Tools for Measuring the Performance and Diagnosing the Behavior of Distributive Simulations using Time Warp

Steven Bellenot
Departments of Mathematics
and Computer Science,
The Florida State University,
Tallahassee, FL 32306

Michael DiLoreto
Jet Propulsion Laboratory,
California Institute of Technology,
4800 Oak Grove Drive,
Pasadena, CA 91109

Abstract

Like any new design, Time Warp (a distributive simulation operating system) needs tuning in order to obtain maximum performance. While we do not claim to have tuned Time Warp to this peak, but we offer a collection of tools and observations we have found useful in improving Time Warp's performance. We caution the reader that we do not claim to present the lastest in graphics nor the last word in performance measuring tools. We modestly are documenting our experience for the benifit of others.

Time Warp has constanting proved to be a source of counter-intuitative results. Over a year ago, we removed about ten percent of the messages in an application thinking that this will make it go faster, only to find that it completely broke the system. Indeed, we are even still able to be completely wrong on the consequence of a change to Time Warp.

Our tools are either graphical or statistical. In both cases we take a log file the system made of everything which happens. We can do this for short simulations (around 20 seconds on 32 nodes). With this complete log file we can find several tibbets. The graphic tools give us a idea of what the simulation was doing. Time Warp allows the simulation to go down wrong paths. We can get a idea of how many and how far these incorrect paths are executed. Offen the difference between a good run and a bad run was not noticable on the graph. Basically it just took longer, this lead us into the study of the delay times in the queues.

The Time Warp Environment:

For this paper, *Time Warp* refers to a special purpose operating system for running discrete event simulations on multi-processors built and maintained by the Jet Propulsion Laboratory for the Army Model improvement program (see [Jefferson 87]). To run on top of Time Warp, a simulation must be broken into objects which schedule events for each other via messages. We will call such a collection of objects an *application*. The Application layer (in theory) is transparent to the number or kinds of computers it runs on. Currently, the application objects are statically assigned to nodes at run time, and hence knowledge of the application is used to load balance the nodes. The simulation time, that is the time that simulation events are scheduled is called *virtual time* in Time Warp.

By the Time Warp Layer, we mean the layer of abstraction between the objects on the application level and the lower machine dependent level. In practice, this lower level can be viewed as message passing kernal which logically connects any two nodes. Time Warp has been ported to a number of machines, a network of Suns and a Butterfly, for examples, but results of this paper are based on Time Warp running atop the Mercury message passing kernal on one of the Jet Propulsion Laboratory's Mark III hypercube with 32 nodes.

For the purpose of this paper, we may view the application has senting *positive* messages, which may or may not be correct. Time Warp will *rollback* an object which has gone down an incorrect simulation path and Time Warp will send *negative* messages (often called *anti-messages*) to cancel the effects of the incorrect postive messages. Time Warp also has system messages of which the most important for our purposes are those used to compute *global virtual time* (also denoted by GVT).

We will talk about two applications. The important application is ctls87, which is called STB87 in [Weiland 88]. Ctls87 is a miltary like simulation with 3 phases and a reasonable run time of 20 seconds. The other application is an artifical one called slooow. Slooow is a fully connected model designed to give wrong answers when out of synchrony. Slooow stresses the message subsystem as well as violate everything we know about what makes a good Time Warp object. It has large fan in and large fan out as well as a very

high communication to computation ratio. Moreover, slooow is time driven and (at least in theory) completely parallel.

Graphic Tools

Since there are two kinds of time in a simulation, real time and virtual time, it seems natural to plot graphs with the two times as the different axes. The virtual time of an event is easy to determine, however to obtain the absolute real time of events several nodes required some care (see synchronizing the clocks below). The first graphic was of execution (real) times of an objects vs the virtual time at which they were running. The real execution time of an object was graphed as an horizontal interval at height given by the virtual time of the execution.

The second graphic was for messages. Each message was represented as a line from (real sent time, virtual sent time) to (real receive time, virtual receive time). A Silicon Graphic's Iris Workstation was the target machine for the graphics. We could use a wide range of colors for the different objects, or just a couple differing messages and anti-messages or good positve messages from those which were cancelled.

The log file must be create while running, it is stored in RAM and put into a single file after the execution of the application is over. These files are quite large, a typical simulation which runs about 20 seconds could produce about 70,000 messages and 180,000 execution pieces. Each execution log entry requres at least one object name, one virtual time and two real times, and message log entries required twice the room. The time needed to read the output file was long. A program that coded this information into binary format was used to decrease the set-up time of the graphics programs.

It is difficult to overrate the impact of graphics. Our pictures showed objects running down incorrect paths, ... Effects one has dicussed with great difficulting using words, become look at this....

Mplot and fplot

Time intervals:

One of the results of the graphics was the interest in delta time intervals as opposed to total time. There are operations in

Time Warp whose total time investment was small. But these same operations required large time investments each time they were done. The graphics plots were showing large time periods where nothing was being done.

Interestingly enough, the largest problem was a race condition in the Mercury program. The problem was known to those people which were maintaining Mercury. They were using spin locks to test the bidirectional channels for which way it was now. The code now spins at different speeds on different nodes.

After the Mercury problem was fixed we tested it by trying to find all large delta times in Time Warp. Two time periods were on the order of 50 - 100 milliseconds, garbage collection and a printf. (Communication with the outside world from a hypercube is much slower than innernode communication.) Garbage collection is done more or less at the same time by all the nodes.

Somehow getting the clocks in synch and trying to message the performance on Mercury got into the act. We designed hc (hypercircle) to show the channel use between nodes and the amount of conflict for the use of these channels. We pumped the messaged times and it looked like there was alot of conflict.

Histograms:
    We like it so much we did ... and ... and even ...
    hc
Other Tools
The GVT collection
The zip forward
The Hardware Race
The speed of anti-messages

References

[Jefferson 87] Jefferson, David, *et al.*, "Distributive Simulation and the Time Warp Operating System." *ACM Proceedings of the Symposium on Operating System Principles*, (November 1987),