Load Balancing for Distributed Battlefield Simulations: Initial Results

David R. Pratt, Ph.D.
SAIC
Orlando, FL
prattda@saic.com

Amy E. Henninger, Ph.D.
Soar Technology, Inc.
Orlando, FL
amy@soartech.com

ABSTRACT

It is well known that the system performance of parallel discrete event simulations depends on the assignment of workload to processors. If one processor is heavily loaded while the others are lightly loaded or idle, the overall performance of the system may be improved by offloading some of the workload to a less loaded processor. Load migration is the means by which this workload is dynamically moved from one processor to another. Central to the successful implementation of load migration is the ability to effectively predict processor loading and to apply domain relevant heuristics to select the source, the entities, and their destination for migration.

The modified discrete-event simulation (DES) scheduling paradigm used by battlefield simulations impedes the system's ability to effectively predict processor loading. In this paradigm, the real-time or scaled real-time systems depend on the busy wait construct to cycle through a time-delay loop and fire an event at the correct time. Because these cycles are consumed during what would otherwise be idle time, the inefficiencies inherent in this type of polling are moot with respect to processing time at the application level. However, at the system level, where processor loading is normally ascertained, since all of the cycles are fully consumed by the application, there is no practical way of predicting the processor loading.

In this paper we will present a modification to the DES scheduler used in real-time systems such that it allows for the identification of trends in processor loading. Briefly, this modification will monitor the queue access request so it is possible to generate and analyze the pertinent data. Next, the paper will discuss how items are selected for offloading once the processor has been determined to be in the "overloaded" state. This involves implementing a number of application and system configuration heuristics as they relate to the entity-based simulation domain. Finally, the paper will present our sample implementation and the results of our analyses.

ABOUT THE AUTHORS

Dr. Pratt is currently the Vice President and Director for Technology for the Orlando Group of Science Applications International Corporation (SAIC), a Fortune 500 company. Dave's responsibilities include developing and fostering continued leading edge Information Technology (IT) and Modeling and Simulation (M&S) technologies. Supporting the Group headquarters in Orlando and offices in Tallahassee, eight other states and four overseas locations, he provides both strategic and tactical guidance in both technical and programmatic matters. Before joining SAIC, Dr. Pratt was the Technical Director for the JSIMS program. Formerly a Tenured Associate Professor of Computer Science at the Naval Postgraduate School and Adjunct Teaching Instructor at the University of Central Florida, he has an extensive academic background that includes over 50 publications and $5M of external academic research funding.

Dr. Amy Henninger is a Senior Scientist at Soar Technology, Inc., principal investigator for the Individual Combatant's Weapons Firing Algorithm for U.S. Army Soldiers Systems Command, and project manager for the phase II STTR on Emotional Cognitive Synthetic Forces for the U.S. Army Research Institute. Prior to joining Soar Technology, Dr. Henninger worked in the training and simulation community as a Staff Scientist at SAIC, a Research Assistant at the Institute for Simulation and Training, and a Research Fellow for the U.S. Army Research Institute at U.S. Army STRICOM. She holds BS degrees in Psychology, Industrial Engineering and Mathematics from Southern Illinois University; MS degrees in Engineering Management and Computer Engineering, from Florida Institute of Technology and University of Central Florida (UCF), respectively; and a Ph.D. in Computer Engineering from UCF. Dr. Henninger currently teaches as Adjunct Faculty for the Modeling and Simulation Graduate Program at UCF.

Load Balancing for Distributed Battlefield Simulations: Initial Results

David R. Pratt, Ph.D.
SAIC
Orlando, FL
prattda@saic.com

Amy E. Henninger, Ph.D.
Soar Technology, Inc.
Orlando, FL
amy@soartech.com

PROBLEM INTRODUCTION

This paper reports on the development and testing of a load-balancing model for distributed battlefield simulations. Load balancing is the act of optimizing the utilization of all processors in a parallel computing system. Because of the message-passing paradigm used in distributed battlefield simulation and the chaotic and busy nature of the communication and processing modes, dynamic load balancing is a prime candidate for improving system performance. That is, if one processor is heavily loaded while others are less loaded or idle, the system's overall performance may be improved by passing some of the workload from the heavily loaded processor to a less-loaded processor. Load-migration is the term used to describe the means used to move the workload from one processor to another. And, one of the aims of load migration can be to support load balancing.

There are a number of issues related to distributed-battlefield simulations that make it difficult to optimize load-migration. These include the scheduling paradigms used by discrete-event simulations and the difficulty in predicting battlefield behavior. The following two sub-sections present an introduction to load balancing, in general, and specifically, to load-balancing issues related to parallel discrete-event simulations (PDES) and distributed battlefield simulations. Those readers who are already familiar with basic load- balancing concepts and DES time advance mechanisms may proceed to the next major section without loss of critical information.

