Load Balancing for Distributed Battlefield Simulations: Tradeoffs in Workload and Communications

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

Load balancing attempts to optimize the utilization of processors in a parallel computing systems, and dynamic load balancing is a prime candidate for improving the performance of distributed battlefield simulation systems. Last year we reported on development of test-bed developed to assist in the empirical exploration of a number of dynamic load balancing heuristics. Unique to the test-bed was a modification of the classical discrete-event simulation (DES) scheduling paradigm that enabled us to determine processor load at the application level. Experimental runs considered a number of load-balancing heuristics, corroborated results reported by other researchers, and provided confidence that our approach is indeed feasible.

Absent from this initial study was a consideration of cost measures for the system and a recognition of the conflict between distributing workload evenly and minimizing communication costs. For a load balancing system to be effective, the cost of balancing load must be less than the cost of the status quo. The cost is manifested by the monitoring, selection, transport, and initialization time, versus the processing and bandwidth requirements. Without proper monitoring and calibration, it possible to spend more time trying to balance the load than it does actually processing productive work. In this paper we present the implementation of additional test-bed infrastructure designed to capture these tradeoffs. Moreover, we motivate the selection of heuristics to be considered, present the results of experimental runs with these heuristics, and discuss the implications of the results.

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 largest simulation software effort ever undertaken by the Department of Defense. Former 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. where she is principal investigator for the Individual Combatant's Weapons Firing Algorithm for U.S. Army Soldiers Systems Command, Natick, MA. 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 (now PEOSTRI). 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 has also taught as Adjunct Faculty for the Modeling and Simulation Graduate Program at UCF.

Load Balancing for Distributed Battlefield Simulations: Tradeoffs in Workload and Communications

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

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

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.

To be effective, load-balancing strategies must find a compromise between distributing the work and minimizing communication costs. Because the workload is moved through message passing, there is a conflict between distribution for load balance and distribution for low communication overhead. That is, if workload is distributed among the processors such that inter-processor 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.

As presented in initial paper (Pratt and Henniner, 2003), typically, a dynamic load-balancing strategy consists of four components that define the policies to be used: Transfer Policy, Selection Policy, Location Policy, and Information Policy. In review of research provided in last year's paper we report on studies comparing migration policies: Willebeek-LeMair and Reeves (1993), Vaughn and O'Donovan (1998), Wilson and Shen (1998) and, we offer review of approaches to estimating processor load found in Deelman et al. (1998), Pears and Thong (2001) or in Wilson and Shen (1995), Willebeek-LeMairs and Reeves (1993), Kunz (1991), and Glazer and Tropper (1993). Lastly, in that paper we also review of discrete event simulation scheduling paradigms based on Pooch and Wall (1993).

For those benefiting from a review of this topic, the next sub-sections present an overview of load balancing, in general, and load balancing issues in distributed battlefield simulations, respectively. Those not interested in this review may proceed to the next major section without loss of critical information.

Load Balancing

Load balancing algorithms attempt equally spread the load on processors. This implies that the difference between the heaviest loaded and the lightest loaded processors should be minimized. In general, load balancing algorithms can be broadly characterized as dynamic or static, centralized or decentralized, periodic or non-periodic, entity-based or space-based, and those with thresholds or without thresholds.

Dynamic vs. Static. 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). Static load balancing has been formulated as an optimal assignment problem that involves assigning tasks to processors prior to 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.

Dynamic load balancing involves the reallocation of tasks to processors after their initial assignments according to variations in system load. This is done by migrating tasks from overloaded nodes to other lightly loaded nodes to improve overall system performance. Such an assignment problem is NP-complete.

Both approaches, static and dynamic, have been investigated in parallel discrete event simulations; however, because static approach 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.

Centralized vs Decentralized. In a centralized load-balancing algorithm, the load-balancing scheduler on an individual processor will make all the load-balancing decisions based on the information that is sent from other processors. In contrast to centralized approaches, a diffusive approach allows all nodes to communicate and share tasks. Since every processor in the system keeps track of the global load information, load-balancing decisions can be made on any processor.

A centralized algorithm imposes fewer computational overheads on the system; however, it can be a communications bottleneck and it can be less reliable since the failure of the central scheduler will result in the system-wide malfunction of the load-balancing policy. A decentralized algorithm, on the other hand, is easier to implement and tends to be more reliable since the CPU load and message-passing load are distributed over all processors.

Periodic vs Non-periodic. This part of the algorithm specifies how to collect the workload information required to make the load-balancing decision. There is a constant tradeoff between the cost of collecting global load information and maintaining the accurate state of the system. In a periodical collection of workload information, individual processors will report their load to each other at a predetermined time interval. This is advantageous in the sense that the load-balancing operation can be initiated based on the well-maintained workload information without any delay. However, setting the interval for information exchange can be problematic. An interval that is too long would introduce inaccuracies in the decision making while a short time interval would incur heavy communication overhead.

