virtual time less than the new event horizon be completed, where as a GVT calculation can be done at anytime.

Tracker

In light of the limitations of our riskfree TWOS we designed and implemented Tracker. Tracker simulates a no overhead SPEEDES simulation. Tracker has one delay parameter which makes it also a simulation of our riskfree TWOS. Tracker requires two steams of input data which we collected in to separate runs of the sequential simulator. One run just collected a list of all messages (events) send during the simulation. The other run collected event execution times. Tracker assumes

there is no other run time overhead. Tracker interrupts events in progress if an earlier event becomes available. The list of Tracker assumptions is given below.

Tracker Assumptions

1. The event execution real time is the same for early (wrong) executions as it is for the correct (sequential) event execution time.
2. Event horizons are calculated "exactly" at the instant they can be known. The delay to spread this event horizon to the nodes is the Tracker parameter commit delay.
3. There is no synchronization overhead and messages have zero communication delay time.

Effect of the commit delay parameter in Tracker

Between the real time the event horizon is known and the time all the nodes learn the new event horizon could be called the twilight zone. This time difference is called the *commit delay.* While in the twilight zone, each node is executing events which could be rolled back. If there is not enough events available to execute, this twilight zone of time will be idle CPU time. That is if the commit delay parameter is large enough, then Tracker (and a riskfree simulation) will run out of simulation work to do while waiting for the new event horizon. Idle CPU time, if long enough, would rob the simulation of any speed up. The estimated commit delay for the current riskfree TWOS implementation is very high, around $30-60$ milliseconds for $10-70$ nodes (see Ping below). This long commit delay can lead to idle CPU time.

The BeRisky simulation

The BeRisky simulation consists of n objects named 0 through $n-1$. Each object only sends messages to itself and every event execution takes the same amount of real CPU time. At time $i*n+j$, object j sends a messages to itself at time $i*n+j+1$. And at time $i*n+j+1$ object j sends a message to itself at time $(i+1)*n+j$. Time Warp (and conservative methods) have no problem (in theory) of obtaining the full n-fold parallelism of BeRisky. However, at each virtual time T, the next event horizon is $T+1$. Hence a riskfree simulation could at most obtain only a two fold parallelism. As a model this simulation is really the same as n objects which at time T just send a message to itself at a time $T+L$ in the future. Here all the objects are synchronized to the virtual time axis and in theory the riskfree method could obtain the n-fold parallelism of the new simulation. (See also Pucks below.)

Riskfree TWOS implementation

The two main changes to TWOS needed to make it riskfree were to eliminate anti-messages and changes to GVT. The original code modifications was done by a student Pi-chiang Chang. Actually anti-messages were kept and positive messages were removed from the code.

The changes to GVT are less clear cut, and we tried several approaches. A modified request scheme was finally implemented. Since in TWOS a GVT calculation is always started by node zero. Node zero became the only node which would start a GVT on demand. When a node, say number $i>0$, wanted to request a GVT, it would instead pass its request along with its current estimate of the next event horizon to the node numbered $(i+1)/2-1$. This lower numbered node would incorporate the received upper bound on the next event horizon into its own current estimate. If the received estimate was lower than the receiving node's estimate, then the receiving node might discover it was executing events beyond its new idea of the next event horizon and in turn request a GVT.

Benchmark Performances

All measurements were made on JPL's BBN Butterfly GP-1000. TWOS version 2.7 was the baseline. The three standard JPL benchmarks of Pucks, STB88 and Warpnet were measured as well as two artificial simulations Ping and Bank. Summary statistics and references for these simulations are in [B92]. The results for STB88 are omitted, they were between those of Pucks and Warpnet.

Ping Benchmark

There are two objects in Ping, and the simulation has one message which bounces back and forth. Figure 1 shows the run times of Ping when run by TWOS, when run by riskfree TWOS and a logarithm curve which fits this second plot. Note that there is no parallelism available in Ping, it is a sequential simulation. For SPEEDES and riskfree TWOS, every message is a new event horizon. This Ping sent 2500 messages and so there were 2500 event horizon calculations in the riskfree TWOS. TWOS on the other hand does almost no GVT calculations since Ping runs so fast. Ping is used to estimate the GVT calculation time (roughly $14ln(x)$ milliseconds, where x is the number of nodes) and hence the event horizon commit delay time for use in Tracker.

