Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

EXPERIMENTS IN LOAD MIGRATION AND DYNAMIC LOAD BALANCING IN SPEEDES

Linda F. Wilson and Wei Shen

Thayer School of Engineering
Dartmouth College
Hanover, NH 03755–8000 U.S.A.

ABSTRACT

It is well known that the performance of a parallel discrete-event simulation (PDES) depends on the allocation of the workload to processors. In particular, poor performance may be the result of an imbalance of the workload on the processors. In earlier work, we examined automated load balancing techniques that statically allocated the workload based on prior run-time data. However, a good initial distribution of the workload may result in a load imbalance if the characteristics of the simulation change over time. Furthermore, some of the processors may gain external workloads at some point in time. Thus, there is a need for dynamic load balancing methods that can adapt to changes in the simulation or the system. In this paper, we discuss our experiments in dynamic load (object) migration and load balancing using the SPEEDES simulation framework.

1 INTRODUCTION

The primary goal of parallel discrete-event simulation (PDES) is the fast execution of large discrete-event simulations. In particular, PDES attempts to gain speedup by distributing the simulation on multiple processors executing in parallel. However, the distribution of the workload has a significant impact on PDES performance. If one processor is heavily-loaded while others are lightly-loaded or idle, the performance may be improved by shifting some of the workload to a less-loaded processor.

In a typical PDES, components of the system under examination are mapped into logical processes (LPs) that can execute in parallel. The LPs are distributed among the physical processors, and communication between LPs is accomplished by passing messages. There is a conflict between distribution for load balance and distribution for low communication overhead. If LPs are distributed among the processors such that interprocessor communication is minimized, some processors may sit waiting for something to do while others are overloaded with work. At the other extreme, a "perfectly-balanced" workload may induce high communication costs. Thus, load-balancing strategies must find a compromise between distributing the work evenly and minimizing communication costs.

Researchers have examined various load-balancing approaches for PDES (e.g. Nandy and Loucks 1992, Reiher and Jefferson 1990, Schlagenhaft, et al 1995). In many cases, load-balancing algorithms are designed for a particular class of simulation problems (e.g. digital circuit simulation). In other cases, application of the load-balancing algorithm requires significant modification of the user's simulation code. Our work focuses on load balancing for general-purpose simulations such that little modification of the user's code is required.

In earlier work (Wilson and Nicol 1995, 1996), we described our experiences in developing an automated load balancing strategy for the SPEEDES simulation environment. In addition, we discussed experiments with three different mapping algorithms used in conjunction with our automated scheme. That work concentrated on static methods for allocating the workload to the processors at the beginning of the execution.

In this paper, we discuss our initial work in dynamic load balancing using object (load) migration in SPEEDES. Section 2 introduces SPEEDES while Section 3 discusses dynamic load balancing and object migration. Section 4 presents three dynamic load balancing algorithms developed for this work. Section 5 presents experimental results from executing the three algorithms on a network of SGI workstations. Finally, Section 6 presents our conclusions and directions for future work. Complete details of the experiments and results summarized in this paper may be found in Shen (1998).

2 SPEEDES

SPEEDES (Synchronous Parallel Environment for Emulation and Discrete-Event Simulation) is an object-oriented simulation environment that was developed at the Jet Propulsion Laboratory (Steinman 1992). Designed for distributed simulation, SPEEDES supports multiple synchronization strategies (including Time Warp, Breathing Time Buckets, and Breathing Time Warp) that can be selected by the user at runtime. In addition, SPEEDES provides a sequential simulation mode (with most of the parallel overhead removed) so that a particular simulation model can be executed serially or in parallel.

In SPEEDES, the user is completely responsible for the mapping of the simulation objects to the processors. While SPEEDES gives the user freedom to choose an appropriate mapping, it is quite likely that the user does not know *a priori* how to choose a good allocation of objects to processors. Most variations of the mapping problem are computationally intractable, so optimal mappings are extremely difficult to obtain. Furthermore, many users and potential users of PDES would prefer to let "the system" make such decisions, especially if the resulting performance is "good enough". Finally, the characteristics of the simulation may change through time such that a static allocation does not yield good performance. Thus, this work examines automated dynamic load balancing through object migration in SPEEDES.