Alternatively, non-periodic approaches may be based on on-demand or on-state collections. The on-demand exchange collects load information just before making a load-balancing decision. This rule is good for minimizing the number of communication messages, but it comes with the cost of extra delay for load-balancing operations. Alternatively, in the case of on-state, a processor will broadcast its load information to the rest in the distributed system whenever its load status changes. While it does maintain a global view of the system state, the large overhead in communication makes this rule impractical for large systems.

Entity-based vs. Space-based. The load-balancing algorithm is implemented by making each node responsible for a portion of the problem domain. This portion can be assigned either by using Lagrangian method or the Euclarian method. In the Lagrangian method, each node is responsible for a fixed set of entities. Since entities usually move in the simulated space by coordinating with their neighboring entities, this approach requires that each node keeps a snapshot of the complete simulated space and updates it throughout the simulation. This approach also requires that each node communicate with all other nodes in order to detect potential proximity of entities in the simulated space. Because of these requirements, this approach does not scale well in the case where between entity movement patterns are scattered and irregular.

The Euclarian method assigns to each node a portion of the simulated space, along with the set of entities located in that area. This approach is more scalable since each node only requires a snapshot of its neighboring nodes' simulated space. It also requires only near-neighbor communication for the purpose of proximity detection. However since entities migrate among nodes, a load imbalance is much more likely to occur, even if all entities perform the same type and amount of work. To correct the imbalance in this case, the load-balancing algorithm must operate dynamically; that is, it must periodically re-distribute the simulated space among the nodes, together with the entities residing in the various partitions. This method would also work well in cases where there's not a high degree of movement.

Threshold or Non-threshold. The dynamic load balancing strategy can often make use of a threshold policy. That is, when the load of a node exceeds some pre-specified threshold to send, T_s, the transfer policy will try to transfer part of that load to less loaded nodes. The location policy looks for those nodes whose load is less than the receiving load threshold, T_r, $T_r < T_s$. Of these nodes, the one with the lowest load is chosen as the destination for the object migration. If no such node is found, no object migration occurs.

The following sub-section details a number of characteristics unique to real-time distributed battlefield simulations that both complicate and facilitate the efficient implementation of these algorithm components.

Load Balancing Issues in Battlefield Simulations

The traditional approach to battlefield partitioning has been a static, predetermined partitioning of the battlefield into non-overlapping sectors and assignment of an equal number of sectors to each processor. In battlefield simulations, however, where a basic principle of military strategy is to concentrate forces in a small area, this approach often leads to a majority of the objects in one small part of the battlefield. Thus, a small number of processors end up doing the vast majority of computation.

Because workload in battlefield simulations tends to be positively correlated to physical space, a number of researchers have experimented with approaches used in partitioning spatial problems (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 used a Euclarian mapping of domain to processors instead of a Lagrangian mapping of units to processors. For example, Nicol (1987) partitions battlefield into sectors shaped like hexagons and then assigns these sectors to processors.

Niedringhaus (1995) reports on the development of a dynamic load balancing scheme for wargaming simulation with 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 1. He uses this approach to achieve dynamic balancing through swapping terrain parcels.

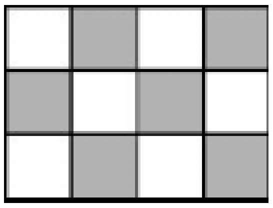


Figure 1. Grid-based Partitioning Scheme

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 other leaving large areas underutilized and others having spiked loads. Having said that, we adopted the elements of geographically partitioning the simulation space and implement a number of application and system configuration heuristics as they relate to the entity-based battlefield simulations. The next section describes the algorithm used in Phase I system model.

TESTBED: PHASE I SYSTEM MODEL

A migration policy is based on two decisions: when to migrate processes and which processes to migrate. Once migration controller is invoked, our policies support these decisions as characterized in Table 1 below. As reviewed in earlier section "Load

Table 1. Summary Comparison Analysis

	Random	Lightest	Closest	Fixed AOI	Dynamic AOI
Threshold or Non-threshold	Non-threshold	Threshold	Non-threshold	Non-threshold	Non-threshold
Euclarian or Lagrangian	Lagrangian	Lagrangian	Lagrangian	Euclarian	Euclarian
Diffusive or Centralized	Diffusive	Diffusive	Diffusive	Diffusive	Both[1]
Periodic or Non-periodic	Periodic	Periodic	Periodic	Periodic	Periodic
Dynamic or Static	Static[2]	Dynamic	Dynamic	Dynamic	Dynamic

[1] Load information policy is diffusive, but dynamic adjustment of AOI is centralized.
[2] While executed during runtime, the random policy makes no use of runtime information and is therefore comparable to static policies.

Balancing", either or both of the policies can be based on processor loading. The next subsection illustrates how we derived processor load and the subsection after details the five policies governing when and where to move the entities.

