JET PROPULSION LABORATORY **INTEROFFICE MEMORANDUM**

SFB: 363-89-002

20 July 1989

TO: Distribution

FROM: Steven Bellenot

SUBJECT: Performance Results: STB88 with different cancellation policies

INTRODUCTION:

The STB88 Benchmark was used to compare the three following cancellation policies:

1. Jump Forward - an optimization of lazy cancellation.
2. Lazy - lazy cancellation without jump forward.
3. Aggressive - aggressive cancellation.

A special version of TWOS near version 2.1 was used to measure these run times. Configuration files for STB88 were those used to run the 2.0 benchmarks, those being the only ones available. Runs were made on 44, 40, 28, 24, 20, 16, 12, 8 and 4 nodes. Other nodes > 28 were not run because those 2.0 config files were not easily available. Generally, this version is fair to all three cancellation policies. At the end of this paper we will consider the remaining unfairness.

CONCLUSIONS:

1. The jump forward optimization is not!
2. Lazy is faster than aggressive on 16 or more nodes, but not by much.
3. There are parameters where lazy is solid but aggressive wobbles.

JUMP FORWARD:

What the jump forward optimization to lazy cancellation does is to compare the new state with the old state (if there is one) at the end of each event. If the two states are equal, then this object can jump ahead to the next unprocessed message in the future. Jump forward was added to TWOS when there were queries. Queries never change the state and hence jump forward always works for "straggler" queries. Besides the cost of comparing states, jump forward requires unmarking of input messages when processed and careful remarking of messages on rollback. Jump forward is the more complex in terms of code than lazy cancellation which in turn is more complex than aggressive. Jump forward is currently in TWOS 2.1 and has been in TWOS for over a year or so.

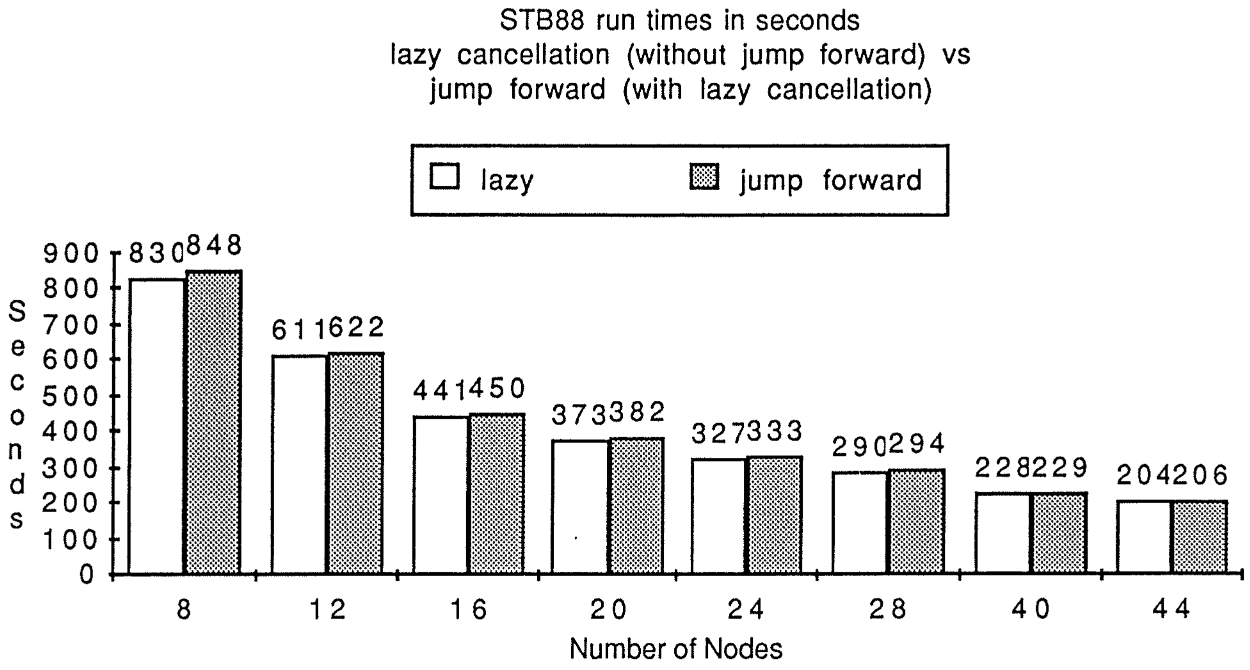

Figure 1.