Load Balancing

Load balancing has been a research area in both the parallel computing and distributed simulation communities for a number of years. Traditionally, the evaluation of load balancing algorithms has been performed in test-beds simulating the physical system, the system's workload, and the algorithm being evaluated. This allows researcher to efficiently evaluate a number of strategies that would otherwise not be practical to implement in the physical system.

While, as seen in Table 1, different researchers may conceptually partition the load-balancing model differently, all load balancing models perform the same basic functions. First, a load-balancing algorithm must determine if there is a need for load migration. In accomplishing this, the algorithm must estimate the load for each processor, determine if a significant imbalance exists, and then determine if the potential net gain for the system is greater than the overhead incurred from the load balancing exercise. Next, the algorithm must determine which loads from which processors (source) should be migrated and where they should be migrated (destination). Finally, the workload is transferred from the source to the destination.

Source	Vaughan and O'Donovan (1998)	Wilson and Shen (1998)	Willebeek-Lamair and Reeves (1993)	Pratt and Henninger (2003)
Load Balancing Algorithm Components	Information policy Transfer policy	Level 1 (pending load, cpu utilization, processor load)	Processor Load Evaluation Load Balancing Profitability Determination	Determine load (current and projected)
	Selection policy Location policy	Level 2 (selecting load, packing load, initiating migration, migrating load, unpacking load, and updating global information)	Task Migration Strategy	Select load to migrate
	Acceptance policy		Task Selection Strategy	Migrate the load

Table 1: Comparison of Terms Describing Components of Load Balancing

All of these load balancing components or "policies" are candidates for experimentation. For example, the information policy can make use of a number of different measures related to current and projected workload, as can the transfer policy in making load-sharing decisions. Also, the transfer policy can consider a number of different schemes to determine whether the benefits of load balancing will outweigh the costs. If it is determined profitable, then the selection policy can apply a number of different optimization techniques or heuristics in deciding which jobs to transfer. Similarly, the location policy can use a number of different strategies in recommending where to transfer them (i.e., the destination). Finally, if it's even used, the acceptance policy can use different rules to govern whether a destination processor will accept a proposed job transfer.

Load balancing can be done either statically (Wilson and Nicols, 1995; 1996) or dynamically (Boukerche and Das, 1997; Deelman and Szymanski, 1998) or through both combined (Boon et al., 2000). In the static approach, logical processes (LPs) are assigned to processors prior to run-time and the LPs run on the same processor during run-time. This approach would not require load-migration methods. In the distributed simulation world, this type of static load balancing is normally done though scenario based allocation of system to computational platforms. In the dynamic approach, LPs are migrated from one processor to another according to variations in system load. Both approaches, static and dynamic, have been investigated in parallel discrete event simulations; however, because static approaches cannot accommodate the dynamic nature of a simulation, the general consensus is that the static approach, in isolation, is not adequate to achieve good performance. As such, more recently, researchers have focused on using a dynamic approach.

With the goal of developing a dynamic load-balancing algorithm, the policies we focus on here include the information, selection, and location policies. There are a number of characteristics unique to real-time distributed battlefield simulations that both complicate and facilitate the efficient implementation of these policies. With this in mind, the following sub-section provides review on one of those characteristics – DES scheduling paradigms. This information is presented as a baseline against which to measure a modification we later propose that will facilitate the information collection process in a dynamic load-balancing model.

Parallel Discrete Event Simulation (PDES)

A discrete-event simulation (DES) model assumes that the state variables being simulated change instantaneously at discrete points in time called "events" (Pooch and Wall, 2000). Per Fujimoto (1990), the three basic data structures of a DES are:

1. **System state -** the collection of state variables necessary to describe the system at a particular time
2. **Simulation clock -** a variable giving the current value of simulated time.
3. **Next event list -** a list containing the next time when each type of event will occur.

Because of the dynamic nature of discrete event simulation models, it is necessary to keep track of the current value of simulated time as the simulation proceeds. Thus, some mechanism is required to advance simulated time from one value to another. By and large there are two approaches by which time can be managed in a DES: time-stepped (aka fixed-time increment, periodic scan, or fixed-increment time advance) and event stepped (aka event-advance, variable time increment, next-event time advance, or event-driven). As seen in Figure 1, in an event-stepped approach, the simulation clock is advanced to the time of the next significant event. The time increment is made in concert with the time of occurrence of the highest priority event. For example, if the next event is not scheduled to occur for the next 1 or 2 minutes, the clock will advance forward that amount in one step. The nature of the jumping between significant points in time means that in most cases the next event mechanism is more efficient and allows models to be evaluated more quickly. On the other hand, in the time-stepped approach, the simulation clock is advanced at fixed intervals in increments of Δt time units. That is, the time increment is made regardless of whether anything will happen (i.e., event-firing) in the model.