3 LOAD BALANCING AND LOAD MIGRATION

Dynamic load balancing policies use run-time system state information to make decisions to move work from one host to another during execution. This approach can react to changing simulation or system characteristics and rebalance the workload if it fluctuates through time. However, tradeoffs are involved since dynamic approaches can create additional overhead from collecting system state data, analyzing the data, and making load balancing decisions.

Load migration, sometimes called *process migration* or *object migration*, is the mechanism used to move workload from one processor to another. While load balancing is concerned primarily with balancing the workload evenly among the processors, load migration is concerned with transferring load, and load balancing can be an objective of load migration.

Load balancing and load migration attempt to increase the performance of parallel and distributed simulations. However, due to variations in systems, algorithms that work well in one distributed system may not obtain the same performance in another system. The dynamic load migration algorithms developed in this work attempt to improve the performance of SPEEDES simulations running on a network of workstations.

4 RCL DYNAMIC LOAD MIGRATION POLICIES

In this work, we developed three policies for dynamic load migration in SPEEDES. The algorithms are referred to as random (R), communication-based (C), and load-based (L) policies, or simply RCL policies.

Complexity is an important aspect of a load migration policy, and making reasonable assumptions can simplify the complexity of algorithms. The RCL dynamic load migration policies were designed using the following assumptions.

- The distributed system is heterogeneous.
- The hosts are non-dedicated; interactive users can use the machines at any time.
- The number of hosts is limited (e.g. 4 to 8).
- The fluctuation of the network bandwith is negligible.
- Communication delays are constant.
- The migration cost of a piece of workload is proportional to the physical size of the migrating workload.

RCL policies use a two-level decision-making process that combines centralized and decentralized approaches. A central coordinator (processor) makes global migration decisions and each sending processor is responsible for selecting the load to be migrated.

At Level I, a central coordinator is chosen to be the decision maker among all the hosts. During a load-migration step, every host stops processing work while the central coordinator collects load information. The coordinator then analyzes the overall system load based on a pre-migration procedure which computes the CPU utilization and load distribution in the system, and makes a decision on whether a load migration is necessary. If the system load status meets the criteria for migration, the central coordinator starts redistributing the remaining load among the hosts based on the processing rate of each processor. The origin and destination of the migrated loads are determined and the corresponding sending hosts and receiving hosts are notified by the central coordinator. On the other hand, if load migration is not necessary, the central coordinator will broadcast this information to all other hosts. In this case, Level II actions are ignored and every host resumes processing its local events.

At Level II, once the sending hosts receive load migration notices from the central coordinator, they start selecting the local load to be transferred based on one of the RCL algorithms. After loads are selected on all sending hosts, load migration takes place. Load migration steps between a pair of hosts include handshaking, packaging and

sending data by the sender, and receiving and unpackaging data by the receiver. At the end of each load migration step, the receiving hosts broadcast and inform all other hosts about the new locations of the migrated loads, and normal processing resumes on each host. The following subsections provide some of the details on the actions taken at each level.

4.1 Level I Actions

At the beginning of a load migration step, all hosts stop processing their local loads and each host gets information regarding the local load situation. The local load status information includes the following factors:

- Pending load: the total amount of pending load.
- CPU utilization: local processor CPU utilization, i.e., how much time does the processor spend on SPEEDES applications.
- Processed load: the amount of load processed since the last load migration.

Once the central coordinator collects load information from all the hosts, it starts analyzing the overall load condition for the whole system. Specifically, it computes the following metrics: variance of CPU usage, total amount of pending load, and variance of pending load distribution. These metrics will be used in the decision-making process. The load in SPEEDES is defined to be the amount of processing time required to process a certain number of events.

- *Variance of CPU usage*: The variance of CPU usage can be used as an indication of how the system CPUs are used. For example, if some hosts have several applications running in addition to the SPEEDES application while others are idle or lightly-loaded, this value can be large. The larger the value, the less efficient the use of system resources.
- *Total amount of pending load*: The total amount of pending load in the system is the sum of the pending loads on each host. If the amount is too small, load migration may not increase the performance gain since the migration overhead may easily outweigh the benefit of balancing a small load.
- *Variance of pending load*: The variance of pending load is computed from the pending loads of each host. A large variance indicates a load imbalance among the hosts in the system. This is used as a secondary criterion when the variance of CPU utilization is below a threshold value.

