A TIME SLICING APPROACH TO EXTERNAL WORKLOAD MANAGEMENT ON BSP TIME WARP

MALCOLM YOKE HEAN LOW
Singapore Institute of Manufacturing Technology
71 Nanyang Drive, Singapore 638075
E-mail: yhlow@SIMTech.a-star.edu.sg

ABSTRACT

The performance of a BSP Time Warp parallel simulation system on a large-scale cluster of workstations can be severely affected due to the presence of external workload on individual machine in the cluster. This paper describes a new approach to managing external workload for BSP Time Warp parallel simulation on a cluster of workstations using the approach of time slicing. Experimental results comparing the performance of this new approach and the one proposed previously show that the new approach is resilient to interruption from external workload on multiple computing nodes in the cluster of workstations.

1 INTRODUCTION

Parallel simulation is an emerging technology that enables the execution of large-scale simulation model in a shorter timeframe compared to its sequential counterpart. Many of the existing parallel simulation protocols are developed with the assumption that the underlying parallel computing platform is dedicated and thus most do not consider the factor of variation in system load of the computing platform due to interruption from external workload.

The increasing popularity of large-scale cluster of workstations as the execution platform for parallel simulation requires a new approach in the design of parallel simulation protocols. 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 cluster can fluctuate widely due to the presence of jobs from other users.

In [6, 7], we have reported our initial effort in designing a dynamic load-balancing (DLB) algorithm for the Bulk Synchronous Parallel Time Warp (BSP-TW) with external load-management capability. The external load-management module described in [7] uses the approach of evicting simulation workload from a processing node whenever the system load of the processing node exceeds a load threshold parameter. The drawback of this approach is that the performance of the system deteriorates rapidly in the presence of external workload on multiple nodes of the cluster of workstations. It is also difficult to determine the optimal values for the load threshold parameter.

In this paper, we present a new approach to external workload management using time slicing. Our experimental results show that the new approach is able to maintain high performance in the presence of external workload on multiple nodes of the cluster of workstations without the need to use the load threshold parameter.

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 BSP-TW DLB_{ccle} algorithm proposed in [7] is described. We then describe the new BSP-TW DLB_{ccls} algorithm which uses a time slicing approach to external workload management in section 4. Section 5 presents experimental results comparing the new DLB algorithm with the existing one on a manufacturing simulation model. Some related work are described in section 6. Section 7 summarizes the paper and outlines future research directions.

2 BSP TIME WARP

The BSP model first proposed in [9] is designed to be 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 [8] shown in Figure 1 is designed to be an efficient realization of an optimistic synchronization protocol ([3], [4]) 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.

The global virtual time (GVT) measures the progress of a simulation run. An estimate of GVT is computed after every

Proceedings 17th European Simulation Multiconference
(c) SCS Europe BVBA, 2003 ISBN 3-936150-25-7

```
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.

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

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

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.

From the BSP cost model, 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 MANAGING EXTERNAL WORKLOAD BY EVICTING PROCESSORS

The BSP-TW DLB$_{ccl}$ algorithm first described in [6] 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. In [7], we proposed an extension to the BSP-TW DLB$_{ccl}$ algorithm, referred to as BSP-TW DLB$_{ccle}$ algorithm, to allow external workload management.

```
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 2. Algorithm for BSP-TW DLB$_{ccle}$.

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

Figure 2 shows the pseudo-code for the 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. The pseudo-code for the BSP-TW DLB$_{ccl}$ algorithm is not shown here as it is essentially BSP-TW DLB$_{ccle}$ without module D0.

At each migration point, one of the four modules will be activated based on factors such as the amount of external workload, computation imbalance and 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 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 [6].

The BSP-TW DLB$_{ccl}$ algorithm described in [6] 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 3.

The state variable $P_i.la$ is used to track the average system 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

```
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.la < θ/2 then
            set P_i as active;
        endif
    endfor
```

Figure 3. Algorithm for Balancing External Workload.

queue averaged over various periods of time. The one 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 system 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 it.

4 MANAGING EXTERNAL WORKLOAD BY TIME SLICING

Although the BSP-TW DLB$_{ccle}$ protocol does solve the problem of external workload interruption, it sacrifices the complete use of a processor whenever it is loaded with external workload, regardless of the amount of external workload in the processor. Also, the performance of BSP-TW DLB$_{ccle}$ depends largely on how θ is set. If the value of θ is set too low, many processors may be evicted due to the presence of very small external workload. If the value of θ is set too high, the BSP-TW DLB$_{ccle}$ algorithm may not react effectively to the presence of external workload.