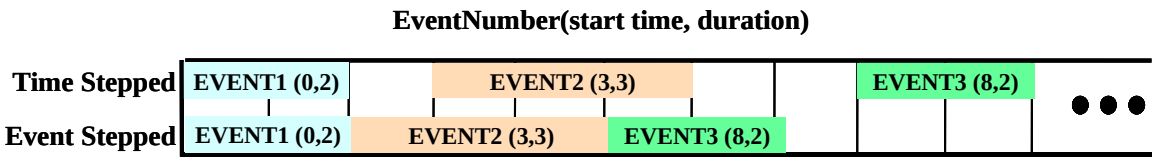

Figure 1. Arbitrary Example of Clock Advance Algorithm for Event Stepped versus Time Stepped

This concept is illustrated in Figure 1, where between Events 1 and 2 or Events 2 and 3, the clock advances independent of the scheduling of events. In this scheme, after each clock advance, a check is made to determine whether any event has occurred during the previous interval of length Δt and the system state is updated accordingly.

While the simulations we consider, real-time or scaled real-time battlefield simulations, are not technically time-stepped simulations (e.g., they do not use a common global clock, they are not conservative, etc.), they employ a similar type of scheduling paradigm. Rather, they use the assumption the system clock on each of the computational nodes does not drift with respect to the other systems and can be used with a virtual global clock. In doing this, the problems of rollback and global time management are assumed away. The only real problem with this synchronization paradigm is when one system cannot maintain the processing speed; it will lag behind. In terms of dynamic load balancing, the consequent of using this paradigm is that the busy wait used to synchronize to the system clock complicates the process of determining CPU load. A more detailed explanation of this problem and the approaches used to address it are presented in the following section on Related Work.

RELATED WORK

For background and context, the following section presents a general review on how other researchers have approached the load-balancing problem. Chiefly, it reviews a number of large-scale load balancing investigations and describes the policies they have examined. Where appropriate, the review also emphasizes methods particularly relevant to our research. Specifically, we focus the latter part of our review on approaches used to estimate load (information policy) and approaches used to assign LPs to processors (selection/location policies) in simulations of spatially significant domains such as battlefield simulations.

Studies Comparing Migration Policies

Willebeek-LeMair and Reeves (1993), as shown in Table 1, conceptualize four phases to their load-balancing algorithm. Of these, they exercise phases 2 and 3 experimentally, while policies guiding phases 1 and 4 are held constant. In phases 2 and 3, Profitability Determination and Task Migration Strategy, respectively, the authors experiment both with distributed implementations for scalability and centralized implementations for accuracy. Specifically, the factors they manipulate in their experiments are:

1. sender vs. receiver initiation of balance
2. size and type of balancing domains
3. degree of knowledge used in the decision process
4. aging of information in the decision process
5. overhead distribution and complexity

Combining these factors according to their knowledge of the domain, Willebeek-LeMair and Reeves consider five different combinations:

1. Sender-Initiated Diffusion (SID) – source initiates load balancing and distributes load to lightly-loaded nearest neighbors.
2. Receiver-Initiated Diffusion (RID) – destination requests transfer from neighbors, when load is light.
3. Hierarchical Balancing Method (HBM) – organizes process into hierarchy and applies increasing degree of knowledge as load balancing asynchronously moves from lowest to highest echelons.
4. Gradient Model – uses threshold approach (low-water-mark and high-water-mark) to determine if processor is candidate for sending or receiving load. Must have at least one of each, for load balancing to execute.
5. Dimension Exchange Model (DEM) – uses an iterative approach like HBM by performing steps in a synchronized manner.

Vaughn and O'Donovan (1998), in their experiments for general parallel systems, investigate load-balancing algorithms with the objective of reducing mean job response time as compared with system performance when no load balancing is used. Using CPU queue length for their estimate of load index, Vaughn and O'Donovan implemented and evaluated 9 algorithms:

1. random – randomly selects a non-local processor as the execution site for an eligible job.
2. global - centralized information policy and decentralized location policy. Uses some threshold to determine, globally, if processors have lower load index than a local node.
3. central – both information and location policy are completely centralized.
4. disted - completely decentralized information and location policies.
5. threshold - information collection is demand-driven and then the selection policy is random. Must meet the threshold required for transfer to occur.