These metrics are used in the pre-migration procedure to help determine whether a load migration is necessary. The first step involves the comparison between the total amount of pending load and its specified threshold value. If the value is less than its threshold value, load migration is disabled for this step. If the value is greater, the variance of CPU usage is calculated. If the variance of CPU usage variance is greater than its corresponding threshold, load migration is enabled; otherwise, the variance of pending load distribution is calculated. If the variance of pending load distribution is greater than its threshold, load migration is enabled, otherwise, load migration is disabled for this step. At the end of this phase, if load migration is not enabled, the central coordinator will inform all other hosts to continue processing their local loads.

Once load migration is enabled, the central coordinator informs the other hosts that a load migration is pending. It then decides which hosts will be senders and which ones will be receivers. Based on the *step processing rate* (i.e., the amount of load a host has processed locally divided by the total amount of load processed among all processors since the last load migration), the central coordinator redistributes the total amount of pending load. For example, if a host has processed 10 percent of the total amount of load since the last migration step, it will get 10 percent of the total pending load after the current migration step. Consequently, those hosts which have processed more load will get a bigger share of the pending load. If no extra load is created and the processing rate of each host stays constant, all hosts will finish their local loads at roughly the same time.

The central coordinator notifies each host what its pending action is: sending, receiving, or no-action. Sending hosts will be informed of the amount of load to be migrated and the IDs of their corresponding receiving hosts, while receiving hosts will be notified about their corresponding sending hosts and the amount of load to be received.

Finally, the central coordinator completes its duty and each host prepares for load migration if migration is enabled. That is, Level II actions will be required if load migration is enabled. Otherwise, each host resumes processing its local load.

4.2 Level II Actions

Level II actions involve every host in the distributed system. A certain amount of load is passed from one host to another. Main actions involve selecting the load, packaging the load, initiating migration, migrating the load, unpackaging the load, and updating global information.

Each logical process (LP) in SPEEDES consists of a number of simulation objects and an event queue. Events are inserted into an object's event queue in non-decreasing timestamp order. The load is defined as the total work

for processing the pending events in the event queue. Since SPEEDES is modified for this work to enable load migration, each type of event is associated with a complexity factor which ranges from 1 to 10. The user must define the complexity factor for each user-defined event. The closer the complexity factor is to 10, the more time is required to process that event. The pending load of an object is computed based on the number of pending events of each type and the complexity factor for each event. For example, suppose a simulation object has two Type I events and three Type II events in its event queue. If the complexity factor is 4 for the Type I event and 8 for the Type II event, the total pending load for this object is (2*4) + (3*8), which equals 32.

Load migration is achieved by moving simulation objects from one host to another. Three policies are used to select local objects (loads). They are the random, communication-based, and load-based algorithms. Each sending host can use any of these three policies to select the objects to be migrated.

- *Random*: This approach is simple and easy to implement. Given an amount of load to be migrated, the sending host selects local objects randomly until the combined load of the objects fulfills the requested amount. Advantages include easy implementation, low overhead, and short selection times. However, this approach may create extra residual dependency since a large number of objects may be selected and migrated.
- *Communication-based*: In a SPEEDES simulation, objects on each LP can communicate with objects on other LPs to exchange data or schedule new events. The communication-based algorithm attempts to move those objects which communicate frequently with the receiving host. During a simulation, each LP keeps track of the number of communications between each of its local objects and any of the other LPs. When migration is enabled, the sending host chooses the local objects that communicate most frequently with the receiving hosts. This approach can potentially reduce the communication between two LPs. However, the policy creates extra overhead during the load migration process. First of all, each host must keep a communication lookup table to keep track of the frequency of communication between each local object and all other hosts, and the table must be updated constantly. Secondly, in order to select the most appropriate objects, a sorting or searching algorithm must be implemented and will create overhead if the number of local objects is large.
- *Load-based*: The load-based policy attempts to minimize the number of selected migrating objects. During a migration step, simulation objects are sorted based on their pending loads (computed based on the number of each type of event and the complexity factor for the event). The object with the largest pending load is selected first. The implementation is similar to the random approach with the addition of a sorting routine. Because fewer objects are selected compared to the other two approaches, this approach can reduce the amount of selection time and the synchronization overhead.