Migration Information Policy

The following represents a modification of the classical discrete-event simulation (DES) scheduling paradigm that enabled us to determine processor load at the application level. In short, If the schedule time for the first event on the queue is greater than the current time, we wait for the time to elapse. That waiting time is the idle time. By recording the first time we checked and found the time greater than the current time and when the event did fire, we were able to track (and record) the time we were idle. At the end of one second we sum up the times to get the total idle time the interval. Knowing we are either busy or idle and how much time there was in the interval, we can then compute the busy time and the load. Essentially then,

Load = (Total Time - Free Time)/Total Time

where the scheduling algorithm used to capture this information is presented below with modifications to typical scheduler boldfaced in red.

```
protected void processEvents(long stopTime){
   long curTime;
   boolean idleFlag = false;
   long idleStart = clock.getTime();
   while ((curTime = clock.getTime()) < stopTime){
     if ( (sched.theQ.size() > 0) &&
(sched.theQ.getNextTime() <= curTime)){
       if(idleFlag == true){
         idleFlag = false;
         busytimes.add(new
TimeRec(idleStart,curTime));
       }
       sched.theQ.dequeue().run(curTime);
     } else {
       if (idleFlag == false){
         idleStart = curTime;
         idleFlag = true;
       }
     }
   }
 }
```

Each node in our system also needs to keep the load information of other nodes of the system. In order to update the load information load information is acquired periodically (1 second ticks) through a multicast Load Distribution message and recorded in a local load table. Then, during processing of a load-balancing event each node accesses its local load table.

Migration Selection and Location Policies

As seen in Figure 2, basic set up for implementation

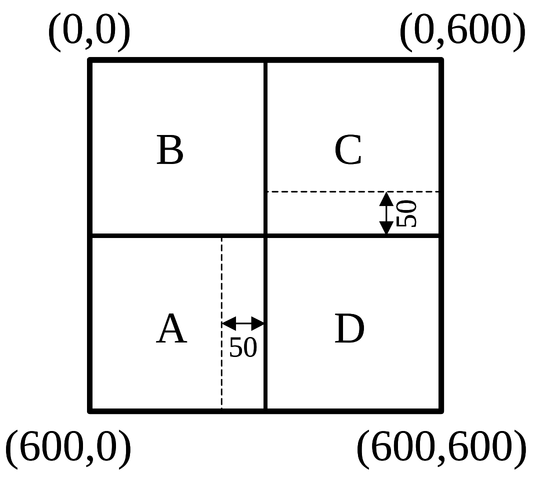


Figure 2. Initial Processor Mapping to Terrain Parcels

used to run experiments in this effort was a grid-based mapping of four-cells to four processors. As presented earlier and detailed later in this section, in some policies this mapping was entity-based (e.g., all entities created in cell A mapped to host A) and in others it was terrain-based (e.g., cell A mapped to host A and all entities residing in cell A are mapped to host A by default).

We defined the following quantities used by the selection and location policies.

Symbol	Definition
m_k	*machine (host) k (k = a,b,c,d)*
e_i	*entity i (i = 1,…,n)*
$m_k(e_i)$	*machine on which e_i resides*
L_k	*total load of m_k*
MC_k	*mass center (X,Y) of entities on host k*

$AOI(e_i)$	*terrain parcel on which e_i operates*
$Pos(e_i)$	*X,Y location of e_i*
R_s	*random threshold for sending*
R_i	*random variable drawn for e_i*
migcnt	*randomly generated index to hosts*
t_{now}	*clock time*
t_{stop}	*simulation end time*

```
while $t_{now} < t_{stop}$
{
    update $L_k$ (for all k)
    execute load-balancing event
    schedule next events
    continue
}
```

where executing a load-balancing event consists of deciding whether a migration should take place, and then to where the entities should migrate. These components form the heart of the policies we implemented. Each is detailed below.

1. Random: This policy randomly selects whether the object should be moved and uniformly distributes where the object will be moved.

```
For every $e_i$ on $m_k$
  If $R_i < R_s$,
    then migrate $e_i$ to (migcnt+1)%3+1
```

2. LightestLoad: This policy will move object if another node has a lighter load and move object to node with overall lightest load.

```
For every $e_i$ on $m_k$
    If $L(m_k(e_i)) > min[L(m_k != m_k(e_i))]$,
      then migrate $e_i$ to $min[L(m_k != m_k(e_i))]$
```

3. Closest: This policy will move object if it is "nearest neighbor" to another node's center of mass.

```
For all $m_k$
  calculate $MC_k$
For every $e_i$ on $m_k$
    If $dist(Pos(e_i), MC(m_k(e_i))) >$
      $min[dist(Pos(e_i), MC(m_k != m_k(e_i)))]$,
      then migrate $e_i$ to $min[dist(Pos(e_i),$
        $MC(m_k != m_k(e_i)))]$
```

4. FixedAOI: This policy will move object when the new position of an entity is outside of the current node's environment. Thus, the entity will automatically migrate to the appropriate neighboring node.

```
For every $e_i$ on $m_k$
  If $AOI_k(e_i) != m_k(e_i)$
    then migrate $e_i$ to $(m_k = AOI_k(e_i))$
```

5. Dynamic AOI: when the new position of an entity is outside of the current node's environment, the entity automatically migrates to the appropriate neighboring node, as in the Fixed AOI case. The Dynamic load balancing is then performed by moving the partitioning boundaries, which then redistributes load.

