

# Dynamic Load Balancing in Parallel Simulation Using Time Warp Mechanism

Ming-Ru Jiang    Shiuh-Pyng Shieh    Chang-Long Liu

Department of Computer Science and Information Engineering  
Nation Chiao Tung University  
Hsinchu, Taiwan 300

## Abstract

*This paper presents a load balancing algorithm for a Time Warp discrete event simulation running on non-dedicated heterogeneous processors. This algorithm dynamically balances the load on processors in order to reduce the number of rollbacks, and thus increase the total simulation speed. Simulation processes are allowed to migrate according to the load on processors. An emulated multiprocessor environment was developed in order to evaluate the algorithm. The simulation results indicate that the running time of the Time Warp simulation can be substantially reduced.*

## I. Introduction

Parallel simulation mechanisms broadly fall into two categories: conservative and optimistic [3][11][13]. *Conservative* approaches strictly avoid the possibility of any causality error ever occurring. These approaches rely on some strategy to determine when it is safe to process an event. On the other hand, *optimistic* approaches use a detection and recovery approach, which can detect causality errors, and employ a rollback mechanism [5] to recover. Comparing to conservative approaches, optimistic approaches have the advantage that it allow the simulator to exploit parallelism in situations where causality errors may occur.

The *Time Warp* mechanism, based on the Virtual Time paradigm, is the most well-known optimistic protocol [4][9][10]. It assumes that a large percentage of event messages arriving at a process will have timestamps greater than or equal to the local process simulation clock.

Whenever a message arrives with an earlier time stamp

---

\*This research was supported by National Science Council of Republic of China under grant No. NSC-83-0408-E-009-001.

$T_{early}$  the simulation state of the process is rolled back to that time. Therefore, it may be necessary to cancel previously sent messages bearing a timestamp larger than  $T_{early}$

Reiher and Jefferson present preliminary results for a method based upon the notion of effective utilization [14]. They define effective utilization to be the proportion of work done by a processor that is not rolled back. Their algorithm balances the effective utilization of the processors by using a bin-packing algorithm in conjunction with a process migration protocol. They apply their algorithm to a military simulation and a colliding puck simulation, comparing the results to a static load balance of the simulations. The results presented do not give evidence of its importance over a good static algorithm. A speedup of up to 25% was observed over a random process allocation.

A recent paper by Glazer and Tropper [7] presents a new load balancing algorithm for Time Warp parallel simulation and claims a substantial performance gain. They define the simulation advance rate to be the rate at which a process advances its simulation clock as a function of the amount of CPU time it is given. And they minimize the difference between the local simulation clocks by controlling the amount of CPU time allocated to each process, which is achieved by assigning longer time slices to slower processes. To solve the imbalance coming from the time slice adjustment of processes in the processor load, they migrate processes between different processors.

Glazer and Tropper applied their algorithm to three simulation models, manufacturing pipeline model, hierarchical communication network model, and distributed network model, and compare their result to a reasonable static allocation in all three models and a remarkable speedup of up to 29% for manufacturing pipeline model.

The load of a process defined in Glazer et al's algorithm is a measure of the amount of CPU time it requires to advance its local simulation clock one unit. Afterward, Glazer et al's algorithm uses these load information to allocate to each processor a percentage of the total load proportional to its speed. However, during the load information collecting phase, their algorithm ignores the fact that faster processor can execute more instructions than the slower one during the same period. Therefore, their algorithm cannot accurately estimate the real process load in a heterogeneous environment, and thus cannot optimally balance the load. Furthermore, their algorithm is not applicable to a non-dedicated system in which the system load on each processor varies in time. In next section, we present our algorithm which can be applied to heterogeneous and non-dedicated systems.

## II. A New Load Balancing Algorithm

The new load balancing algorithm we proposed is based on Glazer's algorithm [7]. The load balance calculation is performed in two steps. First, the lengths of process time slices are derived from their individual simulation rates. The second step is to allocate processes to processors with a bin packing algorithm.