In this section, we consider another approach to managing external workload by considering the available time slice for the BSP process on the heavily loaded processors, rather than leaving the processors completely out of the parallel computation.

4.1 Example of External Workload Management using Time Slicing

We first illustrate our approach using the example shown in Figure 4. The figure shows the computation workload of a superstep for eight processors. Processors P0 to P3 are each loaded with two external workloads, indicated by the

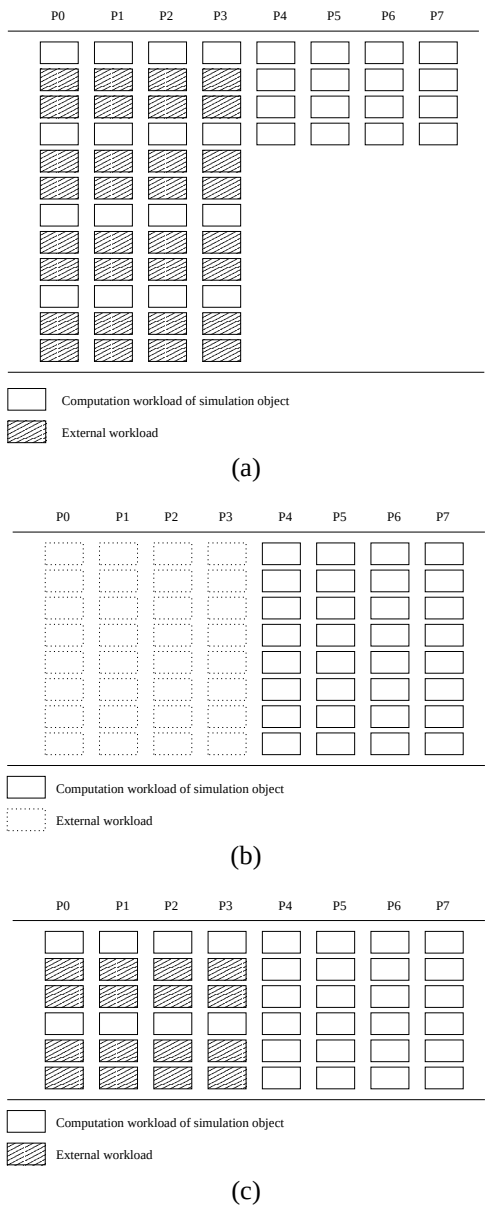

Figure 4. An Example of External Workload Management using Time Slicing.

shaded boxes. The computation workload of simulation objects on all the eight processors in the superstep are the same, as shown by the white boxes. Each white box can be considered the computation workload of a simulation object.

Due to the presence of external workload, the superstep on processors P0 to P3 takes three times the amount of time to complete, as compared to those on processors P4 to P7. We can also say that the simulation workload is only given one-third slice of the CPU processing time. If we assume that each box (white or shaded) consumes one unit of CPU processing time, the superstep takes 12 units of CPU processing time.

Figure 4b shows the workload configuration using the BSP-TW DLB$_{ccle}$ algorithm. All the simulation objects are evicted from the four loaded processors and distributed to processors P4 to P7. The resulting configuration is such that

processors P0 to P3 each can complete the superstep with minimum delay while processors P4 to P7 now have twice the amount of workload to process. The CPU processing time for this superstep is reduced to 8 time units.

Another approach to managing the workload is to consider the fact that the BSP workload on the heavily loaded processors still have access to one-third slice of the CPU processing time. We can migrate parts of the simulation objects out of these processors so that the overall workload for all processors (taking into account the external workload) after the migration is still balanced.

Figure 4c shows an example of how this is done. Two simulation objects are migrated out of each processor loaded with external workload. The resulting workload configuration is balanced across all processors. The superstep now requires only 6 units of CPU processing time.

The reason for the improvement over that using BSP-TW DLB$_{ccle}$ is due to the use of the remaining one-third slice of CPU processing time on those heavily loaded processors to process part of the simulation objects' workload. The increased in workload of those processors not affected by external workload is reduced compared to that using BSP-TW DLB$_{ccle}$.