6. lowest – has demand-driven information policy which employs a threshold at the destination and selects destination with the lowest load.
7. krmod – uses same information policy as disted. Location policy is similar to disted but probes the local node to confirm load information.
8. adsend - sender initiated, adaptive algorithm. Does not use execution time filter for transfer policy, but only considers CPU queue threshold.
9. reserve – only receive-initiated algorithm.

Wilson and Shen (1998) use their two-phased approach (see Table 1) to consider a number of different policies used to assign LPs. In the first phase, they use a central processing mechanism to make migration decisions globally, and then use a local processor to select the load to be migrated. In their implementation, every host stops processing work while a central coordinator collects load information. This coordinator analyses systems on a pre-migration procedure that computes the CPU utilization and load distribution in the system, and makes a decision on whether load migration is required. If it is required, the algorithm redistributes load according to processing rate on each processor and we proceed to "level 2".

The local load status (given to central coordinator) includes information on the total amount of pending load, the local processor CPU utilization, and the amount of load processed since the last load migration. Then, the central coordinator computes the variance of CPU usage, total amount of pending load, and variance of pending load distribution. These metrics help determine whether load-migration is even necessary. Estimation considers pending load with some threshold. If pending load is over threshold, load migration is enabled and Phase 2 begins.

In the second phase, the sending hosts have been informed of transfer and start selecting the local load to be transferred based on one of a random, communications-based, or load based (RCL) selection policy. In the random policy, the sender randomly selects local objects until the combined load of the objects fulfills the requested amount. In the communication-based policy, the sender moves the objects that communicate the most with the receiving host. And, in the load based policy, the objects with largest pending loads are migrated first.

Approaches to Estimating Processor Load

One of the differences between load balancing for general parallel problems and PDES is deriving a measure of processor load. As mentioned in Deelman et al. (1998), general algorithms use CPU utilization. This strategy can also be used in event-stepped simulations, as in Pears and Thong (2001) or in Wilson and Shen (1995), where the kernel load metric is derived from the CPU utilization or the operating system load figure. However, because these measures are meaningless in the time-stepped paradigms, researchers have developed other metrics to estimate/predict processor loading. These include a simple count of the number of tasks pending execution or queue-length (Willebeek-LeMairs and Reeves, 1993; Kunz, 1991), counting the number of objects assigned to a given LP, and the simulation advance rate (Glazer and Tropper, 1993).

The reason why the OS load figure is useful in event-stepped paradigm, but not in a time-stepped paradigm can be illustrated through Figure 1. For the slice of time considered in Figure 1, the event-stepped paradigm scheduled three events contiguously over seven time units. This can be abstractly represented as a 70% utilization rate at the system level. However, because the event firing is not contiguous in the time-stepped paradigm, this system relies on the busy wait construct to cycle through a timed-delay loop so that the event can fire at the appropriate time. Thus, between Events 1 and 2 and between Events 2 and 3, the busy wait cycle is active. As a result, at the system level, the cycles are fully consumed by the application. This can be abstractly represented as a 100% utilization rate at the system level. Thus, for the time-stepped paradigm, deriving the processor load from the system load report will always report a higher load (i.e., fully-loaded) than the application is practically experiencing.

The queue length method of load determination falls apart due to the heterogeneous computational event requirements. For example, it normally takes significantly longer to determine the geometric intervisibility between two entities than it does to update the position of a destroyed vehicle. Not only that, but this scheme also does not take into account the amount of time over which the events are scheduled. Likewise, the number of objects assigned to a process fails to take into account the computational and communication load of the various types. For example, a truck is normally significantly less expensive to model than a helicopter. However, knowing the relative cost of each of the objects is one of the most commonly used methods of doing static scenario based load-balancing.

As way of getting around the fact that we cannot rely on one system for CPU utilization, nor use any of the

counting methods, we present a modification to the DES scheduler such that it allows for the identification of trends in processor loading. Briefly, the modification involves tracking the amount of time spent in doing the busy waits; in essence the application's free time. This time is then summed over a given time increment, notionally 1 second. From this information, it is possible to compute both the instantaneous load and historical load trends at the application level. We use this load information to determine if a processor is a candidate for load migration. Details of this implementation may be found in Pratt (2003).