Pucks Benchmark

Figure 2 shows the comparison in run times of Pucks when run by TWOS and when run by riskfree TWOS. The poor performance of the riskfree Pucks was expected. Figure 3 shows Tracker run times with Pucks. (The length of the Pucks benchmark was shorted for these figures, usual Pucks is ten times longer than the Pucks in Figure 2 and forty times longer than the Pucks in Figure 3.) Even the case with no commit delay, the idealized no overhead run times are twice as long as an actual TWOS run times. With a commit delay of 10 milliseconds, speed up drops by almost one fifth. And with the actual riskfree TWOS estimated delays, Tracker Pucks runs slower as the number of nodes increase.

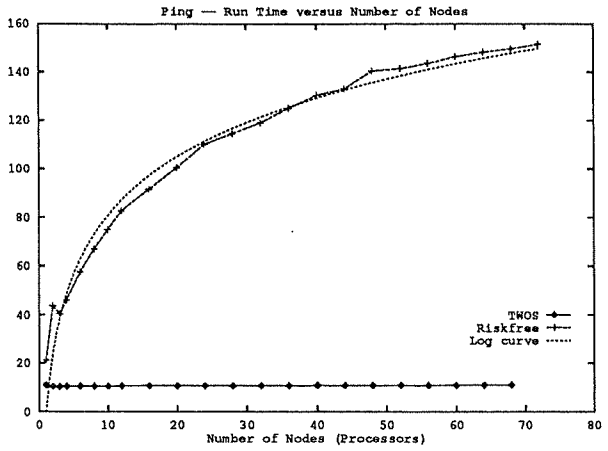

Figure 1. Ping run times.

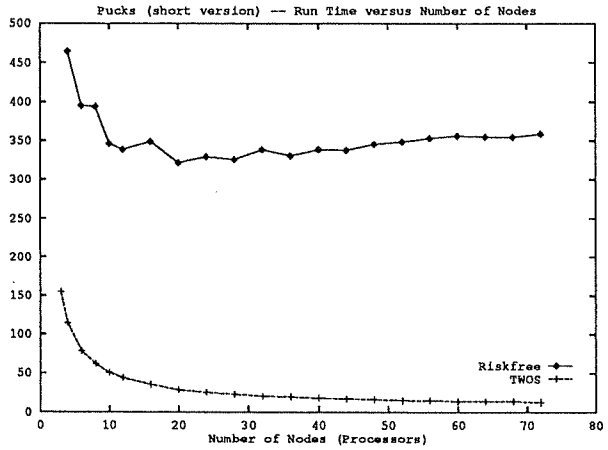

Figure 2. Pucks run times.

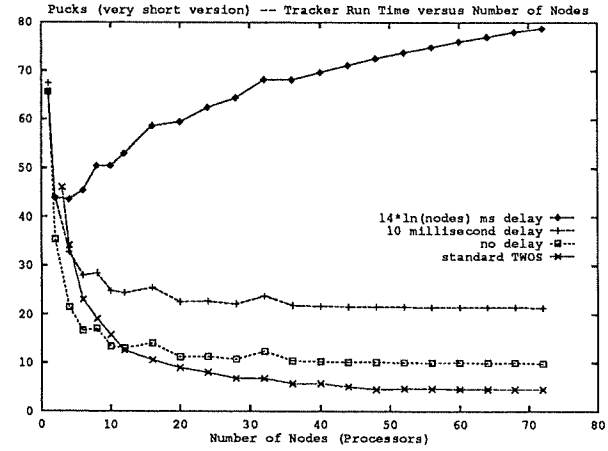

Figure 3. Tracker Pucks run times.