The run times are graphed in Figure 1. Each data point is an average of three runs so differences of one second are not significant. The 4 node runs were not graphed but jump forward loses here too, 1659 seconds to 1632 seconds for lazy. Lazy is always faster than jump forward, but jump forward gets closer to the run time of lazy as the number of nodes increases. Perhaps this due to a greater amount of idle time per node as the number of nodes increase. That is, jump forward makes good use of time which would otherwise be idle in the lazy cancellation case. However, it seems that jump forward is one of those "short cuts" which actually take longer. As far as STB88 is concerned, jump forward should be removed from TWOS.

LAZY vs AGGRESSIVE:

As expected, the run times STB88 under lazy or aggressive are roughly the same with lazy the "winner" by a few percent. Figure 2 graphs the run times, each data point is the average of three runs so differences of one second are not significant. The 4 node runs were not graphed, but aggressive was faster 1600 seconds to 1632 seconds for lazy. There is a slight tend for lazy to improve over aggressive as the number of nodes increases, but it is the early to tell if this is significant or not. A certain amount of tuning of the queueing parameters was done to optimize the aggressive run times on 40 and 44 nodes. STB88 would vote to keep lazy cancellation at least as an option in TWOS.

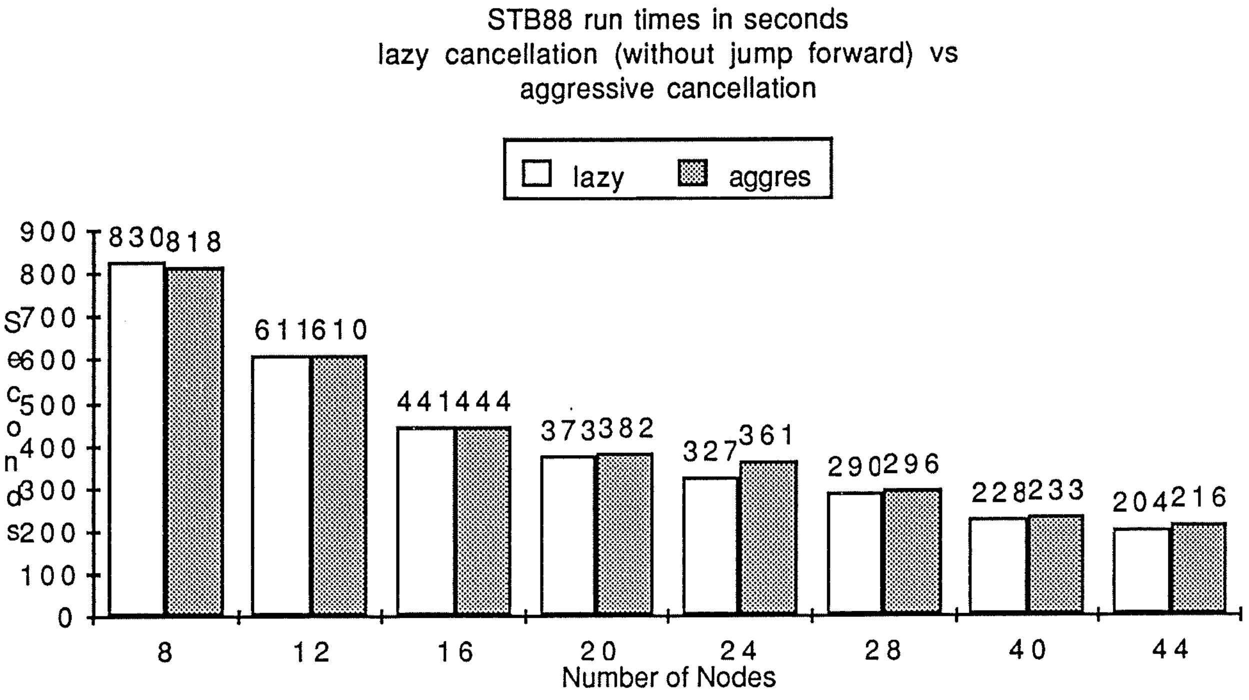

Figure 2.