4.2 BSP-TW DLB$_{ccls}$ Algorithm

We now describe the BSP-TW DLB$_{ccls}$ algorithm that provides an alternative solution to managing external workload by considering the allocated CPU time slice for the computation workload in each processor in the system. The outline of the BSP-TW DLB$_{ccls}$ algorithm is essentially the same as the BSP-TW DLB$_{ccl}$ algorithm. Unlike the BSP-TW DLB$_{ccle}$ algorithm which adds another module to the BSP-TW algorithm, the BSP-TW DLB$_{ccls}$ algorithm works within the `balance_computation()` module. Also, it should be noted that the BSP-TW DLB$_{ccls}$ algorithm does not require the use of the processor load threshold parameter θ.

Before we describe the modification to the module for balancing computation workload, we first need to resolve the condition for detecting imbalance in computation workload. For example, we would want to consider the workload configuration in Figure 4a as unbalanced while the configuration in Figure 4c as well-balanced. As the time taken to complete a superstep in each processor is computed by summing up the time taken for executing each event in the superstep, the `balance_computation()` module will instead consider the configuration in Figure 4a as well-balanced while the configuration in Figure 4c as unbalanced.

To resolve this problem, we can make use of the additional knowledge of the average system load of each processor ($P_i.la$) to work out a better approximation of the workload on each processor. We first scale the computation workload of each processor ($P_i.wl$) by its corresponding average system load as follows:

$$P_i.wl := P_i.wl * P_i.la\,. \tag{2}$$

balance_computation()
  while $WB > \epsilon$ **do**
    let P_{max} be the processor with the max. computation workload;
    let P_{min} be the processor that yield the min. average workload;
      when paired with P_{max}
    $x := \dfrac{P_{max}.wl * P_{max}.la - P_{min}.wl * P_{min}.la}{P_{max}.la + P_{min}.la}$
    computation_migrate(x, P_{max}, P_{min});
    $P_{max}.wl := P_{max}.wl$ - x;
    $P_{min}.wl := P_{max}.wl$
    compute WB;
  endwhile
  return *flag*;

Figure 5. Algorithm for Determining Amount of Computation Workload to Migrate taking into account System Workload. ϵ is the Load-imbalance Threshold Parameter.

Note that for those processors with average system load less than 1.0, $P_i.la$ will be set to 1.0.

The calculation of the computation imbalance, WB, of the system is shown in equation (3).

$$WB = \frac{\max(P_i.wl) - \text{mean}(P_i.wl)}{\text{mean}(P_i.wl)}. \tag{3}$$

Using this formula, the load imbalance for the superstep shown in Figure 4a will be 0.5 while the superstep in Figure 4c will be treated as having perfectly balanced workload.

The BSP-TW DLB$_{ccls}$ algorithm has exactly the same structure as that of BSP-TW DLB$_{ccl}$. The difference lies in the `balance_computation()` module, which now needs to take into consideration the system load of those processors involved in the load transfer process.

Figure 5 shows the pseudo-code for the `balance_computation()` module. Note that the computation workload used in the module have all been scaled by the average system load of individual processors. The processor P_{min} is not taken to be the one with the lowest computation workload, but rather the processor that will yield the lowest average workload when selected to engage in the load transfer process with the processor P_{max} that has the heaviest computation workload.

Figure 6 shows an example to illustrate why the processor that has the lowest computation workload is not chosen to be processor P_{min}. The example shows three processors and their respective computation workload for a superstep. We see that processor P0 is loaded with one external workload and processor P1 is loaded with four external workload. Although processor P2 is free from any external workload, its overall computation workload is still higher than processor P1.

Suppose processor P1 is now chosen to engage in the load transfer process with processor P0. One unit of computation workload will be migrated from processor P0 to processor

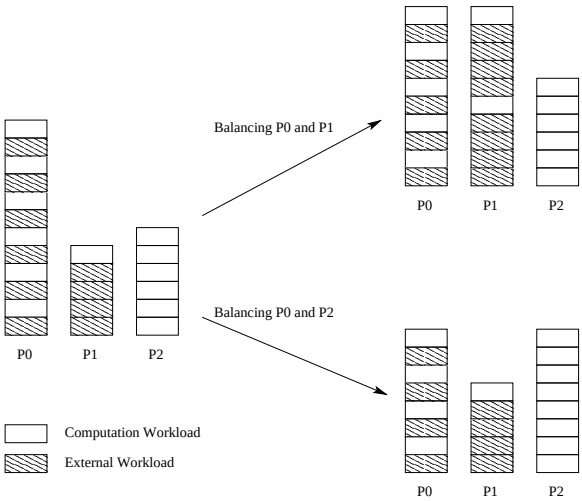

Figure 6. An Example to Illustrate the Selection of Processor P_{min}.

