Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

MANAGING EXTERNAL WORKLOAD WITH BSP TIME WARP

Malcolm Yoke Hean Low

Programming Research Group, Computing Laboratory
University of Oxford
Wolfson Building, Parks Road
Oxford, OX1 3QD, U.K.

ABSTRACT

This paper describes an extension to the existing BSP Time Warp dynamic load-balancing algorithm to allow the management of interruption from external workload. Experiments carried out on a manufacturing simulation model using different partition strategies with and without interruption from external workload show that significant performance improvement can be achieved with external workload management.

1 INTRODUCTION

Most of the studies on parallel simulation are conducted on dedicated systems and many experiments even go to great length to ensure minimum external interference on the simulation runs. As a result, most existing parallel simulation protocols are designed with the assumption that uninterrupted system resources are available to execute simulation runs.

With widespread use of large scale computing cluster, a new approach in the design of parallel simulation protocols is required. Very often computing resources in these clusters are not dedicated and are usually shared among multiple users. The workload on each computing node in the clusters can fluctuate widely due to the presence of jobs from other users.

This scenario poses a great challenge for any parallel simulation protocol to run efficiently. In this paper, an extension to the dynamic load-balancing (DLB) algorithm for the Bulk Synchronous Parallel Time Warp (BSP-TW) optimistic simulation protocol is described. Experiment results on a manufacturing simulation model show that significant performance improvement could be achieved using the DLB algorithm for BSP-TW.

The rest of this paper is organized as follows. Section 2 describes the BSP model and the BSP Time Warp optimistic protocol. In Section 3, the existing DLB algorithm for BSP-TW is described. An extension to the DLB algorithm for BSP-TW that is capable of managing external workload is then proposed. Section 4 presents experiment results using the new DLB algorithm on a manufacturing simulation model. Some related work is described in Section 5. Section 6 summarizes the paper and outlines future research directions.

2 BSP TIME WARP

The BSP model (Valiant 1990) provides a general purpose approach to parallel computing that allows the separation of concerns between computation, synchronization and communication costs. It has a simple cost model for predicting the performance of BSP algorithms on different parallel platforms. A BSP programming model consists of P processors linked by an inter-connecting network and each with its own pool of memory.

A BSP algorithm consists of a set of processors each executing a series of supersteps. Each superstep consists of three ordered phases: 1) a local computation phase, where each processor can perform computation using local data and issue communication requests; 2) a global communication phase, where data is exchanged between processors according to the requests made during the local computation phase; and 3) a barrier synchronization, which waits for all data transfers to complete and makes the transferred data available to the processors for use in the next superstep.

The BSP-TW algorithm (Marín 1998) shown in Figure 1 is designed to be an efficient realization of an optimistic synchronization protocol (Jefferson 1985, Jefferson and Sowizral 1985) on the BSP model. Each processor manages a group of logical processes (LPs) in the system. In BSP-TW, LPs are also referred to as simulation objects and the two terms are used interchangeably in this paper. LPs in the same processor share a common event-list. A series of supersteps are executed by each processor as indicated by the outer `while` loop and the `bsp_sync()` statement at the end of the loop.

```
bsp_begin();
[A] Initialization
while GVT < SimEndTime do
    [B] Receive external events and process rollback;
    [C] Compute new GVT, perform fossil collection and
        compute new event limit n_e every n_g supersteps;
    [D] Execute n_e events;
    bsp_sync();
endwhile
bsp_end();
```

Figure 1: Algorithm for BSP Time Warp

The global virtual time (GVT) measures the progress of a simulation run. An estimate of GVT is computed after every n_g supersteps; n_g is also known as the GVT update interval. The body of the loop terminates when the GVT value is greater than the simulation end time.

The algorithm provides an automatic means of throttling the number of events, n_e, being simulated per superstep based on statistics from fossil collected events. The BSP cost model for a BSP-TW algorithm S can be expressed as

$$\text{cost}(S) = \sum_{i=1}^{n_s} (w(i) + gh(i) + L) \qquad (1)$$

where n_s is the total number of supersteps; $w(i)$ is the computation cost for superstep i; and $h(i)$ is the maximum number of messages sent or received respectively by any processor in superstep i. The architecture dependent parameters g and L represent the communication and synchronization costs respectively.