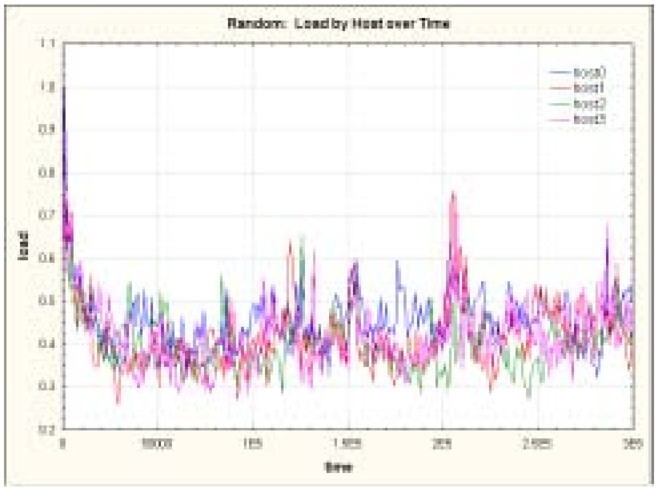
```
For every $e_i$ on $m_k$
  If $AOI_k(e_i) != m_k(e_i)$
    then migrate $e_i$ to $(m_k = AOI_k(e_i))$

If $L(m_k(e_i)) > min[L(m_k != m_k(e_i))]$,
  then increase $AOI_k$
else decrease $AOI_k$
```

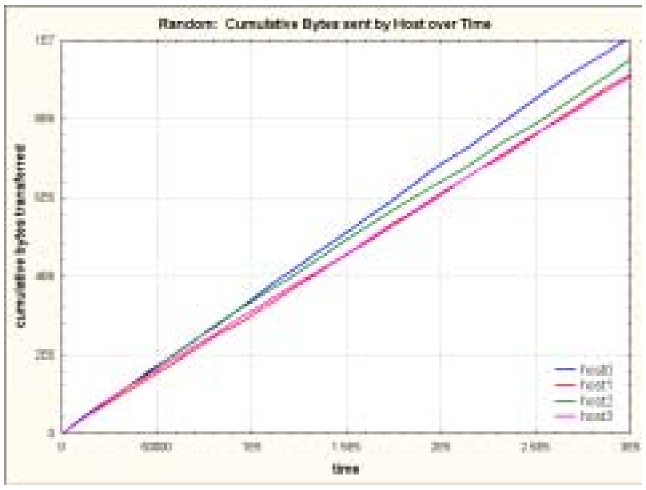
EXPERIMENTS AND RESULTS

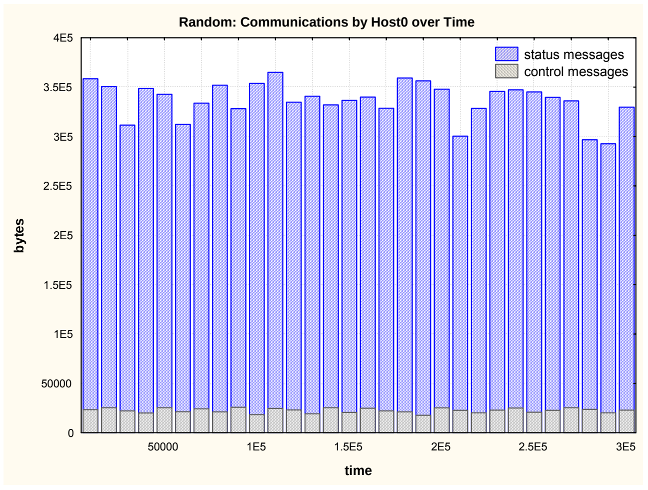
The five migration policies characterized in Table 1 were implemented and evaluated in a four-node system embedded in a 2048 processor system, the SGI Origin 3900, with 700 MHz MIPS R16000. All the hosts are homogenous in the sense that they have the same software and hardware architecture and the same processing power, and communication between hosts was done solely by message passing.

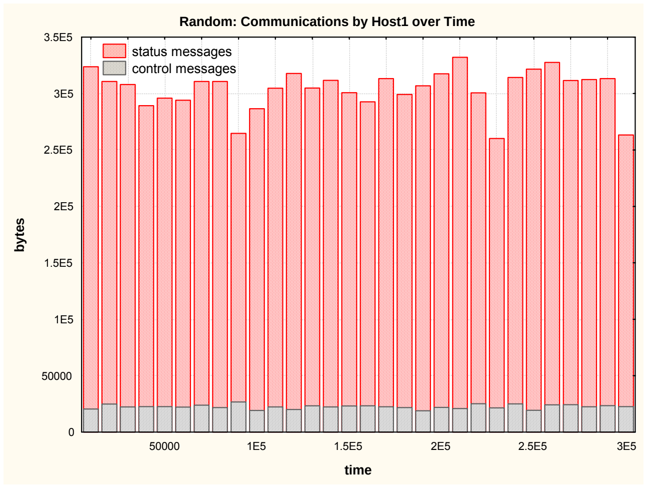
In all five experiments, we loaded each node with 50 identical entities and gathered information on load and messages transferred over time for all nodes in the system. The results from these runs are presented below in Figures 3a – 7f, corresponding to the Random, Lightest load, Closest, Fixed AOI, and Dynamic AOI policies, respectively.