Approaches used in Partitioning Spatial Problems

Workload in battlefield simulations tends to be positively correlated to physical space. Because of that, characteristics of the battlefield simulations have a bearing on problem partitioning, processor load definition, and load migration (Deelman and Szymanski, 1998). For example, the fact that units dynamically move has profound impact on load balancing. Also, computational load and interaction requirements tend to increase when entities are geographically close, as each entity positions itself and communicates and interacts with sets of nearby entities. Thus, many load-balancing algorithms for spatially explicit problem domains (e.g., battlefield simulations) have mapped regions of domain to processors instead of units to processors. For example, Nicol (1987) partitions battlefield into sectors shaped like hexagons, as seen in Figure 2a, and then assigns these sectors to processors. He couples this strategy with the use of redundant computation as a way of exploring the space between optimizing load-balancing while satisfying constraints on communications requirements.

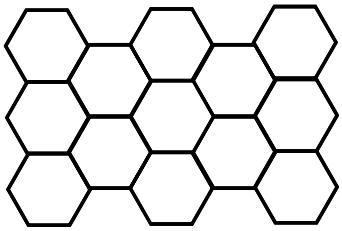

Figure 2a. Hexagon Partitioning Scheme

Niedringhaus (1995) reports on the development of a dynamic load balancing scheme for a wargaming simulation with an order of magnitude 10K – 100K vehicles. In his approach, he partitions the battlefield terrain across nodes through use of a grid- partitioning scheme, as seen in Figure 2b. He uses this approach to achieve dynamic balancing through swapping terrain parcels.

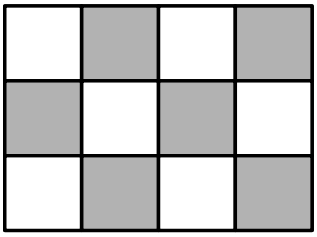

Figure 2b. Grid-based Partitioning Scheme

Though not working with battlefield simulations, Deelman and Szymanski (1996) also recognize the utility of applying a spatial partitioning scheme by using a stripped partitioning scheme (Figure 2c).

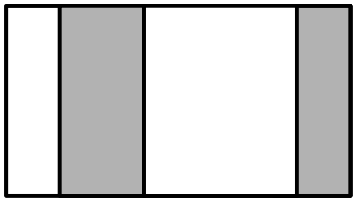

Figure 2c. Strips Partitioning Scheme

Using this scheme, they develop dynamic load balancing scheme for PDES simulation of the spread of Lyme disease at the ecological level. This approach required knowledge of the loads of ALL the processors (either broadcasted or centralized) and uses nearest-neighbor migration necessitated by the spatial characteristics.

Many of the geographic based partitioning schemes have found their way in to the Data Distribution Management (DDM) functions of distributed systems to take advantage of the locality of communication between the objects. The problem with using these static schemes for load balancing is the objects tend to move en mass from one area to another leaving large areas underutilized and others having spiked loads. Having said that, we adopted the elements of geographically partitioning the simulation space and implementing a number of application and system configuration heuristics as they relate to the entity-based battlefield simulations. The implementation and experimentation of these policies are described in greater detail in the following section on Methods, where we present our test-bed, the assumptions in our design, and our experiments.

METHOD

The previous section reviewed past approaches to load balancing. This section distinguishes our work from and builds on these approaches.

Testbed Design

We made the conscious decision early on in this program to focus on the scheduling and load balancing issues rather than the operational correctness of the system. As such, we decided it would be easier to implement our own very lightweight simulation system than it would be to integrate our ideas into a production level system such as OneSAF.

The benefits of using object-oriented (OO) paradigms for the specifications of simulation, particularly DES, are well understood. For example, the class concept provides an intuitive mapping to the simulation model and instances of classes can be naturally mapped to the simulation entities. OO principles like these coupled with a number of features in Java and the Java Virtual Machine (JVM) that lend themselves to the development of distributed simulations, led us to use Java in the development of the testbed reported in this study. For example, because the LPs and their model instances are executing in separate JVMs (i.e., distributed over multiple processors), in order to affect the simulation, the LPs need to exchange user-defined information. Java supports this capability directly and transparently through the interface *java.io.serializable*. Any instance of a class that implements this interface is easily transferred from one JVM to another. We make use of this feature in the distribution of model instances (e.g., in our Entity class), the exchange of event messages (e.g., in our MsgObj class), as well as assignment of sectors (e.g, in our AOI class). This significantly simplified the implementation of the networking infrastructure.

Our test-bed framework supports modularity, localizes network interface code in a few key classes, and is composed of a number of interacting base classes that provide the basic simulation functions discussed in the Introduction section. These include:

EntityList - an indexed data structure that provides an index to all the simulation entities. This is used when we must step though the entities to find a specific one or perform aggregated functions such as center of mass computations.