What does Pucks do that ruins the riskfree performance? First Pucks has no lookahead. When two puck objects collide, both pucks gain new velocity vectors. Both pucks send information messages containing their new velocities to the sector(s) they are currently in. These information messages are almost for now, they are guaranteed to be the next events in the simulation. The sector(s) respond to these information messages by sending informational messages to its neighboring sectors, cushions and the pucks in its sector. These information messages are also almost for now, they are also guaranteed to be the next events in the simulation. This is a disaster for riskfree performance, there are two event horizons after every collision with essentially no simulation work. Thus a riskfree Pucks essentially sequentializes the pucks collisions losing most of the parallelism.

We believe that Pucks with lazy cancellation TWOS beats the critical path because of lazy cancellation's ability to obtain temporal parallelism (See [RBJ91]). Pucks is known to run $30-60\%$ slower with TWOS with aggressive cancellation [B89] for similar reasons. Although Pucks is a disaster for SPEEDES, apparently other proximity detection models can obtain a high degree of parallelism [S92]. (But note the Time Warp which Steinman uses to compares with SPEEDES uses the less than optimal aggressive cancellation.)

Warpnet Benchmark

We were surprised by the poor riskfree Warpnet performance. Figure 4 shows the run time for Warpnet comparing riskfree TWOS with TWOS. Events in Warpnet only happen at integer virtual times and there is a good deal of parallelism available at each virtual time. However, the lookahead in Warpnet is poor. At most virtual times T, there is an event scheduled for virtual time $T+1$. The poor lookahead and the lengthy GVT calculation times for riskfree TWOS lost this potential parallelism.

Figure 5 shows the various Tracker speedups for Warpnet, with various commit delay values compared to the real TWOS and riskfree TWOS values. Note that even the idealized no overhead no delay Tracker speedup curve flattens before the actual TWOS speedup curve peaks. Indeed at each increased level of commit delay in the Tracker curve, has the Warpnet speedup curve flatten earlier. When the commit delay is $14ln(nodes)$ the speedup becomes horizonal at about 36 nodes, essentially the same shape as the real riskfree TWOS speedup curve. These Tracker speedup curves were what we expected Warpnet would perform under riskfree TWOS. The flattening of these Tracker speedup curves is due to increased idle time, the simulation runs out of work to do and most wait for the next event horizon for more work.

Bank Benchmark

The Bank benchmark was introducted in [B92]. Each of the 16 bank objects reacts to a message by gen-

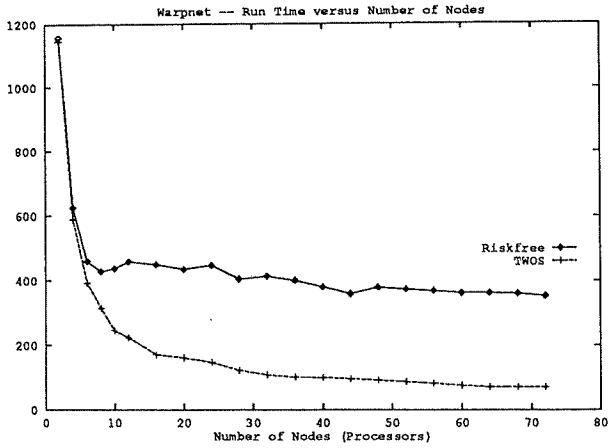

Figure 4. Warpnet run times.

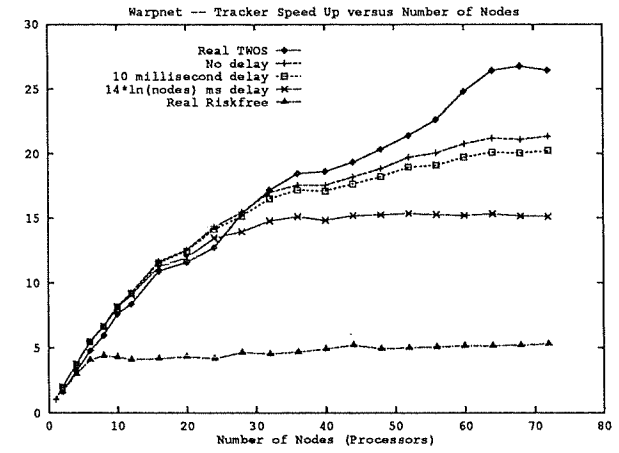

Figure 5. Tracker Warpnet speedups.