AGGRESSIVE WOBBLES:

The most interesting feature of the lazy vs aggressive race happens in the 40 and 44 node cases. This feature is the existence of queueing parameters where the run time of aggressive could vary by as much as 31% while the run time of lazy seems rock solid. Aggressive ran 9% to 43% slower than lazy in the 44 node case. This variation of run times is shown in Figure 3. The queueing parameter which seem to have the most effect was reducing max_acks from 4 to 2. Figure 4 shows run times for a "cutoff" STB88 with different queueing parameters. Max_acks is the maximum number of positive messages in transit per node. The version of TWOS used does not limit negative messages by max_acks, however TWOS 2.1 may limit the number of both positive or negative messages in transit by max_acks. The other parameter "mna" is the maximum length of the receive queue. Each read would acquire at most mna messages off the FIFO "dual queue". The number of messages waiting in the dual queue could be as large as 64 * num_of_nodes but only max_acks * num_of_nodes could be positive. The runs in Figures 1 and 2 for nodes < 30 had ma = 4, mna = 10 and for nodes > 30 had ma = 2, mna = 30.

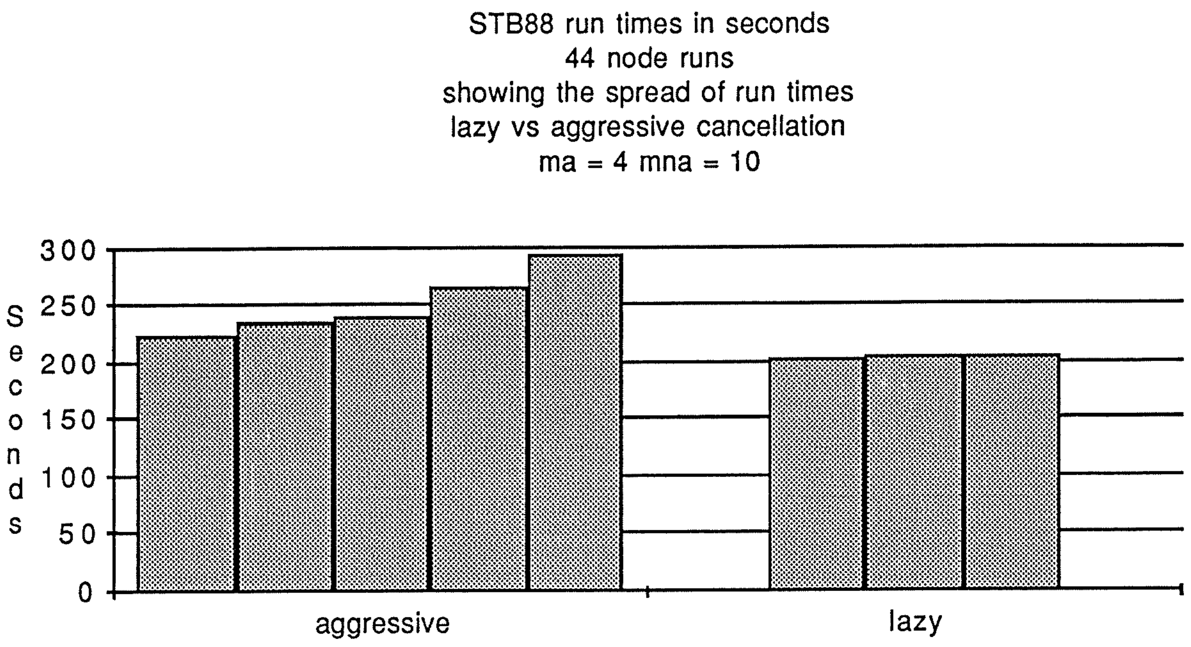

Figure 3.