The load of process  $m$ ,  $Load_m$ , is a measure of the amount of CPU time it requires to advance its local simulation clock one unit. For heterogeneous systems, we give different weight to processors with different speed. It is calculated as

$$Load_m = \frac{CPU_m \times SpeedRatio_n}{Advance_m}$$

$$SpeedRatio_n = \frac{MIPS_n}{TypicalMIPS}$$

where  $CPU_m$  and  $Advance_m$  are the CPU allocation and simulation advance of the process.  $SpeedRatio_n$  is the weight given to processor  $n$ , where faster processor is given greater weight.  $TypicalMIPS$  can be any constant value, and therefore it is reasonable to set the mean value of all processors'  $MIPS$  to  $TypicalMIPS$ . If the number of processes is  $M$ , the mean load is

$$MEAN = \sum_{m=1}^M Load_m / M$$

The length of a process time slice,  $Slice_m$ , is determined by its load. Processes are given time slices in proportion to the ratio of their load to the mean. Processes having loads equal to the mean are allocated time slices of

length  $BasicSlice$ , and is subject to the bounds of  $MinSlice$  and  $MaxSlice$

$$Slice_m = BasicSlice \times \frac{Load_m}{Mean} \left. \vphantom{\frac{Load_m}{Mean}} \right\} \begin{array}{l} MaxSlice \\ MinSlice \end{array}$$

where the  $Slice_m$  is estimated with the assumption that all processors are identical. Therefore, in a system with heterogeneous processors, after the finish of the processes allocating to processors, the actual time slice for process  $m$  on processor  $n$ ,  $Quota_m$ , is calculated as

$$Quota_m = \frac{Slice_m}{SpeedRatio_n}$$

With time slice lengths derived in this manner, the execution of a single time slice by each process will advance their local simulation clocks by approximately the same amount on any processor. Therefore, in a non-dedicated system, the system load varies in time. By the principle of time locality, we can predict the future available CPU time from the recent collected load information. To settle a perfect processor allocation, we should allocate each processor a percentage of the total load proportional to its estimated future available CPU time multiplied by its speed (i.e., a faster or unoccupied processor should receive a greater part of the load). This percentage is computed as

$$MIPFRAC_n = \frac{EffCPU_n \times MIPS_n}{\sum_{i=1}^N EffCPU_i \times MIPS_i}$$

where the  $EffCPU_n$  is the sum of CPU time used by all processes belonging to this parallel simulation on processor  $n$  during the previous scheduling period. The load allocated to processor  $n$ ,  $Alloc_n$ , is  $MIPFRAC_n$  multiplied by the total amount of each time slice.

$$Alloc_n = MIPFRAC_n \times \sum_{m=1}^M Slice_m$$

This leaves the problem of allocating  $M$  processes with slice times of  $Slice_m$  to  $N$  processors with total slice time limits of  $Alloc_n$ . This is the classical "Bin Packing" problem. In many cases it is impossible to get a "perfect fit" (i.e., a solution where each process gets its desired load). Instead, we look for the "best fit" solution. This problem can has been shown to be intractable (NP complete)[6]. Therefore, a heuristic must be used. The heuristic tries to find an allocation in which the difference between the desired processor loads and the

actual load is within a specified (tolerance) range. It also attempts to limit the number of process migrations.

In order to compare the experiment results, we chose the same "tolerance" of 10% as that used by Glazer and Troppers. Once the load of all processors is within 10% of their desired load, the algorithm terminates. The tolerance helps reduce both the running time of the heuristic, and the number of costly process migrations. Note that the migrations eliminated are ones that produce small speedups. As the tolerance approaches zero, the number of process migrations increases.

The following is a brief description of the bin packing algorithm employed. A more detailed description with pseudo-code may be found in [8]. The reader should note that any bin packing algorithm may be employed, as long as it tends to a minimization of the number of process migrations.

#### Bin Packing Algorithm

- 0) In the case of the initial load balance, use a uniform random allocation.
- 1) Move single processes from one processor to another, until no better balancing can be achieved.
- 2) Exchange pairs of processes between two processors, until no better balancing can be achieved.

### III. Experimental Model

This section describes the simulation models, and system environments employed in our experiments.

#### A. Simulation Model

Our simulation model is a manufacturing pipeline (Fig. 1). This model is like the production line in factories. The model has multiple stages of events, and stages can work simultaneously. Many other applications can also be modeled into this type. Therefore, we chose the pipeline model to compare our load balancing algorithm with others.