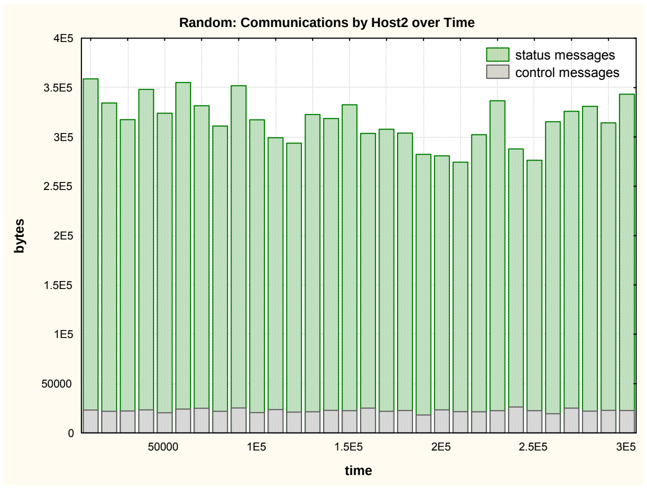
With the goal of a load-balancing algorithm being to minimize the variance of the load among all the machines in the cluster, we offer the following observations.

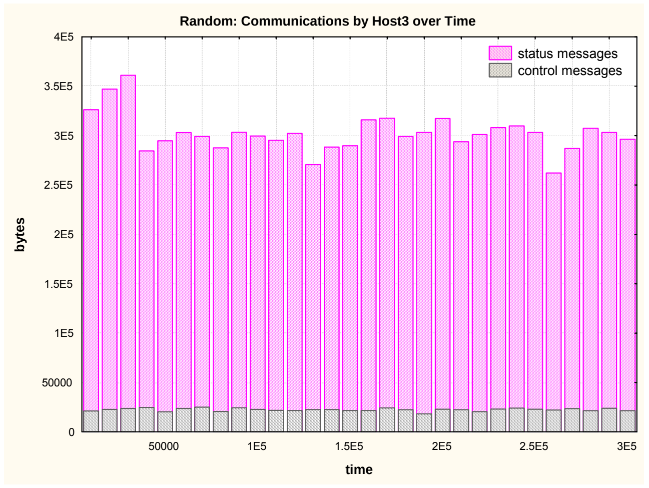




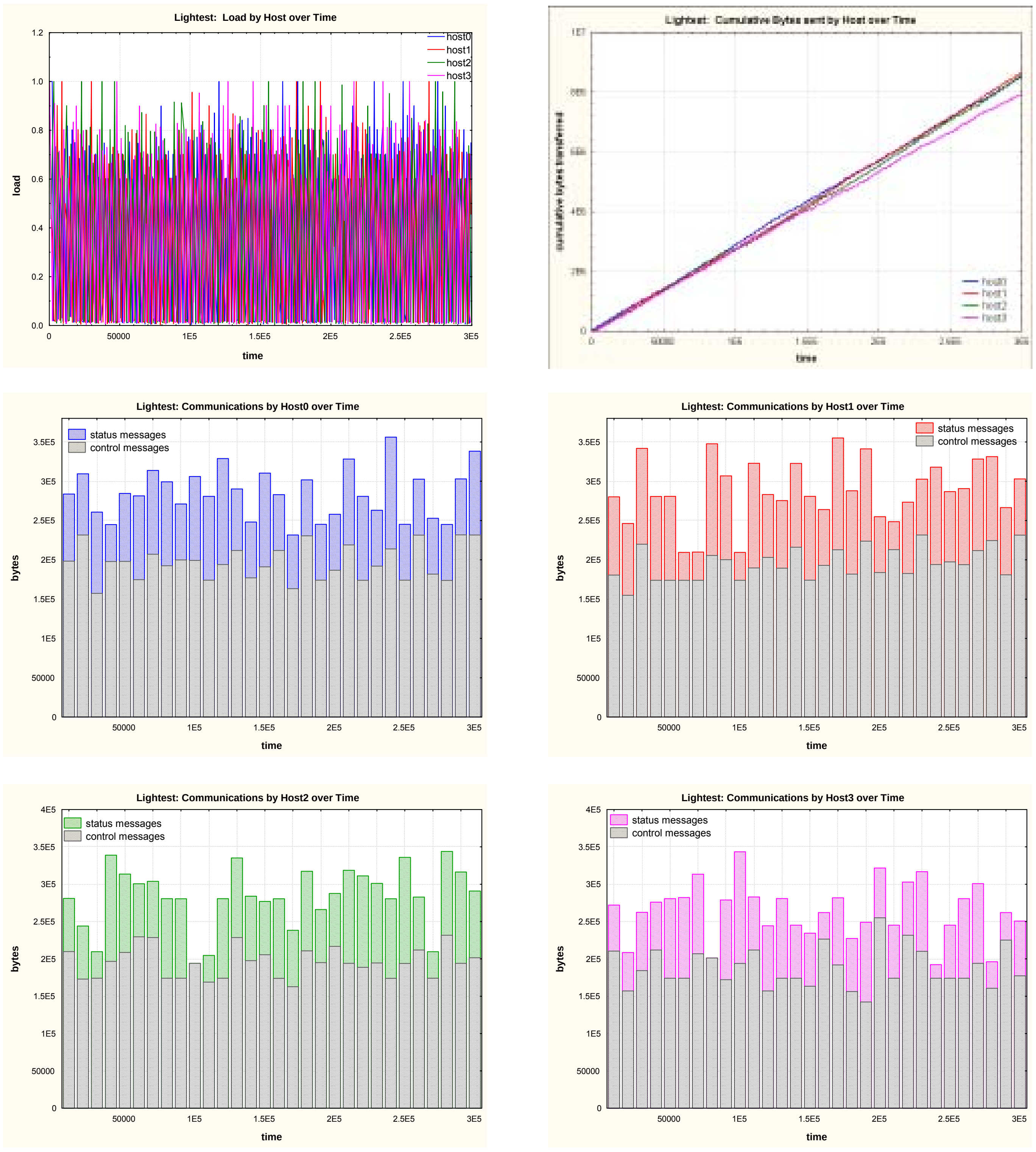




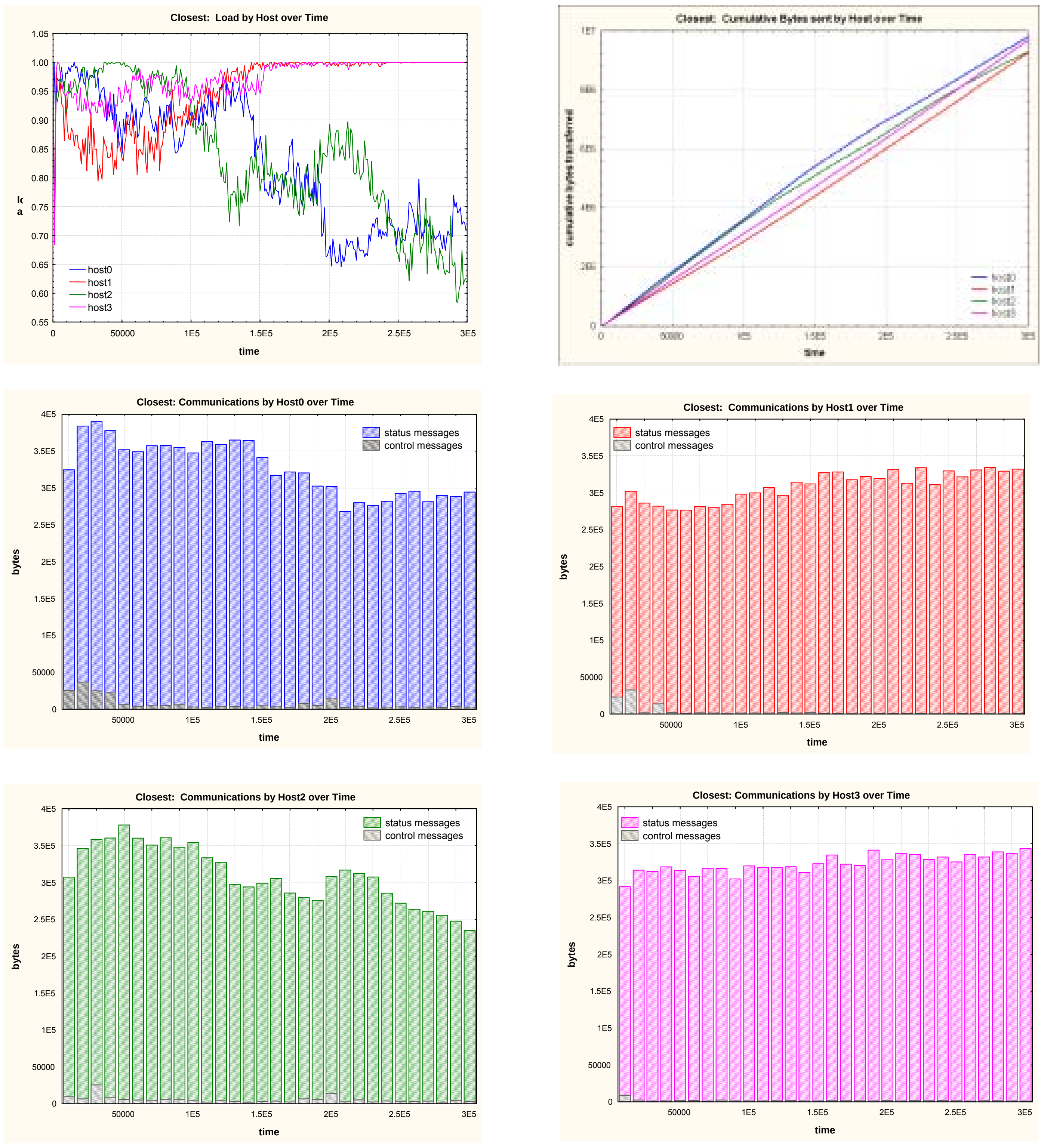