Although the cost model is relatively simple, we can see that the performance of a BSP-TW algorithm relies on three factors: a) computation balance; b) communication balance; and c) n_s, the total number of supersteps.

Computation and communication imbalance can result from the dynamic changing nature of the workload of the simulation model and interruption from external workload. The total number of supersteps required to complete the simulation depends on the lookaheads on the links between LPs on different processors. Lookahead is defined as the minimum simulation time interval between event arrival, from the source to a destination LP. A dynamic load-balancing algorithm can reduce both computation and communication load-imbalance, as well as optimize lookaheads by migrating simulation objects between processors.

3 DYNAMIC LOAD-BALANCING FOR BSP TIME WARP

The BSP-TW DLB$_{ccl}$ algorithm first described in Low (2002) has facilities to dynamically balance computation and communication load-imbalance, as well as optimize lookaheads between processors. However, the algorithm does not take into account interruption from external workload. For example, Figure 2a shows the computation workload of five processors in a superstep. The shaded boxes show that an external workload is present in processor $P0$. Although all five processors have the same computation workload (represented by the white boxes, each white box can be considered a simulation object), processor $P0$ takes twice as long to complete the supersteps since the CPU cycles are shared between the simulation workload and the external workload.

Figure 2b shows how BSP-TW DLB$_{ccl}$ algorithm balance the computation workload in the superstep across all processors. All the simulation workload on processor $P0$ would be distributed to other processors. Note that in this case BSP-TW DLB$_{ccl}$ carries out computation load-balancing based purely on the length of time each processor takes to complete the superstep. It has no knowledge of the presence of external workload on processor $P0$.

Since processor $P0$ is now without any computation workload, it can complete each subsequent superstep with minimum delay. However, without any knowledge of the presence of external workload on processor $P0$, the BSP-TW DLB$_{ccl}$ algorithm will consider $P0$ to be idle and assume that there is an imbalance in computation workload. It will proceed to migrate simulation objects back into processor $P0$. The load configuration of the system again returns to that shown in Figure 2a. This results in a thrashing situation in which simulation objects are migrated in and out of the processor plagued by external workload.

3.1 BSP-TW DLB$_{ccle}$ Algorithm

In this section, we propose an extension to the BSP-TW DLB$_{ccl}$ algorithm to allow external workload management. The new algorithm is referred to as BSP-TW DLB$_{ccle}$. Figure 3 shows the pseudo-code for the new BSP-TW DLB$_{ccle}$ algorithm. The BSP-TW DLB$_{ccle}$ algorithm consists of four modules and is executed at each migration point, which occurs every λn_g supersteps ($\lambda \geq 1$). We also refer to the λn_g supersteps between two migration points as a migration interval. At each migration point, one of the four modules will be activated based on factors such as the amount of external workload, computation imbalance as well as communication imbalance.

The computation load-balancing in module D1 is carried out by transferring simulation objects from processors with high computation workload to processors with low