In this model, events flow from two sources to two sinks. The pipeline consists of thirty processes organized into nine stages. All processes in the same stage perform the identical task and have delays drawn from the same normal distribution with a standard deviation of 10%. Routing decisions are made randomly at each node of the pipeline.

The primary sources of rollbacks in this model are the multiplicity of paths between the sources and sinks, coupled with the variability of service times. There are

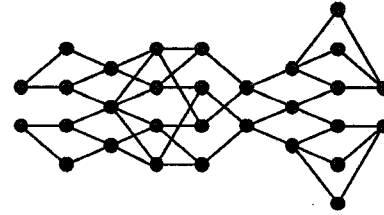


Fig. 1. Manufacturing Pipeline model

no feedback loops and all possible paths are of the same length.

In order to raise the maximum throughput of the simulation model on various conditions (the number of processors, scheduling algorithm, etc.). The source processes generate events continuously, that is, after the source process finish an event and send it to child process, another event is ready to execute. In other words, the event generate rate depends on the amount of CPU time which the source node can obtain. However, boundless difference of event generate rate between source processes will drive the simulation extremely unbalance and may produce tremendous abnormal rollbacks. In order to keep all source processes with almost the same event generate rate, we force that one source process can not acquire new event until the other source processes finish their current events.

#### B. Testing Environment

The Time Warp simulation model was executed in simulated multiprocessor environment. This environment was emulated on a uniprocessor. The INTEL hypercube multiprocessors were chosen as the model for the design of our emulator. The basic units of abstraction were the Time Warp simulation process. Low level instruction execution was not emulated. Simple kernel program was designed to manage the basic structures and mechanisms for each Virtual Processor (VP). Local VP execution was determined by a two-level smallest-timestamp-first process scheduler. Each VP maintained its own local clock as well as the inter-processor communication mechanism.

To calculate the communication delay,  $T_{delay}(k)$ , the function below was applied

$$T_{delay}(k) = T_{startup}(k) + k \times T_{send}(k) + (h - 1) \times T_{hop}(k)$$

- $k$  is the number of bytes in the message.
- $h$  is the number of hops the message must traverse.

- $T_{startup}(k)$  is the time required to prepare the message for transmission. It is a function of the message size.
- $T_{send}(k)$  is the time require to transmit a message.
- $T_{hop}(k)$  is the communication delay incurred at each hop beyond the initial one.

Process migration delays are the sum of the process setup times and the memory transfer delays. We obtained these values for the migration delay from [1]. It takes about 500 milliseconds to migrate a typical process of 100 kilobytes, which is about half of the average time for the execution of an event. Each algorithm was evaluated under the multiprocessor of various sizes (4, 8, and 16 processors) in our experiments. Each simulation ran for 500000 time units(ms), during which statistics on roll-backs and simulation rates were gathered. Different random seed was employed to generate multiple results of each model.

In order to test the algorithm's responsiveness to various systems, the simulation model was executed under three different system states, heterogeneous dedicated system, homogeneous non-dedicated system, and heterogeneous non-dedicated system.

In heterogeneous dedicated system states, we emulate different systems with two version of CPU speed setting, a quarter of the processors with twice speed (minor speed difference) or five times speed (moderate speed difference) of the others. All processors are dedicated to simulating processes with no background workload.

In homogeneous non-dedicated system states, all processors execute in the same speed. And the random walk process is employed to model background workload on each processors. Assume that the simulation workload is one unit, the initial background workload is  $x$  units,  $x$  may be larger or smaller than one. After a certain time period, the background workload  $x$  changes to  $x'$  by adding a uniform random number between a variance interval of  $[+n,-n]$ . The random variance interval determines the rate of system load variability. In our simulation, the initial background workload  $x$  is equal to the simulation workload. Two different random interval, 0.05 (minor load variability) and 0.25 (moderate load variability) are employed to evaluate the algorithms under different system load variation.

In heterogeneous non-dedicated system, we use the moderate speed difference setting and the background

workload as 1 and the random interval as 0.25 (moderate load variability).

## IV. Results and Analysis

In this section, we present the experimental results using the models and algorithms described in previous sections. Our experiment results will be compared with other two scheduling method, the Glazer and Tropper's algorithm and the random, uniform allocation method. The results are listed in tables and depicted graphically. These graphs depict the simulation rate (the entire simulation advances) as a function of the number of processors.