Before a simulation starts, the user chooses which object selection policy will be used. At the end of the object selection step, each sending host has a list of objects to be migrated. Details of the migration process can be found in Shen (1998).

4.3 Implementation

The RCL dynamic load migration policies were implemented in the SPEEDES distributed simulation framework in an attempt to improve simulation performance. As discussed in Section 5, several experiments were conducted to determine whether certain parameter values, such as the migration interval, can potentially affect the execution time of a SPEEDES simulation.

While SPEEDES supports several synchronization strategies, our work uses the Breathing Time Warp (BTW) protocol (Steinman 1993), which combines features of the Time Warp and Breathing Time Buckets protocols. Each cycle in the Breathing Time Warp algorithm starts out using aggressive message-sending methods (Time Warp) but then makes a transition to risk-free message-sending methods (Breathing Time Buckets). After the Breathing Time Buckets stage, BTW computes the new global virtual time (GVT). At this point, the memory allocated for state-saving and rollback mechanisms is returned to the operating system.

In terms of object migration, there are two significant advantages to performing migration at the end of a GVT cycle. First of all, messages are flushed out of the system during the GVT calculation, so there is no danger that a rollback that affects a just-migrated object will occur. Secondly, only the variables and events associated with a simulation object need to be migrated; the data associated with rollbacks does not have to be migrated since the rollback queue is cleared at the end of a GVT cycle.

5 EXPERIMENTAL RESULTS

Several experiments were conducted using a queuing network application called Qnet. The purpose of the experiments was to determine whether the RCL load migration policies can reduce the total execution time of a SPEEDES application under different computing

environments, and if so, which policy or policies will produce the best performance. In addition, load migration statistics were recorded to give insight as to migration behavior in SPEEDES and provide some groundwork for future improvement.

The Qnet application was selected for the experiments because it is a simple application that is commonly used as a PDES benchmark. The Qnet simulation model consists of a certain number of server objects which provide some user-defined service to the customers arriving in a queue associated with each server.

The experiments were conducted in three different computing environments, where the external loads on the machines vary. In the first case (Experiment Set I), each host involved in SPEEDES is either idle or lightly-loaded. That is, the contention for the CPU is low. In the second case (Experiment Set II), one of the hosts is intentionally overloaded while the other hosts are lightly-loaded or idle. In the last case (Experiment Set III), each host is loaded to a different degree. For example, Host 1 has twice as much load as Host 0, while Host 2 has three times as much load as Host 0. The reason for choosing these environments is that these scenarios can represent the loads in the system at different times of day.

Within each computing environment, three separate groups of experiments were conducted. A particular parameter value was varied for each group of experiments. The first group of experiments investigated the relationship between the migration interval and the total execution time. The second group of experiments investigated the impact of the CPU utilization threshold values on the total execution time. The last group of experiments was needed to determine whether varying the load selection tolerance can change the total execution time.

Speedups calculated at the end of each experiment were used to determine how well each policy performed under various circumstances. For the first set of experiments, two speedups were computed. One of them was the speedup based on the sequential execution time on an idle host while the other one was the speedup compared to the regular (non-migrating) distributed execution. For the second and third sets of experiments, one additional speedup value was calculated by dividing the execution time of the sequential execution on an overloaded host by the parallel execution time with load migration.

The experiments were conducted on a network of SGI workstations connected by Ethernet. Specifically, we used 4 SGI O2 machines containing MIPS R5000 processors running at a clock rate of 180 MHz. Half of the machines had 128 MB of RAM while the other half had 64 MB of RAM.

As discussed in Shen (1998), the three experiment sets provided similar results. Due to space constraints, only the results from Experiment Set III will be presented here. Experiment Set III investigated how well the RCL dynamic load migration policies perform in a computing environment in which each host in the distributed system is loaded with a different amount of external workload.