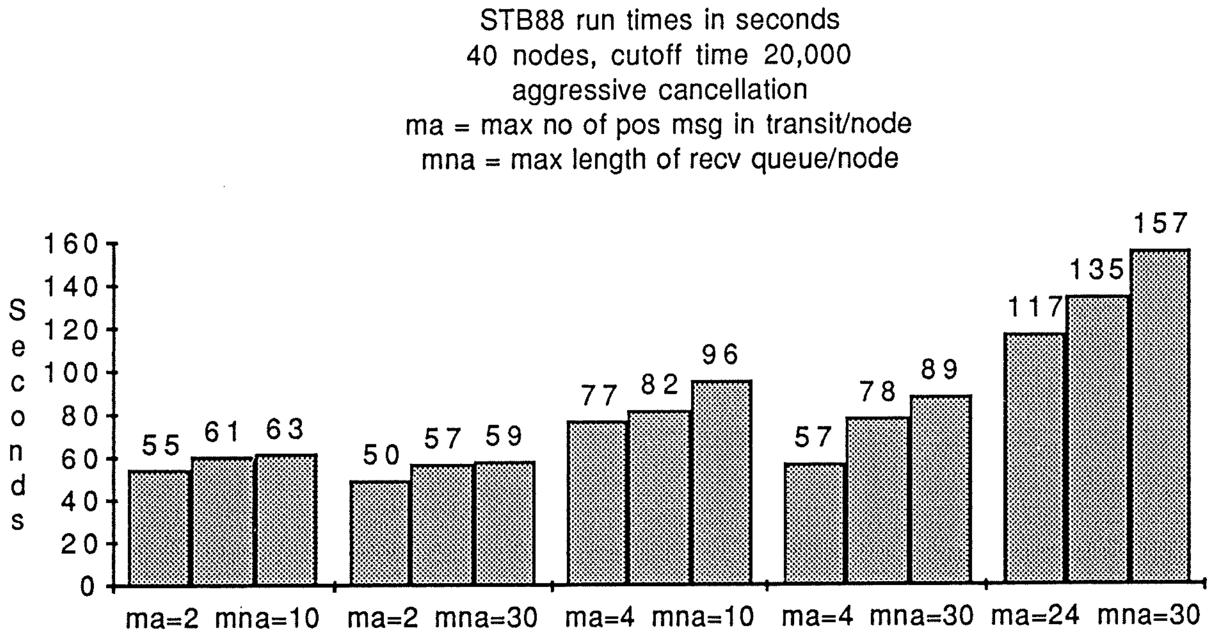

Figure 4.

UNFAIRNESS:

The version of TWOS (in /usr/local/src/test_2.1) is the fastest version on STB88 on 16 nodes. Code (basically a Chrysalis call) which was required to prevent Fujimoto's Lazy Buster from having run-away-behavior was commented out. This function call added to the run time of all three simulations (over 10 seconds on lazy on 16 nodes). TWOS 2.1 (in /usr/local/src/bbn_2.1) stops the Lazy Buster in a different way. On the other hand, the non-limitation of max_acks for anti-messages in the test version favors aggressive over the way it is done in TWOS 2.1. There are other ways which the tested version differs from the official version, but it is not clear that these differences favor lazy or aggressive. The receive queue in test_2.1 is ordered differently than in TWOS 2.1. Also the time which it last checks for an arriving message before running an object is different in the two versions. Lazy cancellation requires the saving of two copies of each message, perhaps aggressive cancellation would have two copies for debugging purposes. Making aggressive a run-time switch introduces 8 tests for "if (aggressive)" or "if (! aggressive)" into the code. Commenting these run-time tests out (or in?) made no noticeable difference in the run-time of STB88 on 16 nodes (under lazy). Since jump forward and lazy cancellation have been a part of TWOS for so long there might be other code in which the run time of aggressive is slowed down unnecessarily. However, no such code is currently known.

10.0 Distribution

Time Warp and CTLS Distribution List
Steve Bellenot
Jay Braun
Mike deGyurky
Paul Firnett
Richard Fujimoto
Hugh Henry
Milt Lavin
Al Loomis
Bob Miller
Martin Orton
Joe Provenzano
Al Silliman
John Spagnuolo
John Weidner