Pqueue – an indexed data structure storing events in increasing timestamp order

Sched/TimedSched – the active entity that drives the kernel forward. It selects the next event, delivers the event to the corresponding object. TimedSched is a special case of Sched with enhancements that support the derivation of a system load estimate appropriate for use with DES scheduling paradigm.

Events – there are four major types of events

1. Master event - centralized on "master" node and performs simulation management functions
2. Status event – locally collects and reports information on load
3. Net event – connecting and communicating over network.
4. Entity event – includes creating, moving, and initiating migration decisions

Once the migration controller is invoked, our policies to support when and where to move entities are defined in the migPolicy and migSelection classes. Four strategies are implemented in each and are detailed below.

MigrationPolicy – decide when to move local objects.

1. Random – randomly selects whether local object should be moved
2. MigPolicyGeographic – will move object if it resides in AOI that is governed by remote node
3. MigPolicyLoad – will move object if another node has a lighter load
4. MigPolicyProximity – will move an object if it is "nearest neighbor" to another node's center of mass

MigSelection – decide where to move local objects

1. Random – randomly selects where the object will be moved.
2. MigSelectionFixedAOI – will move object to node that governs AOI object resides in
3. MigSelectionLightest – will move object to node with lightest load
4. MigSelectionClosest – will move object to node that hosts object's nearest neighbor center of mass

Load Balancing Algorithm

We use a diffusive approach in the implementation of our migration scheme. In contrast to centralized approaches, where a single "master" node determines the loads of all other nodes and determines balancing strategies, a diffusive approach allows all nodes to communicate and share tasks. This approach tends to scale better since the CPU load and message-passing load are distributed over all processors. However, currently only one processor drives the adjustment of the physical sectors (AOI). Thus, briefly, our load-balancing algorithm for the master node in a two node case is to:

```
Update local load measure
Update remote load measure
If own_load > destination_load
then
   Increase AOI
Else
   Decrease AOI
Schedule next load balancing event
Continue
```

Central to our algorithm and novel to the community is the approach developed and used to estimate workload. Following Niedringhaus (1995), we implement this metric as a measure of workload over the just-ended interval. Currently, we have only implemented the instantiations load factor. We are planning to implement the trend load factor to do a better job in the prediction of loads.

Experiments

In the conduct of the initial set of experiments, we made a series of assumptions. Outlined below, we feel that these assumptions do not limit the general applicability of the system.

- Physical movement of and intervisibility between entities on battlefield is biggest contributor to load.
- The distributed system is made up of heterogeneous platforms
- Processors are dedicated
- Communications lags are negligible
- The functionality testing was conducted on a four processor distributed cluster, the experiments were conducted a two processor cluster of workstations.

Five combinations of policies were implemented and six configurations were evaluated in a two node, master/client, system. The five cases and six experiments considered were:

1. No load balancing
2. Random by Random (Conservative)
3. Random by Random (Liberal)
4. MigPolicyGeographic by MigSelectionFixedAOI
5. MigPolicyLoad by MigSelectionLightest
6. MigPolicyProximity by MigSelectionClosest

In all six experiments, the master node was initially loaded with a demanding number of entities and the load measures for both the master and client were examined over time. The master machine had a 1GHz AMD Athlon processor, 512MB RAM, and was running Windows 2000 Professional. The client machine had a 1GHz Intel Pentium III processor, 512 MB RAM, and was also running Windows 2000 Professional. Since both the simulations and the processing capabilities of each machine were similar, we simulated heterogeneity by adding a demanding number of print-lines to the simulation on the master node.

We loaded the master node with 100 entities and gathered information on load over time for both nodes. Thus, the master node would be fully loaded at the start of a simulation run and the client would have no appreciable load. Thus, for a policy to be "ideal", we would expect the loads of the two processors to even out. Or, stated another way, we would expect the difference in load of the two processors to be 0.

Experiment 1 served as the primary baseline for the experiments and experiments 2 and 3 served as a secondary baseline. While these experiments (Random by Random) were implemented as migration schemes, they made no use of dynamically gathered system state information (e.g., load or AOI). Thus, these algorithms can be thought of as "static" load balancing implementations and their experiments provide a useful data point for measuring the effectiveness of dynamic algorithms that employ heuristics (e.g., experiments 4, 5, and 6).

RESULTS

Results of the first three experiments can be seen below in Figure 3, which plots the difference in processors' workload over time. As expected for a fully stressed system, the load for case 1, "no load balancing", was measured at 100%. In this case, the workload was big enough that the processor was stressed to keep up with real-time requirements. Experiments 2 and 3, conservative and liberal load balancing, both illustrate that even load balancing that makes no use of dynamic information still performs better than no load balancing at all. Also, experiments 2 and 3 illustrate that cross comparisons of load balancing algorithms must be interpreted generally and at a high level, because the performance of any one combination of policies can be affected by level of parameters settings. For example, the performance difference between conservative random and liberal random is visually distinguishable, but the only difference between these experiments is the threshold used to decide which randomly generated numbers will result in decision to migrate or not. That is, in these experiments, a conservative random policy will return a decision to migrate 10% of the time, whereas a liberal random policy will return a decision to migrate 50% of the time.