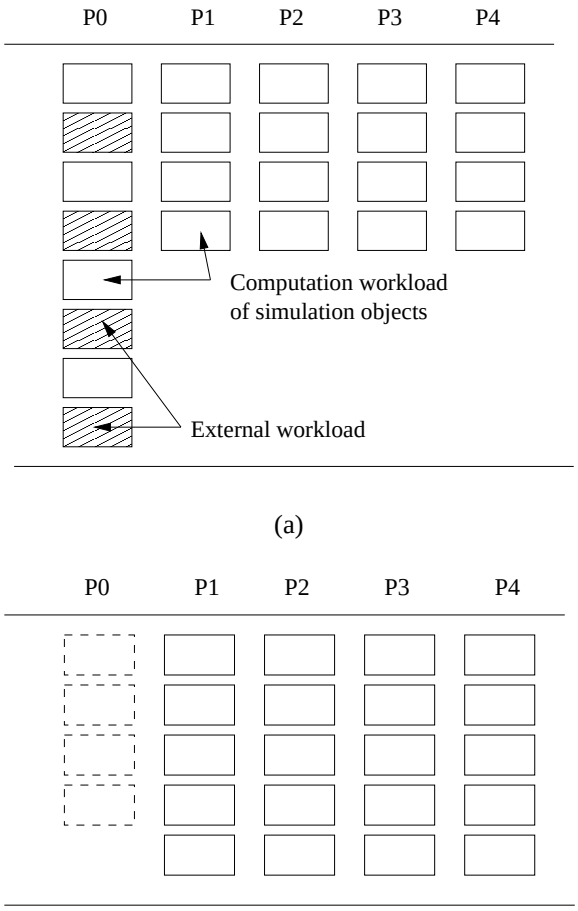

Figure 2: Example of Interruption from External Workload

```
bsp_begin();
[A] Initialization;
while GVT < SimEndTime do
    [B] Receive external events and process rollback;
    [C] Compute new GVT, perform fossil collection and
        compute new event limit n_e every n_g supersteps;
    [D] After each λ GVT computation:
            D0 balance_extLoad();
            D1 balance_computation();
            D2 balance_communication();
            D3 optimize_lookahead();
    [E] Execute n_e events;
    bsp_sync();
endwhile
bsp_end();
```

Figure 3: Algorithm for BSP-TW DLB$_{ccle}$

```
balance_extload()
    foreach processor P_i do
        if P_i.loadavg > θ then
            migrate_all(P_i);
            set P_i as inactive;
        else if P_i.loadavg < θ/2 then
            set P_i as active;
        endif
    endfor
```

Figure 4: Algorithm for Balancing External Workload

computation workload. For module D2, communication load-balancing is carried out by exchanging simulation objects between processors. The module uses load exchange, rather than load transfer to preserve the computation balance achieved in module D1, at the same time improving the balance in communication workload. The lookaheads optimization in module D3 is carried out by merging simulation objects with small lookaheads into the same processor. For more detailed explanation of these three modules, readers are referred to Low (2002).

The BSP-TW DLB$_{ccl}$ algorithm described in Low (2002) is enhanced with module D0 in order to handle computation and communication load-imbalance due to the presence of external workload. The pseudo-code for module D0 is shown in Figure 4.

The state variable P_i.loadavg is used to track the average load of processor P_i. We classify the set of processors with average load greater than the processor load threshold parameter, θ, as heavily loaded. The average load of a processor is obtained by a UNIX system call `getloadavg()`. This system call returns the number of processes in the system run queue averaged over various periods of time. The 1 minute sample returned by the system call is used in the experiments.

At each migration point, the BSP-TW DLB$_{ccle}$ algorithm attempts to evict all the simulation objects out of these heavily loaded processors. The method `migrate_all(`P_i`)` evicts all the simulation objects in processor P_i to other processors with normal workload in a round-robin fashion. The status of processor P_i is then set to inactive. As the dynamic load-balancing modules D1 to D3 only consider the set of active processors, simulation objects will not be migrated back to the processors that are still heavily loaded with external workload. When a previously heavily loaded processor's average load drops below $\frac{\theta}{2}$, the status of the processor is reset to active. This causes the computation and communication load-balancing modules to detect the idle processor and allows simulation objects to be moved back to the processor.

4 EXPERIMENTS WITH MANUFACTURING SIMULATION MODEL

4.1 Simulation Model

In order to study the efficiency of the BSP-TW DLB$_{ccle}$ algorithm, experiments are carried out on a manufacturing simulation model. The same model was used in Lim et al. (1998) to study different runtime systems for a conservative simulation protocol. The manufacturing model consists of different entities of a typical production line with an assembly and test facility. Figure 5 shows the layout of a production line and an assembly and test facility. The configuration of the manufacturing model consists of a total of seven production lines. Each production line consists of 100 production stages. The assembly and test facility consists of 100 assembly stations and 100 testing stations. There are a total of 2417 simulation objects in this model.

This manufacturing model is a challenging model for optimistic simulation protocol such as BSP-TW due to the presence of many zero lookahead links on the `fork` and `join` nodes. Lookahead is crucial to the performance of the BSP-TW protocol since processors with many incoming communication links with small or zero lookaheads are likely to suffer from high event rollback rates.