### A. heterogeneous dedicated system states:

- (1) Minor difference of speed setting (A quarter of the processors are twice the speed than the others):

In this heterogeneous system, our algorithm produced noticeable speedup of 26-51% over the random scheduling method. However, the relative closeness of the speed between different processors left very little room to reveal the inaccuracy of Glazer and Tropper's algorithm on heterogeneous system. Consequently, our algorithm shows only 5-10% faster than theirs. The result is show in Table 1 and Figure 2.

Algorithm\Processor#	4	8	16
Random	1778.6	3612.8	6344.4
GlazerTropper	2254.5	4171.0	8652.1
OurAlgorithm	2364.9	4543.7	9560.4

Table 1. Simulation rates of minor difference speed setting in the heterogeneous dedicated system

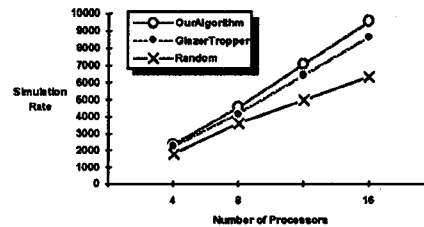


Fig. 2. Simulation rates versus number of processors for minor difference of speed setting in the heterogeneous dedicated system

- (2) Moderate difference of speed setting (A quarter of the processors are five times the speed than the others):

By balancing the simulation advance rates, our algorithm produced the speedup of 60-106% over the

random scheduling method. In the heterogeneous system with moderate differences between processors, our algorithm is faster than Glazer and Tropper's algorithm by 12-31%. The result is shown in Table 2 and Figure 3.

Algorithm\Processor#	4	8	16
Random	1780.1	4683.9	9034.6
GlazerTropper	3270.8	6215.0	11588.5
OurAlgorithm	3672.0	7516.6	15224.6

Table 2. Simulation rates of moderate difference speed setting in the heterogeneous dedicated system

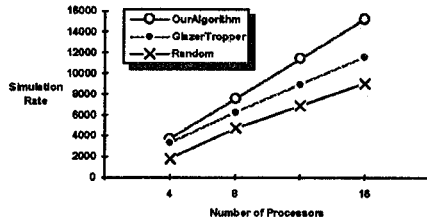


Fig. 3. Simulation rates versus number of processors for moderate difference speed setting in the heterogeneous dedicated system

### B. homogeneous non-dedicated system states

#### (1) Minor load variation:

By predicting the available time ratio of processors, our algorithm provides the speedup of 22-57% over the random scheduling method. Moreover, our algorithm is faster than Glazer and Tropper's algorithm by 7-33% in the non-dedicated system with minor load variability. The result is shown in Table 3 and Figure 4.

Algorithm\Processor#	4	8	16
Random	753.9	1437.4	2538.2
GlazerTropper	862.9	1617.8	2986.7
OurAlgorithm	923.9	1854.1	3975.5

Table 3. Simulation rates of minor load variability setting in the homogeneous non-dedicated system

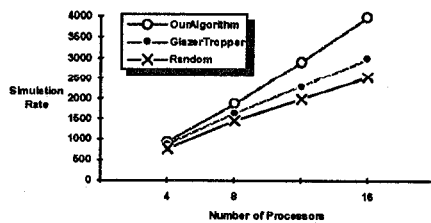


Fig. 4. Simulation rates versus number of processors for minor load variability setting in the homogeneous non-dedicated system

#### (2) Moderate load variation:

Our algorithm improves the simulation rate by 35-59% over the random scheduling method, and by 17-48% over the Glazer and Tropper's algorithm in the non-dedicated system with moderate load variation. Furthermore, we observe that the Glazer and Tropper's algorithm only provide little speedup over the random scheduling method. The result is shown in Table 4 and Figure 5.

Algorithm\Processor#	4	8	16
Random	500.8	996.8	1635.2
GlazerTropper	595.0	1101.2	1756.0
OurAlgorithm	697.1	1349.5	2607.1

Table 4. Simulation rates of moderate load variation in the homogeneous non-dedicated system

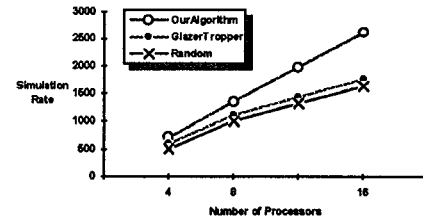


Fig. 5. Simulation rates versus number of processors for moderate load variability setting in the homogeneous non-dedicated system

### C. heterogeneous non-dedicated system

Under the moderate difference speed setting and moderate load variation, our algorithm produces the speedup of 76-97% over the random scheduling method, and the speedup of 14-54% over the Glazer and Tropper's algorithm in the heterogeneous non-dedicated system. The result is shown in Table 5 and Figure 6.