P1. This results in a net decrease of two units of computation workload in P0 and a corresponding five-unit increase in computation workload in P1. The overall improvement in the maximum computation workload is two units.

However, if processor P2 is chosen instead, two units of computation workload will be migrated from P0 to P2. This results in a net decrease of four units in computation workload in P0 and a corresponding two-unit increase in computation workload in P2. The overall improvement in the maximum computation workload in this case is four units. Although P2 is not the processor having the lowest computation workload, selecting it for the balancing process yields better performance compared to selecting P1, which has the lowest computation workload.

The presence of external workload on the individual processor and the scaling of computation workload for each processor requires some modifications to the formula used to compute the amount of workload to be transferred from processor P_{max} to P_{min}. The following formula computes the amount of computation workload, x, that needs to be migrated from processor P_{max} to processor P_{min} so that the resulting workload, y, on both processors after the migration is equal.

$$x = \frac{P_{max}.la(P_{max}.wl - P_{min}.wl)}{P_{max}.la + P_{min}.la} \quad (4)$$

$$y = P_{max}.wl - x \quad (5)$$

5 EXPERIMENTS WITH MANUFACTURING SIMULATION MODEL

In this section, we describe a set of experiments to compare the performance of BSP-TW DLB$_{ccl}$, BSP-TW DLB$_{ccls}$, and the BSP-TW DLB$_{ccle}$ algorithms.

5.1 Simulation Model

The experiments are carried out using a manufacturing simulation model similar to that used in [5] to study different

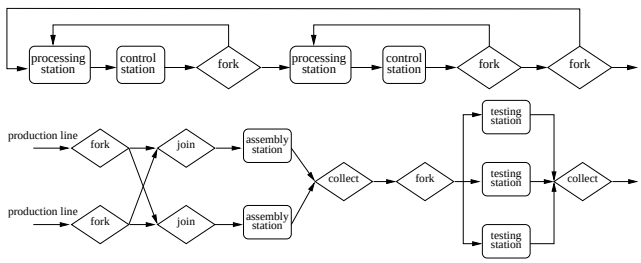

Figure 7. Layout of a Production Line and an Assembly and Test Facility.

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 7 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 processor load threshold parameter of θ=1.5 is used for the runs with BSP-TW DLB$_{ccle}$. 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. The simulation run length of all experiments is 10^4 time units. A block partition strategy is used to assign consecutive block of 25 simulation objects onto the same processor. The experiments are carried out by loading different number (K=1, 2, 4 and 6) of processors with different number (N=1, 2, 3, 4) of external workload. The external workload is introduced from the start of the simulation and lasts through the entire simulation duration.

5.2 Experimental Results

Table 1 shows the execution times using the three different protocols on the manufacturing model. The column under BSP-TW DLB$^*_{ccls}$ is executed using a modified version of BSP-TW DLB$_{ccls}$. This version uses a modified average system load for each processor, which is shown below:

$$P_i.la := (P_i.la)^2. \quad (6)$$

The modified system load of individual processor is then applied to the scaling of the computation workload in equation (2). This modification has no effect on those processors

		BSP-TW			
K	N	DLB$_{ccl}$	DLB$_{ccle}$	DLB$_{ccls}$	DLB$^*_{ccls}$
1	1	602.0	489.4	551.2	520.6
	2	789.8	517.8	559.3	558.0
	3	844.3	566.1	617.9	569.6
	4	993.3	592.2	787.8	606.1
2	1	718.9	596.8	672.9	613.5
	2	1170.5	634.1	705.1	615.1
	3	1318.3	635.9	732.7	651.2
	4	1667.6	693.5	958.8	690.7
4	1	826.2	944.0	768.6	747.8
	2	1502.5	954.7	981.9	749.0
	3	1697.1	987.7	1055.7	767.6
	4	2329.2	1021.6	1275.6	840.5
6	1	918.9	1462.8	934.0	867.8
	2	1704.6	1736.5	1350.0	1038.4
	3	2090.9	1843.2	1449.8	1124.8
	4	2971.3	1870.6	1816.9	1128.7

Table 1. Execution Times (sec.) using BSP-TW DLB$_{ccls}$ with N Number of Processors Loaded with K Number of External Workload.

with no external workload since $P_i.la$ will still be equal to 1.0. For those processors with average system load greater than 1.0, this change has the effect of encouraging the BSP-TW DLB$_{ccls}$ to migrate more simulation objects out of them. Similarly, it also discourages the load-balancing algorithm from migrating simulation objects back into them.