5.1 Effect of the Migration Interval

First, we examined the effect of the migration interval on the execution time and corresponding speedup obtained by each of the RCL policies. Figure 1 shows that the execution times increase as the migration interval increases. Thus, frequent migrations yield better performance. Notice that the load-based policy yields the fastest execution times while the communication-based policy has the longest execution times. Table 1 shows the corresponding speedups. Notice that load migration makes a significant impact compared to the parallel simulation without load migration. Despite the initial load imbalance, the distributed execution with load migration is faster than the serial execution on the lightly-loaded processor as long as the migration interval is small.

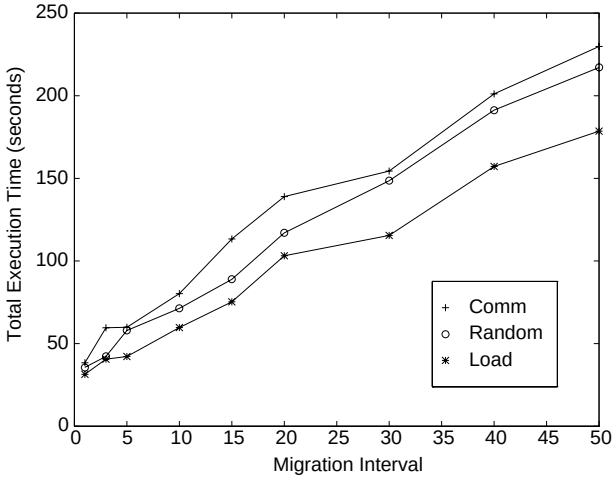

Figure 1: Execution Time vs. Migration Interval

This group of experiments showed that the migration interval has a great influence on the execution times. The larger the migration interval, the longer the execution time and the smaller the speedup. Among the three policies, the best performance is achieved using the load-based policy. The highest speedups are obtained when the migration interval is less than 5, especially when compared to the execution time of the parallel version without load migration.

5.2 Effect of the CPU Variance Threshold

The second group of experiments examined whether varying the threshold value of CPU variance had any influence on

Table 1: Speedup vs. Migration Interval

	Lightly Loaded Node			Heavily Loaded Node			Parallel Without Load Migration		
Interval	*R*	*C*	*L*	*R*	*C*	*L*	*R*	*C*	*L*
1	1.75	1.63	1.99	12.88	11.94	14.59	125.25	116.09	141.95
3	1.48	1.05	1.54	10.87	7.70	11.30	105.68	74.89	109.84
5	1.08	1.04	1.48	7.92	7.66	10.88	77.04	74.47	105.83
10	0.88	0.78	1.05	6.44	5.72	7.69	62.59	55.60	74.75
15	0.70	0.52	0.83	5.16	3.82	6.09	50.20	37.13	59.28
20	0.65	0.45	0.61	4.81	3.30	4.45	46.79	32.10	43.26
30	0.42	0.40	0.54	3.09	2.97	3.98	30.05	28.90	38.65
40	0.33	0.31	0.40	2.40	2.28	2.92	23.33	22.18	28.39
50	0.29	0.27	0.35	2.11	1.99	2.57	20.55	19.42	24.99

the execution times or corresponding speedups. As shown in Figure 2, changes in the CPU variance threshold had no significant impact on the execution time. Table 2 confirms that the initial load imbalance has a significant affect on the speedup. Notice that the execution times are usually smallest when the load-based policy is used.

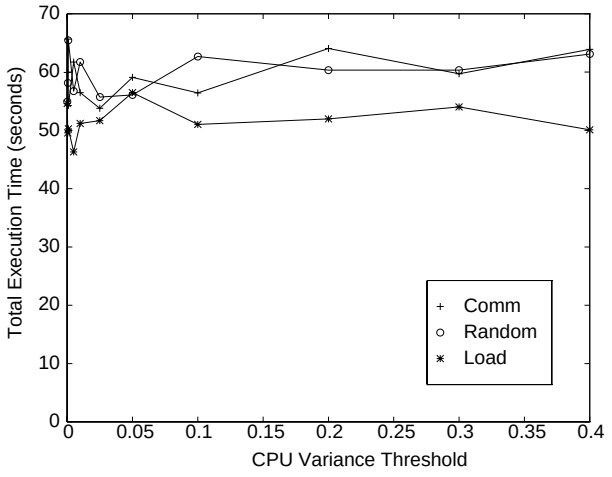

Figure 2: Execution Time vs. CPU Variance Threshold

This group of experiments showed that the threshold values of CPU variance do not change the execution time significantly. Though the execution times among the three policies are close, the performance of the load-based policy is sometimes better than the other two.