erating a message with a random receiver and random exponential interarrivial time with mean 10 (but see below). We modified Bank so to discover if the riskfree TWOS implementation could ever obtain parallelism. Here is how we varied lookahead. The usual bank message sent at time T arrives at $T + D$ where D is an exponential random variable with mean 10. A new bank parameter of lookahead L changed this arrival time to $T+L+D$ where D is now an exponential random variable with mean $10 - L$. This changes the simulation slightly, but the delta (interarrivial) time between send and receive times always has mean 10. Figure 6 shows the best speedups we obtained for Bank. The message density was 25 and the execution delay is 40 making the average event execution time around 41 millieseconds. The best riskfree Bank speedup was almost 9.5, the same simulation using TWOS got a speedup over 13. Lookahead had the biggest effect on banks riskfree performance.

Conclusions

Time Warp with lazy cancellation can obtain good

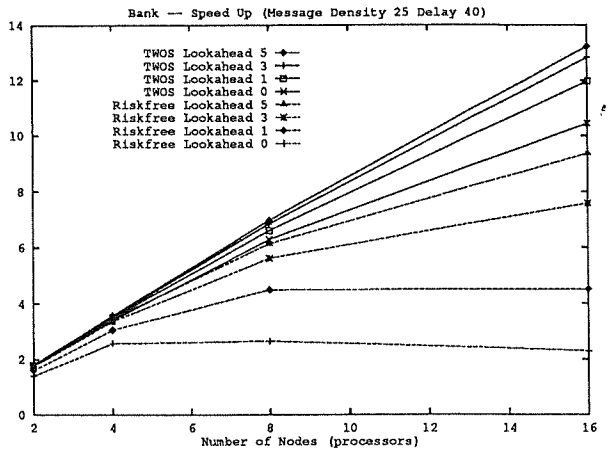

Figure 6. Best Bank speedups.

speedups with simulations with poor lookahead. Time Warp can obtain the temporal parallelism available in BeRisky and Pucks. Good lookahead is needed to obtain good performance in riskfree simulation methods. The TWOS benchmark suite, at least as constructed, would likely have poor performance in any conservative or riskfree simulation method.

References

[B89] S. Bellenot, *Why is Pucks lazy?*, JPL interoffice memo SFB:363-89-005, (1989).

[B92] S. Bellenot, *State Skipping Performance with the Time Warp Operating System*, 6th Workshop on Parallel and Distributed Simulation (PADS92), SCS simulation series 24 (1992), 53-61.

[F90] R. Fujimoto, *Parallel Discrete Event Simulation*, Communications of the ACM, 33 no 10 (1990), 30-53.

[JBW87] D. Jefferson, B. Beckman, F. Wieland, L. Blume, M Di Loreto, P. Hontalas, P. Laroche, K. Sturdevant, J. Tupman, V. Warren, J Wedel, H. Younger, and S. Bellenot, *Distributed Simulation and the Time Warp Operating System*, Proc. 12th SIGOPS - Symposium of Operating Systems Principles, 1987, 77-93.

[RFB90] P. Reiher, R. Fujimoto, S. Bellenot, and D, Jefferson, *Cancellation strategies in optimistic execution systems*, Distributed simulation 22,1 (1990) 112-121.

[R88] P. Reynolds, *A Spectrum of Options for Parallel Simulation Protocols*, Proc of ACM Winter Simulation Conference, (1988), 325- 332.

[S91] J. Steinman, *SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation*, Advances in Parallel and Distributed Simulation, SCS simulation series 23, (1991) 95-103.

[S92] J. Steinman, *SPEEDES: A Unified Approach to Parallel Simulation*, 6th Workshop on Parallel and Distributed Simulation (PADS92), SCS simulation series 24 (1992), 75-84.