Table 1 shows that for the cases with $N = 1$, the performance of BSP-TW DLB$_{ccle}$ drops below BSP-TW DLB$_{ccl}$ as the number of processors loaded with external workload is increased to six. The BSP-TW DLB$_{ccle}$ algorithm is discarding six out of eight processors even though each of the six processors is only loaded with one external workload.

By not discarding completely those processors with external workload, the BSP-TW DLB$_{ccls}$ protocol is able to achieve better performance than the BSP-TW DLB$_{ccle}$ protocol for the cases with $N = 1$ and $K = 4$ and 6. For $K = 6$, the BSP-TW DLB$_{ccls}$ algorithm outperforms BSP-TW DLB$_{ccle}$ for all values of N.

However, as the number of processors loaded with external workload is reduced, the performance of BSP-TW DLB$_{ccls}$ drops below that of BSP-TW DLB$_{ccle}$. This drop in performance in BSP-TW DLB$_{ccls}$ can be attributed to two factors: 1) insufficient simulation objects are migrated out of those heavily loaded processors as the number of external workload on these processors is increased; and 2) side effects from the lookahead optimization module.

In order to verify the first hypothesis, we carried out the runs with BSP-TW DLB$^*_{ccls}$ to test if better performance can be achieved by encouraging more simulation objects to be migrated out of the heavily loaded processors. In a way, the squaring of the average system load of individual processor in equation (6) serves to exaggerate the load situation of those heavily loaded processors such that more simulation objects can be migrated out of them.

Table 1 shows that this approach does significantly improve the performance of the BSP-TW DLB$_{ccls}$ algorithm. For the cases with $K = 4$ and 6, the performance of BSP-TW DLB$^*_{ccls}$ drops gradually with increasing external workload. This shows that the dismal performance of BSP-TW DLB$_{ccls}$ is indeed due to insufficient simulation objects being migrated out of those heavily loaded processors.

However, for the runs with only one processor being loaded with external workload, the performance by either BSP-TW DLB$_{ccls}$ or BSP-TW DLB$^*_{ccls}$ is still slightly worse than that using BSP-TW DLB$_{ccle}$. This performance drop can be attributed to the side effect of lookahead optimization.

Figure 8 shows a breakdown of the computation workload as well as the number of simulation objects on each processor for a run executed using BSP-TW DLB$^*_{ccls}$. In this run, processor P1 is loaded with one external workload. We see that at superstep 1250, the balancing module is activated and the number of simulation objects on processor P1 drops from 307 to 170. Correspondingly, the computation workload for processor P1 decreases from a high level of 17.5 to 6.4.

However, at superstep 1500, an optimization of lookahead is carried out by the BSP-TW DLB$^*_{ccls}$ algorithm. This resulted in 20 simulation objects being migrated back into processor P1. The computation workload of processor P1 is increased to 11.5 in superstep 1750. At this point, the pattern repeats itself with the computation balancing module migrating simulation objects out of processor P1 and the lookahead optimization module migrating simulation objects back into processor P1. The main problem here is that the lookaheads optimization module uses a migration threshold η=0.5 which allows up to 50% of the simulation objects to be migrated during each round of lookahead optimization and this tends to disrupt the load-balance achieved in the computation balancing process.

A possible solution to resolving this issue might be to use a value of η smaller than 0.5. While this will reduce the amount of simulation workload that can be migrated back into the heavily loaded processors, it will also slow down the lookahead optimization process on other processors, causing the performance to drop. An effective solution might require the value of η to be set differently for different processors with different load configurations. Further work will need to be carried out to explore this possibility.

6 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. In this paper, the metrics used are the computation workload together with the average system load of the individual nodes in the cluster of workstations.

In the study reported in [1], Carothers and Fujimoto presented an approach for background execution of Time Warp. The scheme allows a Time Warp system to execute in background and consume unused CPU cycles across a collection