Algorithm\Processor#	4	8	16
Random	667.6	1617.4	2645.6
GlazerTropper	1151.7	2261.1	3281
OurAlgorithm	1317.9	2853.4	5045.5

Table 5. Simulation rate of the heterogeneous non-dedicated system

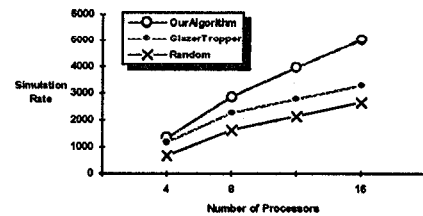


Fig. 6. Simulation rates versus number of processors for heterogeneous non-dedicated system

## V. Conclusions

In order to speedup the Time Warp parallel simulation on a heterogeneous and non-dedicated multiprocessor system, a new load balancing algorithm was proposed. This algorithm minimizes the difference between local simulation clocks to reduce the number of rollbacks. Our simulation results indicate that our algorithm provide better performance when the difference between processors' speed and/or the variation of background workload is large. In the heterogeneous non-dedicated system with sixteen processors, our algorithm provides substantial speedup up to 97% over the random scheduling method, and 54% over the Glazer and Tropper's algorithm. The simulation results indicate that our algorithm can reduce the running time of the parallel Time Warp simulation substantially in heterogeneous and non-dedicated systems.

## References

- [1] Y. Artsy and R.Finkel, "Designing a process migration facility," IEEE Comput. Mag., Sep. 1989
- [2] K. M. Chandy, and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," Commun. ACM 24, 11, Nov. 1981, pp. 198-205
- [3] R. M. Fujimoto, "Parallel discrete event simulation," Commun. ACM, vol. 33, no. 10, pp. 33-53, Oct 1990.
- [4] R. M. Fujimoto, "Time Warp on a shared memory multiprocessor," Trans. Soc. for Computer Simul. 6, 3, July 1989, pp, 211-239.
- [5] A. Gafni, "Rollback mechanisms for optimistic distributed simulation systems," Proceedings of the SCS Multiconference on Distributed Simulation 19, 3, July 1988, pp.61-67.
- [6] M. G. Garey and D. S. Johnson,"Computers and Intractability : A Guide to the Theory of NP Completeness." San Francisco, CA: Freeman, 1979 .
- [7] D. Glazer and C. Tropper, "On Process Migration and Load Balancing in Time Warp," IEEE Trans. on Parallel and Distributed Systems, Vol. 4, No. 3, March 1993, pp 318-327.
- [8] D. W. Glazer, "Load balancing a Time Warp simulation," Tech. Rep. TR-SOCS-91.1, School of Computer Sci, McGill Univ., Montreal, P.Q., Canada,1991.
- [9] D. R. Jefferson, "Virtual time," ACM Trans. Prog. Lang. and Syst. 7, 3, July 1985, pp. 404-425.
- [10] D. R. Jefferson and H. Sowizral, "Fast concurrent simulation using time warp mechanism," Proc. of the SCS Multiconference on Distributed Simulation 1985, pp 63-69.
- [11] M. Livny, "A study of parallelism in distributed simulation," Proceedings of the SCS Multiconference on Distributed Simulation 15, 2, Jan. 1985, pp.94-98.
- [12] G. Lomow, J. Cleary, B. Unger, and D. West, "A performance study of Time Warp," Proceedings of the SCS Multiconference on Distributed Simulation 19, 3, July 1988, pp. 50-55.
- [13] J. Misra, "Distributed-discrete event simulation," ACM Computer Survey 18, 1, Mar. 1986, pp. 39-65.
- [14] P. Reiher, and D. R. Jefferson, "Virtual time based dynamic load management in the Time Warp operating system," Distributed Simulation 22, 1, 1989