For all the experiments, the GVT computation interval n_g is fixed at 50 supersteps. The migration interval λ is set to 5. A load threshold parameter of θ=1.5 is used. The experiments are conducted on a cluster of eight 350MHz Sun UltraSparc workstations connected via a 100Mbits TCP/IP network. All execution times shown are the average of 10 runs. Unless stated otherwise, the simulation run length of the experiments are 10^4 time unit.

Experiments are carried out using BSP-TW on the manufacturing model with two different partitioning methods. The first partitioning method is to assign simulation objects in a round-robin fashion to the processors. This model is referred to as M_u. The second method is to assign consecutive block of 25 simulation objects onto the same processors. This model is referred to as M_b. Both models M_u and M_b have well balanced computation and communication workload in the absence of external workload. However, model M_u is expected to perform worse than M_b since the round-robin assignment on M_u will result in many zero-lookahead links between processors.

Besides using different partitioning strategies, external workload is also introduced onto different numbers of processors. Two types of external workload are used: persistent and transient workload. The persistent external workload is introduced from the start of the simulation and lasts through the entire simulation duration. The transient external workload is introduced sometime after the simulation is started and lasts for a fixed duration.

Table 1: Execution Times (sec.) with Persistent External Workload

Ext. Load	Protocol	M_b	M_u
0	BSP-TW	394.4	1730.0
	BSP-TW DLB$_{ccl}$	386.4	642.8
	BSP-TW DLB$_{ccle}$	392.0	628.3
1	BSP-TW	646.6	2541.8
	BSP-TW DLB$_{ccl}$	572.0	1082.4
	BSP-TW DLB$_{ccle}$	495.1	818.7
2	BSP-TW	983.8	4102.2
	BSP-TW DLB$_{ccl}$	723.1	1034.0
	BSP-TW DLB$_{ccle}$	527.1	888.1

4.2 Persistent External Workload

For the first set of experiments, the manufacturing models M_b and M_u are executed using the original BSP-TW, BSP-TW DLB$_{ccl}$, and BSP-TW DLB$_{ccle}$ protocols. External workload is applied throughout the entire simulation duration on one of the processors. The number of external workload is varied from 0 to 2.

Table 1 shows the execution times for models M_b and M_u using different BSP-TW protocols under different numbers of external workload. For the case with no external workload, the performance for all three protocols on model M_b is similar since the model has both well-balanced computation and communication workload, as well as good lookahead configuration. For model M_u, the lookahead optimization module in both the BSP-TW DLB$_{ccl}$ and BSP-TW DLB$_{ccle}$ protocols yields significant improvement in performance compared to the BSP-TW protocol.

In general, we expect the performance of BSP-TW DLB$_{ccl}$ and BSP-TW DLB$_{ccle}$ to be similar in the absence of external workload. The small difference in execution times between BSP-TW DLB$_{ccl}$ and BSP-TW DLB$_{ccle}$ for model M_u is attributed to the variation in system load.

Table 1 shows that the performance of BSP-TW on both models deteriorates as external workload is introduced. While the performance for model M_b is only affected by the presence of external workload, the performance for model M_u is affected both by the presence of external workload as well as the poor lookahead configuration.

Figure 6 shows the GVT rates for the different algorithms on both models under different numbers of external workload for the first 1000 units of execution time. GVT rate is defined as the ratio between the advancement of GVT between two migration points and the amount of elapsed wall-clock time. We see that both BSP-TW DLB$_{ccl}$ and BSP-TW DLB$_{ccle}$ significantly improve the GVT rates of the simulation.