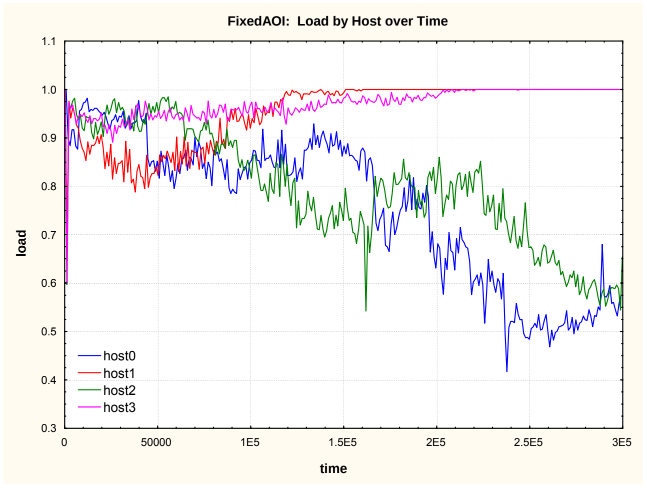
Figures 3a – 3f. Load (a. left, top), Cumulative Communications (b. right, top), and Communications Costs by Host (bottom four) of Random Policy

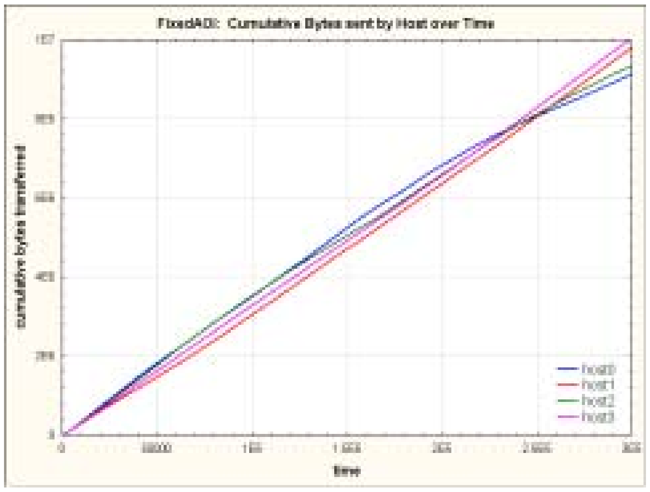


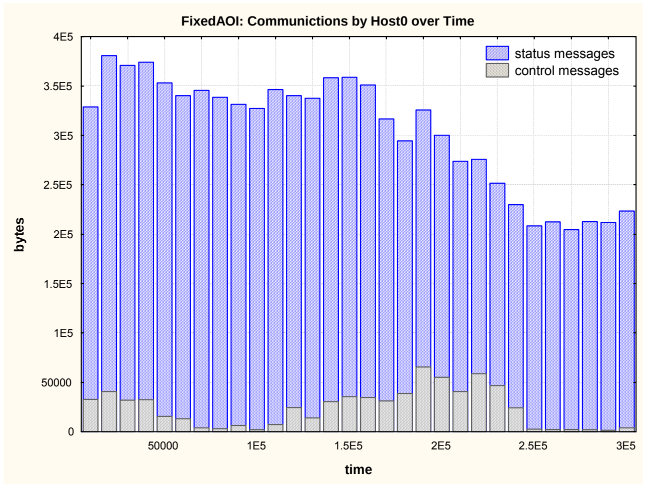
Figures 4a – 4f. Load (a. left, top), Cumulative