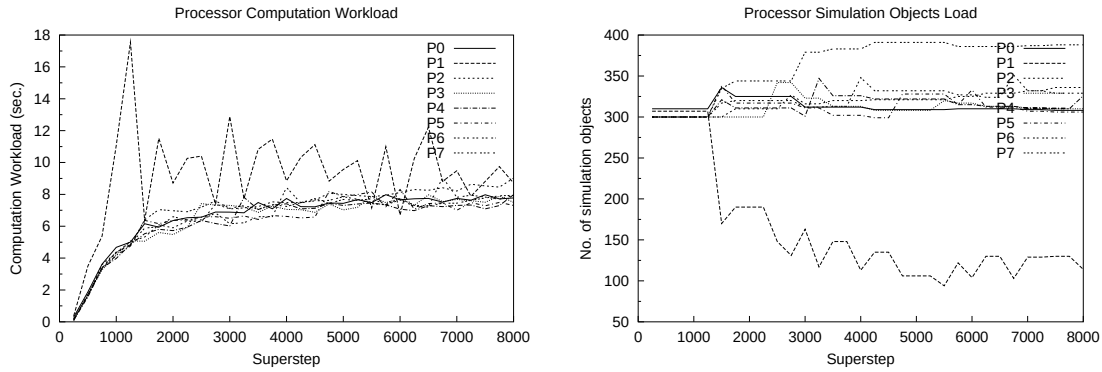

Figure 8. Processors Computation Workload and Number of Simulation Objects using BSP-TW DLB$^*_{ccls}$.

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. A personal communication service network model is used in this study. The experimental results showed an improvement of up to 45% in the presence of external workload.

Glazer and Tropper also described a metric based on time slices [2]. They defined a time slice to be a metric proportional to the ratio of the amount of computation time required by a process over the advance of its simulation time. They presented speedup improvement ranging from 12% to 49% using this approach to balance simulation workload for three different simulation models running on a simulation multiprocessor environment. However, their experiments do not take into consideration interruption from external workload.

7 CONCLUSION

In this paper, we have described a new time slicing approach to external workload management for the BSP-TW parallel simulation protocol. Our experimental results comparing the performance of the BSP-TW DLB$_{ccl}$, BSP-TW DLB$_{ccls}$ and BSP-TW DLB$_{ccle}$ show that BSP-TW DLB$_{ccls}$ protocol is able to achieve better performance over BSP-TW DLB$_{ccle}$ when a high proportion of processors in the BSP-TW computation are burdened with external workload. However, the performance of BSP-TW DLB$_{ccls}$ drops rapidly with increasing system workload on those heavily loaded processors. By amplifying the processor system workload to exaggerate the load-imbalance of the system, we show that the BSP-TW DLB$_{ccls}$ protocol can indeed achieve significant performance improvement over both the BSP-TW DLB$_{ccl}$ and BSP-TW DLB$_{ccle}$ protocols.

ACKNOWLEDGEMENT

This work was carried out when the author was with the Oxford University Computing Laboratory. The work was supported by a postgraduate scholarship from the Singapore Institute of Manufacturing Technology (SIMTech).

REFERENCES

[1] C.D. Carothers and R.M. Fujimoto. Background Execution of Time Warp Programs. In *Proceedings of the 10th Workshop on Parallel and Distributed Simulation (PADS'96)*, pages 12–19, Philadelphia, Pennsylvania, USA, May 1996.

[2] D.W. Glazer and C. Troper. On Process Migration and Load Balancing in Time Warp. *IEEE Transactions on Parallel and Distributed Systems*, 4(3):318–327, 1993.

[3] D. Jefferson. Virtual Time. In *ACM TOPLAS*, volume 7, pages 404–425, 1985.

[4] D. Jefferson and H. Sowizral. Fast Concurrent Simulation Using Time Warp Mechanism. In *Distributed Simulation 1995*, pages 63–69, La Jolla, California, USA, 1985. SCS-The Society for Computer Simulation, Simulation Councils, Inc.

[5] C.-C. Lim, Y-H. Low, W. Cai, W.-J. Hsu, S.-Y. Huang, and S.J. Turner. An Empirical Comparison of Runtime Systems for Conservative Parallel Simulation. In *2nd Workshop on Runtime Systems for Parallel Programming (RTSPP 1998)*, pages 123–134, Orlando, Florida, USA, 30 March 1998.

[6] M.Y.H. Low. Dynamic Load-Balancing for BSP Time Warp. In *Proceedings of the 35th Annual Simulation Symposium*, pages 267–274, San Diego, California, USA, 14-18 April 2002.

[7] M.Y.H. Low. Managing External Workload with BSP Time Warp. In *Proceedings of the 2002 Winter Simulation Conference*, pages 704–711, San Diego, California, USA, 8-11 December 2002.

[8] M. Marín. *Discrete-Event Simulation on the Bulk-Synchronous Parallel Model*. PhD thesis, Oxford University, November 1998.

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