Figure 3. Workload Difference in Two-Processor Load Balancing Trial (Baseline Cases: 1, 2, and 3)

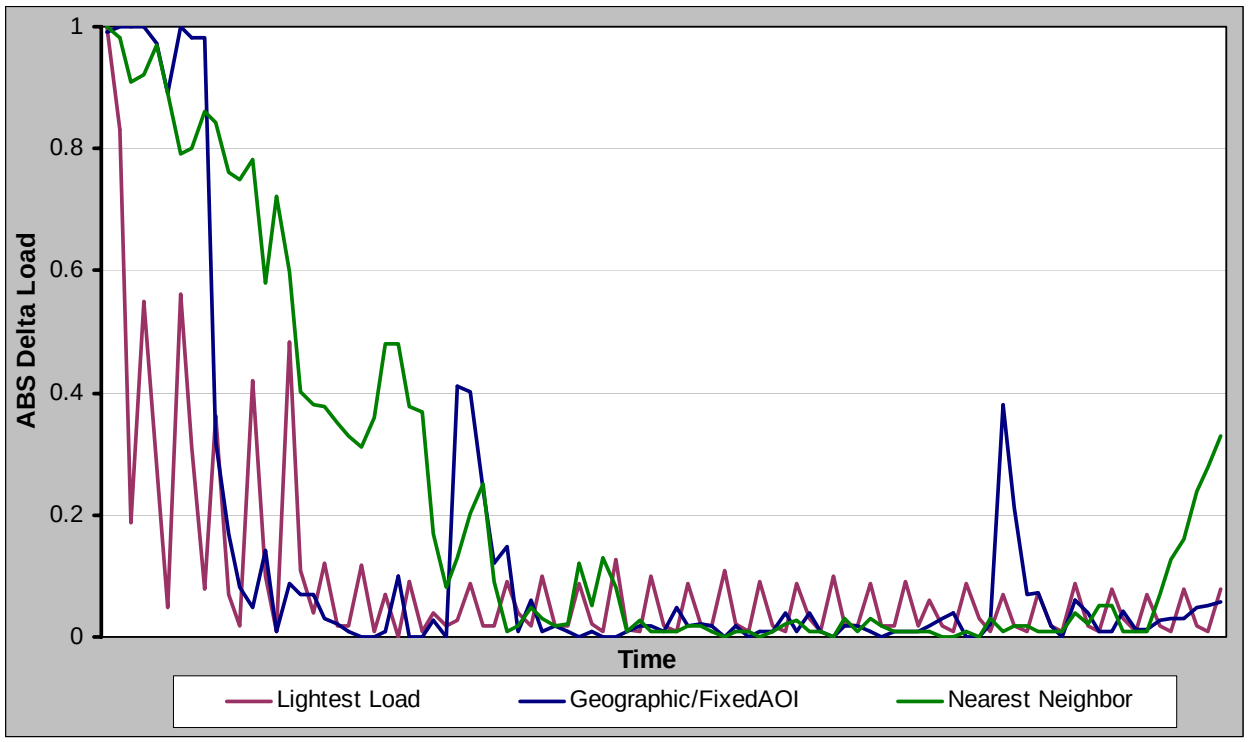

Figure 4. Workload Difference in Two-Processor Load Balancing Trial (Dynamic Cases: 4, 5, and 6)

Figure 4 plots the results of the last three experiments, the geographically-based (experiment 4), load-based (experiment 5), and proximity-based (experiment 6). Clearly, these all perform better than the cases presented in Figure 3, confirming our hypothesis that utilizing state information at execution time improves load-balancing performance. Of the three experiments reported in Figure 4, the Lightest Load strategy tended to provide the most stable load-balancing performance, only fluctuating slightly once it converged. The spatial based strategies were more prone to fluctuate. This is to be expected with a two processor configuration. As we test larger configurations we expect to see the lightest load fall out of favor due to excessive migrations and the geographic and proximity based paradigms exhibit better performance.