For the case with only one external workload, the BSP-TW DLB$_{ccl}$ improves the GVT rates of model M_u to the same level as that of the BSP-TW on model M_b. This shows

Low

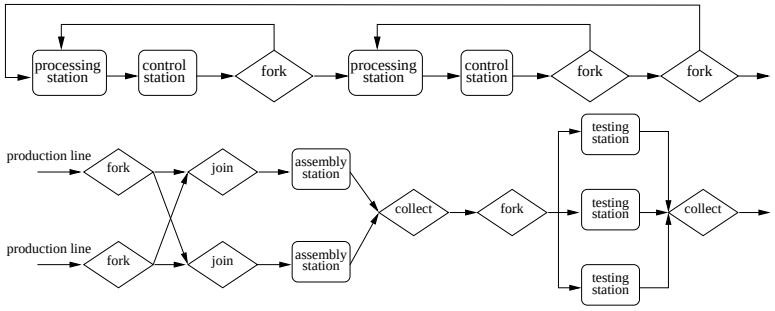

Figure 5: Layout of a Production Line and an Assembly and Test Facility

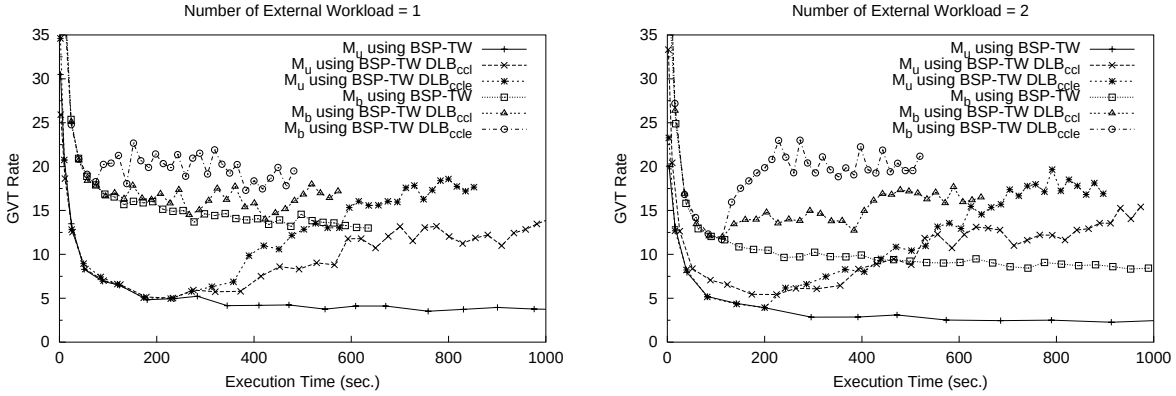

Figure 6: GVT Rates for Models M_b and M_u with Persistent External Workload

that the improvement is attributed mainly to the lookahead optimization. On the other hand, the BSP-TW DLB$_{ccle}$ is able to both optimize lookahead as well as handle the presence of the external workload. This is evident from the much higher GVT rates for BSP-TW DLB$_{ccle}$ on model M_u in Figure 6 compared to that for BSP-TW on model M_b.

When the number of external workload is increased to two, the load-balancing capability of BSP-TW DLB$_{ccl}$ becomes apparent. This enables BSP-TW DLB$_{ccl}$ to achieve a much higher GVT rate on model M_u compared to BSP-TW on model M_b.

4.3 Transient External Workload

In the second set of experiments, we examine the effects of loading one of the processors in the simulation system with transient external workload to observe the behaviour of the system in the presence of the external workload and after the workload has been removed. This set of experiments is performed only on model M_b and the length of the simulation is extended from 10^4 to 2×10^4 time unit. The external workload is applied 200 seconds after the simulation begins. We experimented with transient external workload that last for 200 and 600 seconds respectively. The number of external workload on each processor is also varied from 1 to 4.

Table 2 shows the execution times for the model using BSP-TW, BSP-TW DLB$_{ccl}$ and BSP-TW DLB$_{ccle}$ algorithms. The performance of BSP-TW DLB$_{ccle}$ degrades compared to BSP-TW and BSP-TW DLB$_{ccl}$ for the set of experiments with transient workload of 200 seconds. The effects of external workload management are more evident when the duration of the transient external workload is increased to 600 seconds. The increased duration of the presence of the external workload makes the eviction of simulation objects by BSP-TW DLB$_{ccle}$ worthwhile and allows it to obtain better performance compared to the other two protocols.

Figure 7 shows the GVT rates for different numbers of external workload that last for 600 seconds. The GVT rate for the BSP-TW algorithm drops proportionally to the number of external workload but returns to its original level once the external workload is removed.