Communications (b. right, top), and Communications Costs by Host (bottom four) of Lightest Load Policy

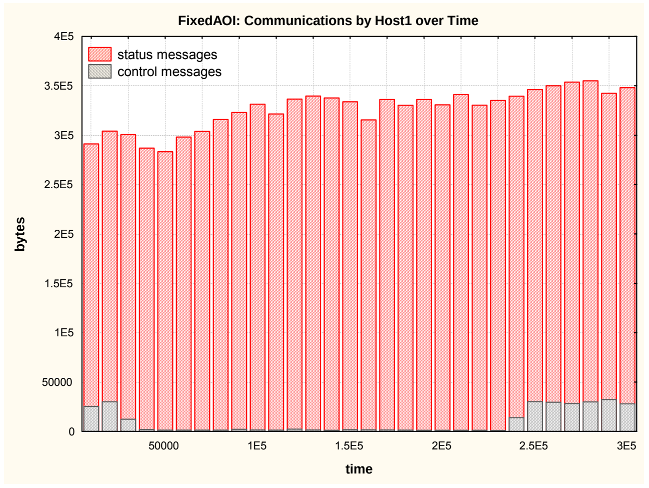


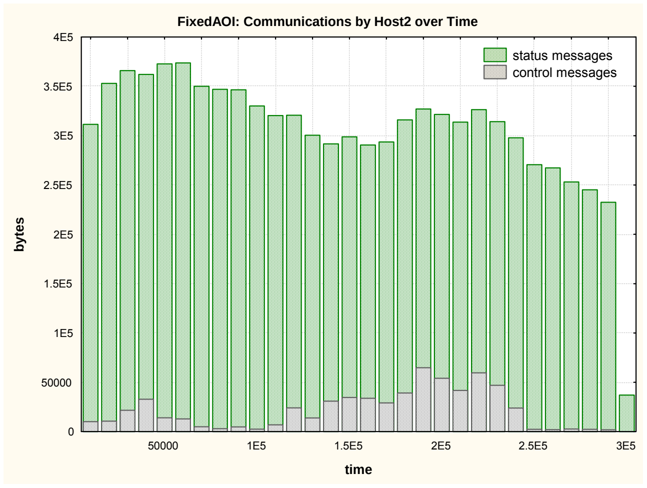
Figures 5a – 5f. Load (a. left, top), Cumulative Communications (b. right, top), and Communications Costs by Host (bottom four) of Closest Policy