CONCLUSIONS AND FUTURE WORK

Many of the results presented here corroborate results obtained from simulations by other researchers. Our approach has been to reinforce this work by evaluating a wide range of algorithm design options contained within many load-balancing implementations. The central contribution of this algorithm is the approach developed and used to estimate workload for PDES that rely on busy wait construct to do scheduling. While we implemented this metric as a measure of workload over the just-ended interval, in future experiments we will enhance this metric to use predictions for the next Δt interval. Future experiments will also focus on the improvement of using this approach in contrast with other approaches used. But, for the time being, the results of this study at least provide some confidence that our approach is feasible.

ACKNOWLEDGEMENTS

Some of the work presented in this publication was made possible through support provided by DoD High Performance Computing Modernization Program (HPCMP) Programming Environment & Training (PET) activities through Mississippi State University under the terms of Agreement No. # GS04T01BFC0060. The opinions expressed herein are those of the author(s) and do not necessarily reflect the views of the DoD or Mississippi State University.

REFERENCES

Avril, H., and Tropper, C. (1996). The dynamic load balancing of clustered time warp for logic simulation, In *Proceedings Tenth Workshop on Parallel and Distributed Simulation PADS '96.* Philadelphia, PA. pp. 20-27.

Boon P., Yoke, H., Jain, S., Turner, S., Wentong, C., Wen, J., Shell, Y. (2000). Load balancing for conservative simulation on shared memory multiprocessor systems. In *Proceedings Fourteenth Workshop on Parallel and Distributed Simulation (PADS ' 00).* Bologne, Italy. pp. 139-46

Boukerche, A., and Das, S. (1997). Dynamic Load Balancing Strategies for Conservative Parallel Simulations. In *Proceedings of the 11th Workshop of PADS.* Lockenhaus, Austria. pp. 20–28.

Catothers, C., and Fujimoto, R., (1996). Background execution of Time Warp programs. In *Proceedings Tenth Workshop on Parallel and Distributed Simulation PADS '96.* Philadelphia, PA. pp. 12-19.

Choe, M., and Tropper, C. (1999). On learning algorithms and balancing loads in Time Warp. In *Proceedings Thirteenth Workshop on Parallel and Distributed Simulation PADS '99.* Atlanta, GA. Pp. 101-108.

Deelman, E.; Szymanski, B. K. (1998). Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. In *Proceedings Twelfth Workshop on Parallel and Distributed Simulation PADS '98.* Alta., Canada. pp. 46-53.

Fujimoto, R.M. (1990). *Parallel Discrete Event Simulation.* John Wiley & Sons.

Glazer, D. W. and Tropper, C. (1993). On process migration and load balancing in Time Warp. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, no. 4, pp. 318-327.

Kunz, T. (1991). The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering*, Vol. 17, no. 7, pp. 725-730.

Nicols, D. (1987). Performance Issues for Distributed Battlefield Simulations. In *Proceedings of the 1987 Winter Simulation Conference*. pp. 624 – 628.

Niedringhaus, W. P. (1995). Diffusive dynamic load balancing by terrain parcel swaps for event-driven simulation of communicating vehicles *Proceedings of the 28th Annual Simulation Symposium.* Phoenix, AZ. pp. 166-74.

Pears, A. and Thong, N. (2001). A dynamic load balancing architecture for PDES using PVM on clusters. In *Recent Advances in Parallel Virtual Machine and Message* Passing Interface. 8th European PVM/MPI Users' Group Meeting. Vol. 2131, pp. 166-73.

Pooch, U. and Wall, J. A. (1993). *Discrete event simulation: a practical approach*. CRC Press, Boca Raton, FL.

Pratt, D. (2003). Application Level Performance Monitoring in FMS Applications. Presented at the 2003 HPC User's Group Meeting.

Vaughan, J. G. and O'Donovan, M. (1998). Experimental evaluation of distributed load balancing implementations. *Concurrency: Practice and Experience* Vol. 10, no. 10, pp. 763-82.

Wilson, L. F. and Nicol, D. M. (1995). Automated load balancing in SPEEDES. In *Proceedings of the 1995 Winter Simulation Conference*, Arlington, VA. pp. 590-6.

Wilson, L. F. and Nicol, D. M. (1996). Experiments in automated load balancing In *Proceeding Tenth Workshop on Parallel and Distributed Simulation. PADS 96*, Philadelphia, PA. pp. 4-11.

Wilson, L. F. and Wei Shen (1998). Experiments in load migration and dynamic load balancing in SPEEDES. *In Proceedings of the 1998 Winter Simulation Conference.* Washington, DC, Vol. 1, pp. 483-90

Willebeek-LeMair, M. and Reeves, A. (1993). Strategies for Dynamic Load Balancing on Highly Parallel Computers, *IEEE Transactions of Parallel and Distributed Systems*, Vol. 4. no. 9.