The sharp drop in GVT rates for BSP-TW DLB$_{ccle}$ protocol in both sets of graphs indicates the migration point at which simulation objects are moved back to the processor that was previously loaded with the transient

Table 2: Execution Times (sec.) with Transient External Workload

	Duration = 200 sec.				Duration = 600 sec.			
	No. of Ext. Workload				No. of Ext. Workload			
	1	2	3	4	1	2	3	4
BSP-TW	920.3	964.0	990.9	1014.1	1087.5	1209.3	1262.1	1311.0
BSP-TW DLB$_{ccl}$	929.8	1014.6	1048.5	1108.8	1065.4	1163.0	1215.8	1305.8
BSP-TW DLB$_{ccle}$	1063.9	1099.7	1104.5	1129.5	1067.4	1078.6	1112.8	1165.4

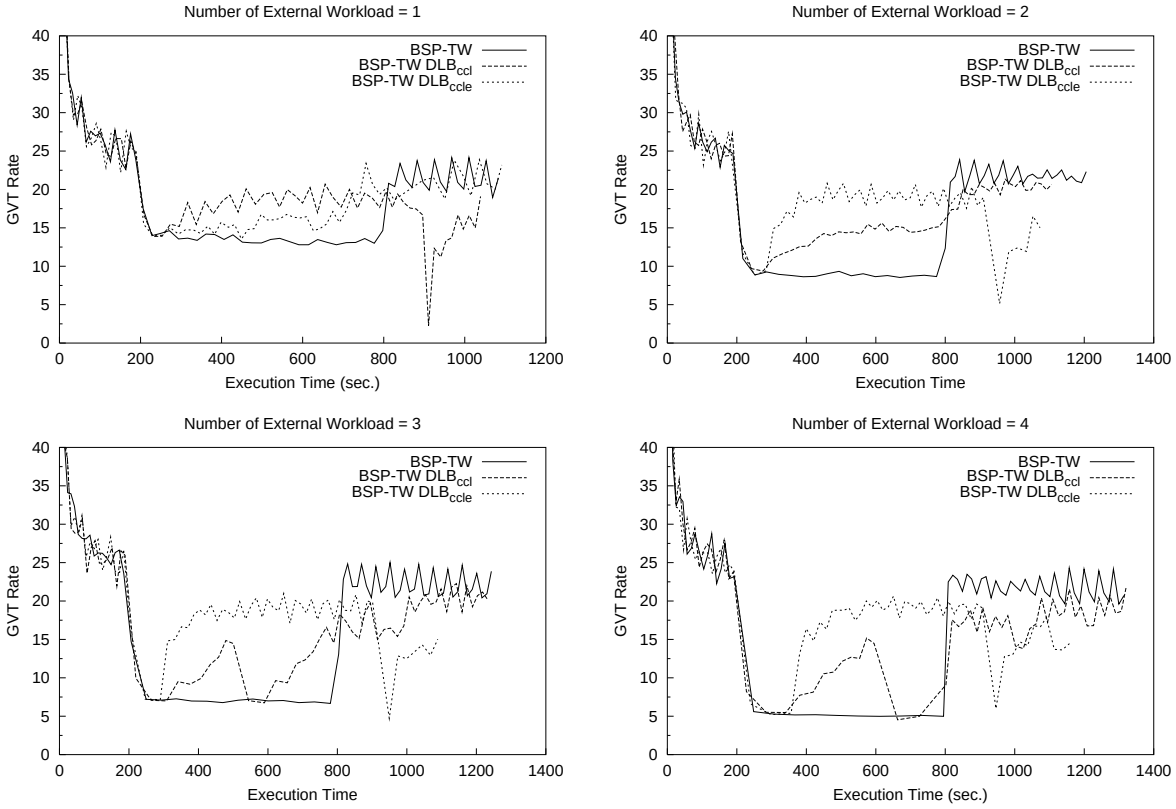

Figure 7: GVT Rates for Model M_b with Different Numbers of Transient External Workload

workload. The re-population of simulation objects onto a processor requires a new event limit to be computed. This resulted in high event rollback rate during the first few GVT computation intervals immediately after the migration point. The movement of simulation objects into the processor also destroys the optimized lookahead configuration between processors. Both the high event rollback rate and the ruined lookahead configuration contribute to the sharp drop in GVT rates.