5.3 Effect of the Load Selection Tolerance Threshold

The third group of experiments investigated whether the threshold values of load selection tolerance have any influence on the execution time in a computing environment in which hosts are loaded by different amounts of workload. Figure 3 shows that the load selection tolerance has no significant effect on the execution time. However, the results in Figure 3 and Table 3 confirm that the load-based policy yields faster execution times.

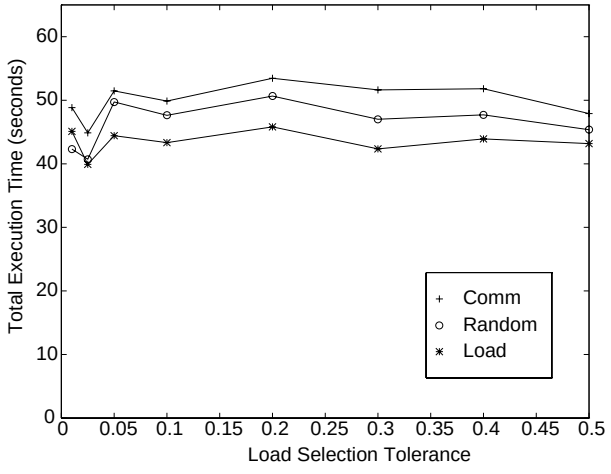

Figure 3: Execution Time vs. Load Selection Tolerance

This section described the three sets of experiments that were used to test the effectiveness of the RCL dynamic load migration policies under various system load conditions. The experimental results have shown that the migration frequency can affect the execution time of a simulation significantly; the larger the frequency, the better the speedup. On the other hand, the threshold value of CPU utilization variance and the threshold value of the load selection tolerance play a less-important role. The results indicate that the performance of SPEEDES can be greatly improved with load migration, especially in a computing environment in which some hosts are more loaded than the others. Among the three policies, the load-based policy consistently yields the shortest execution time. Overall, the benefit of load migration far exceeds the migration overhead.

Table 2: Speedup vs. CPU Variance Threshold

	Lightly Loaded Node			Heavily Loaded Node			Parallel Without Load Migration		
CPU Var.	*R*	*C*	*L*	*R*	*C*	*L*	*R*	*C*	*L*
0.0001	1.25	1.04	1.14	9.21	7.66	8.40	89.51	74.48	81.71
0.0005	1.08	0.95	1.26	7.89	7.01	9.26	76.81	67.98	90.04
0.001	0.96	1.15	1.24	7.01	8.46	9.13	68.18	82.27	88.89
0.005	1.10	1.01	1.35	8.09	7.44	9.91	78.68	72.29	96.37
0.01	1.19	1.11	1.22	8.76	8.13	8.97	85.17	79.01	87.18
0.025	1.12	1.16	1.21	8.24	8.53	8.88	80.11	82.95	86.37
0.05	1.21	1.06	1.11	8.92	7.76	8.14	86.69	75.51	79.01
0.1	0.99	1.11	1.22	7.32	8.13	8.99	71.20	79.08	87.45
0.2	1.04	0.98	1.20	8.81	7.17	8.83	82.79	69.68	85.87
0.3	1.04	1.05	1.16	7.61	7.69	8.50	73.85	74.74	82.60
0.4	1.28	0.98	1.25	9.44	7.18	9.16	91.79	69.84	89.09

Table 3: Speedup vs. Load Selection Tolerance

	Lightly Loaded Node			Heavily Loaded Node			Parallel Without Load Migration		
Load Tol.	*R*	*C*	*L*	*R*	*C*	*L*	*R*	*C*	*L*
0.01	1.48	1.28	1.39	10.85	9.40	10.18	105.45	91.38	98.95
0.025	1.29	1.39	1.57	9.46	10.23	11.50	91.97	99.48	111.79
0.05	1.26	1.46	1.57	9.23	10.73	11.56	89.74	104.29	112.38
0.1	1.31	1.25	1.44	9.64	9.20	10.59	93.72	89.49	102.99
0.2	1.23	1.17	1.65	9.06	8.59	12.14	88.11	83.48	117.94
0.3	1.33	1.21	1.48	9.76	8.89	10.84	94.93	86.45	105.40
0.4	1.31	1.21	1.42	9.62	8.86	10.45	104.01	86.14	101.61
0.5	1.38	1.31	1.68	10.11	9.58	12.34	98.39	93.18	119.93