The graphs show that the BSP-TW DLB$_{ccle}$ algorithm is able to restore the GVT rate to its previous high level. However, a considerable amount of time is required to accomplish this. For the case with a short transient workload of 200 seconds, this means that the benefits obtained from migrating simulation objects out of the loaded processor do not justify the cost of re-populating simulation objects and re-optimizing lookaheads once the external workload is removed.

4.4 External Workload On Multiple Processors

In the final set of experiments, we examine the effects of artificially applying external workload on more than one processor. This set of experiments is conducted using only model M_b with two external workload applied on each of the designated processors throughout the duration of the simulation. The experiments are repeated by loading two processors followed by four processors.

Table 3: Execution Times (sec.) with External Workload on Multiple Processors

	No. of Processor Loaded			
Protocol	0	1	2	4
BSP-TW	394.4	983.8	1002.4	1025.1
BSP-TW DLB$_{ccl}$	386.4	723.1	1050.3	1357.3
BSP-TW DLB$_{ccle}$	392.0	527.1	609.6	949.3

Table 3 shows the execution times for the different algorithms on model M_b with different numbers of processors being loaded. We first note that the performance of the original BSP-TW algorithm is not affected by how many processors are loaded with external workload. This is expected since the BSP cost model predicts that the performance of the BSP-TW protocol is affected by the processor that is the most loaded rather than the number of processors that are loaded.

The performance of BSP-TW DLB$_{ccl}$ deteriorates as more processors are loaded with external workload. While the load-balancing modules try to balance load by migrating simulation objects from heavily loaded processors to less loaded ones, the effects of external workload are not taken into account. The result is that simulation objects are migrated to and fro between heavily loaded and less loaded processors.

Figure 8 shows the accumulated workload of each processor between migration points for the experiment with four processors loaded with external workload. External workload is applied to processors $P2$, $P3$, $P4$ and $P5$. The thrashing behaviour of BSP-TW DLB$_{ccl}$ is evident from the workload of processors $P4$ and $P5$ from time 300 to time 800.

The BSP-TW DLB$_{ccle}$ does not suffer from the thrashing behaviour observed in the BSP-TW DLB$_{ccl}$ algorithm since all heavily loaded processors are not considered for load-balancing purpose. However, as more processors are excluded from computation, the individual workload of the remaining active processors increases and the performance of BSP-TW DLB$_{ccle}$ approaches that obtained using original BSP-TW.

5 RELATED WORK

Past studies of DLB algorithm for optimistic parallel simulation protocols have typically focused on which metrics to use to measure the actual workload of the system. Burdorf and Marti (1993) presented a dynamic load-balancing strategy based on local simulation times matrix. The scheme moves objects from processors that are far behind in simulation time to processors that are further ahead. The aim is to "slow down" the fast processor in order to reduce the amount of rollback. Burdorf and Marti observed that this scheme leads to better load-balance in the presence of external workload. Carothers and Fujimoto (1996) presented an approach for background execution of Time Warp. The scheme allows a Time Warp system to execute in the background and consume unused CPU cycles across a collection of heterogeneous machines. The metric used is "Processor Advance Time" (PAT), which reflects the amount of real time needed to advance the virtual time of a logical process by one unit.

In this paper, the metrics used are the computation and communication workload. The rate of progress in simulation time between supersteps for each processor in BSP-TW is automatically controlled by the adaptive event limit set for each superstep. A comparison of some of the metrics used for DLB of optimistic simulation can be found in El-Khatib and Tropper (1999).

6 CONCLUSION

The BSP-TW DLB$_{ccle}$ described in this paper has facilities to handle computation and communication imbalance, optimize lookahead as well as manage interruption from external workload. The experiment results on a manufacturing simulation model show that significant performance improvement can be achieved using BSP-TW DLB$_{ccle}$ over the original BSP-TW algorithm. However, the presence of transient external workload can result in high overhead in migrating simulation objects to and fro between processors. Further work will involve studying techniques to reduce this overhead as well as preserving the lookahead configuration when migrating simulation objects.