6 CONCLUSIONS

The goals of this project were to enable load migration in SPEEDES and determine whether load migration can improve the performance of SPEEDES. The RCL dynamic load migration policies were designed for the SPEEDES distributed simulation environment. Extensive experiments using the Qnet simulation were conducted to investigate the effectiveness of these three policies. The policies were tested in three sets of experiments in which each experiment had a different distribution of external loads in the system.

The simulation results have shown that all the policies can achieve speedups under various system load conditions. The results are summarized as follows:

- For all three policies, the execution time is shortest when the migration interval is small and close to 1.
- The migration frequency can affect the execution times significantly.
- Compared to parallel execution without load migration, significant speedups are achieved with load migration.
- The load-based policy consistently gives the shortest execution time.

The data and observations indicate that the benefits from load migration far exceed the communication overhead and migration cost incurred by the load migration process. The results confirm that load migration can indeed improve the performance of SPEEDES.

Future work is needed to further improve the robustness of dynamic load migration policies in SPEEDES. In particular, additional modifications need to be made to SPEEDES to allow the dynamic migration of complex objects. SPEEDES does not currently allow the dynamic creation and destruction of simulation objects, so we had to create “dummy” objects (with no initial workload) in which to migrate active simulation objects.

The RCL load migration policies should be tested on applications with more-complicated objects and events.

Preliminary testing indicates that the communication overhead incurred by moving larger objects between processors is outweighed by the advantages of load balancing. However, we would like to confirm this using larger "real-world" simulations.

Finally, the algorithms should be tested on a larger, heterogeneous network of workstations. As stated earlier, these algorithms were designed for small numbers of hosts, but we plan to test them on a network of up to 20 workstations to determine their scalability.

REFERENCES

Nandy, B., and W. Loucks. 1992. An Algorithm for Partitioning and Mapping Conservative Parallel Simulation onto Multicomputers. *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS '92)*, pp. 139–146.

Reiher, P. L., and D. Jefferson. 1990. Dynamic Load Management in the Time Warp Operating System. *Transactions of the Society for Computer Simulation*, 7(2):91–120.

Schlagenhaft, R., M. Ruhwandl, C. Sporrer, and H. Bauer. 1995. Dynamic Load Balancing of a Multi-Cluster Simulator on a Network of Workstations. *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95)*, pp. 175–180.

Shen, W. 1998. Load Migration Policies in Distributed Simulation Using SPEEDES, MS Thesis, Dartmouth College, Hanover, New Hampshire.

Steinman, J. 1992. SPEEDES: A Multiple-Synchronization Environment for Parallel Discrete-Event Simulation. *International Journal in Computer Simulation*, 2(3): 251–286.

Steinman, J. 1993. Breathing Time Warp. *Proceedings of the 1993 Workshop on Parallel and Distributed Simulation PADS'93*, pp. 109-118.

Wilson, L. F., and D. M. Nicol. 1995. Automated Load Balancing in SPEEDES. *Proceedings of the 1995 Winter Simulation Conference*, pp 590–596.

Wilson, L. F., and D. M. Nicol. 1996. Experiments in Automated Load Balancing. *Proceedings of the 10th Workshop on Parallel and Distributed Simulation (PADS'96)*, pp 4-11.

AUTHOR BIBLIOGRAPHIES

LINDA F. WILSON is the Clare Boothe Luce Assistant Professor of Engineering in the Thayer School of Engineering at Dartmouth College. She received her BS degree from Duke University and MSE and PhD degrees from the University of Texas at Austin.

WEI SHEN is an I/T Specialist at IBM in Waltham, Massachusetts. He received his BS degree from the University of Massachusetts at Amherst and his MS degree from Dartmouth College.