ACKNOWLEDGMENTS

This work is supported by Singapore Institute of Manufacturing Technology.

REFERENCES

Burdorf, C., and J. Marti. 1993. Load Balancing Strategies for Time Warp on Multi-User Workstations. *The Computer Journal* 36 (2): 168–176.

Carothers, C. D., and R. M. Fujimoto. 1996. Background Execution of Time Warp Programs. In *Proceedings of the 1996 Workshop on Parallel and Distributed Simulation*, 12–19.

El-Khatib, K., and C. Tropper. 1999. On Metrics for the Dynamic Load Balancing of Optimistic Simulation. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*.

Jefferson, D. 1985. Virtual Time. In *ACM TOPLAS*, Volume 7, 404–425.

Jefferson, D., and H. Sowizral. 1985. Fast Concurrent Simulation Using Time Warp Mechanism. In *Distributed Simulation 1985*, ed. P. Reynolds, 63–69. La Jolla, Cal-

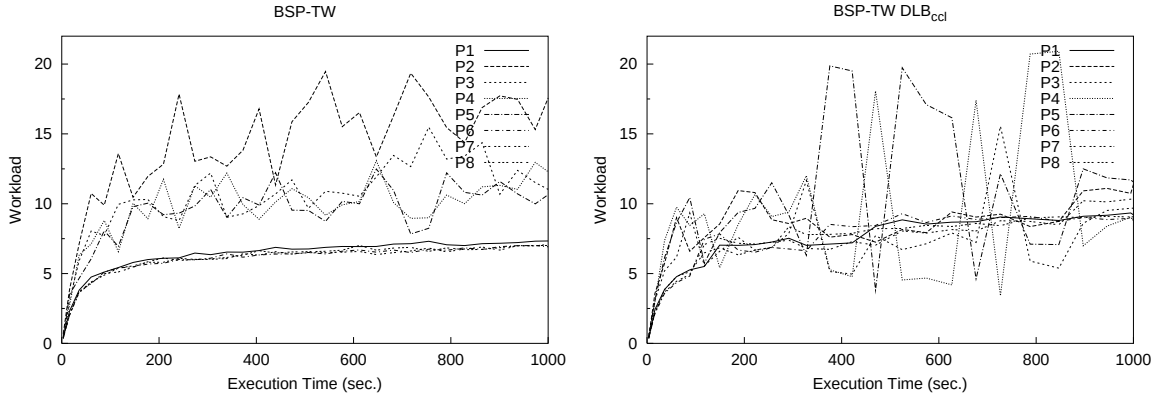

Figure 8: Workload of Individual Processor with Persistent Workload on Four Processors

ifornia: SCS-The Society for Computer Simulation, Simulation Councils, Inc.

Lim, C.-C., Y.-H. Low, W. Cai, W.-J. Hsu, S.-Y. Huang, and S. J. Turner. 1998. An Empirical Comparison of Runtime Systems for Conservative Parallel Simulation. In *2nd Workshop on Runtime Systems for Parallel Programming (RTSPP 1998)*, 123–134. Orlando, Florida, USA.

Low, M. Y. H. 2002. Dynamic Load-Balancing for BSP Time Warp. In *Proceeding of the 35th Annual Simulation Symposium*, 267–274. San Diego, California.

Marín, M. 1998. *Discrete-Event Simulation on the Bulk-Synchronous Parallel Model*. Ph. D. thesis, Oxford University.

Valiant, L. G. 1990. A Bridging Model for Parallel Computation. *Communications of the ACM* 33:103–111.

AUTHOR BIOGRAPHY

MALCOLM LOW received his Bachelor and Master of Applied Science in Computer Engineering from Nanyang Technological University, Singapore, in 1997 and 1999 respectively. He is currently a DPhil student in the Oxford University Computing Laboratory. His research interests are in the areas of adaptive tuning and load-balancing of parallel discrete event simulation systems. His email address is <mlow@comlab.ox.ac.uk>.