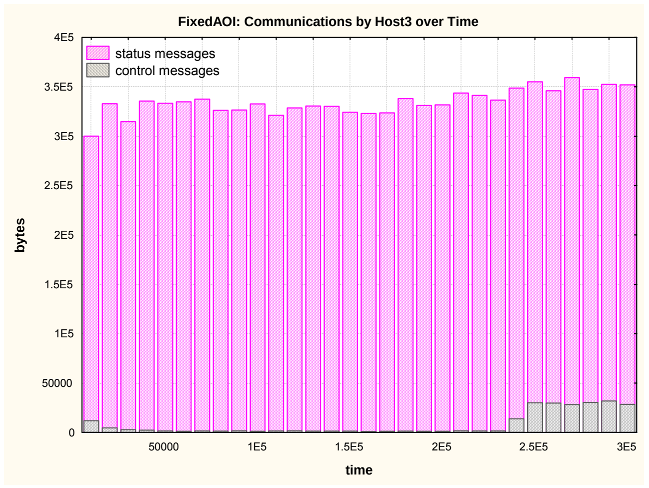




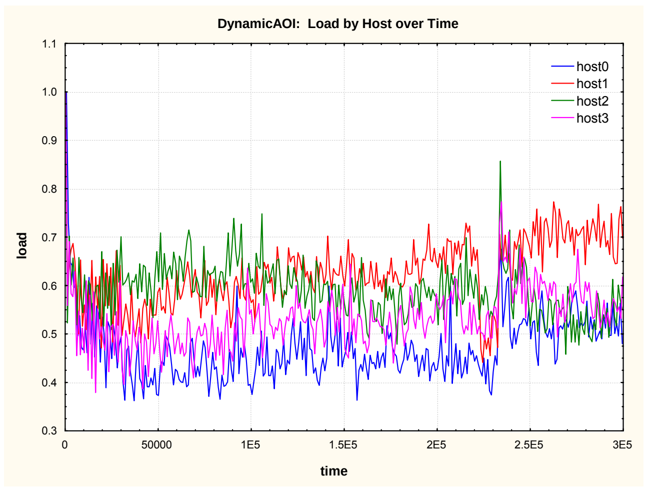


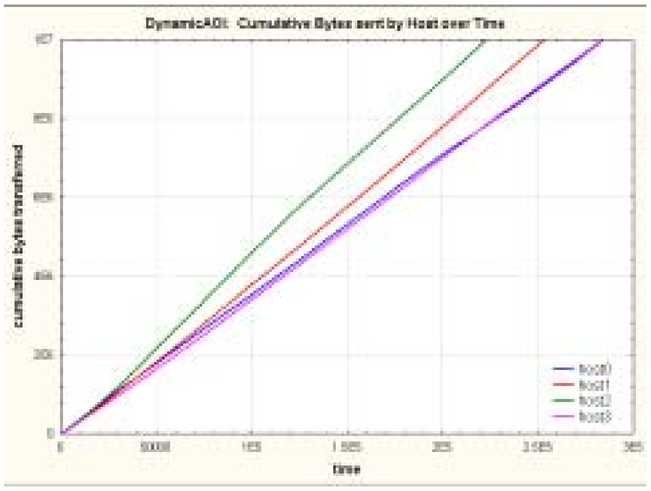


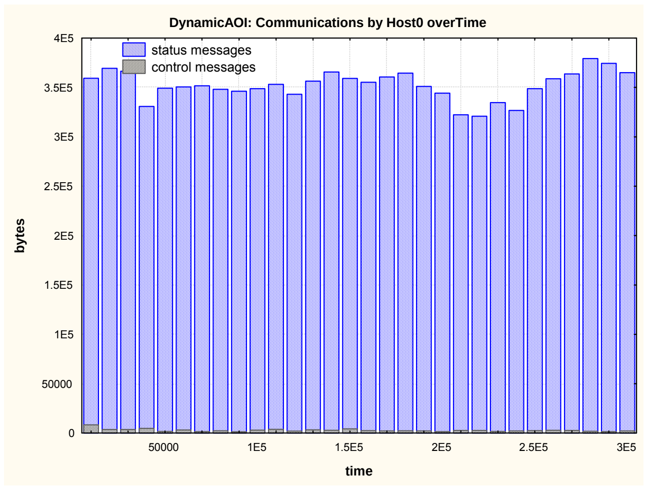


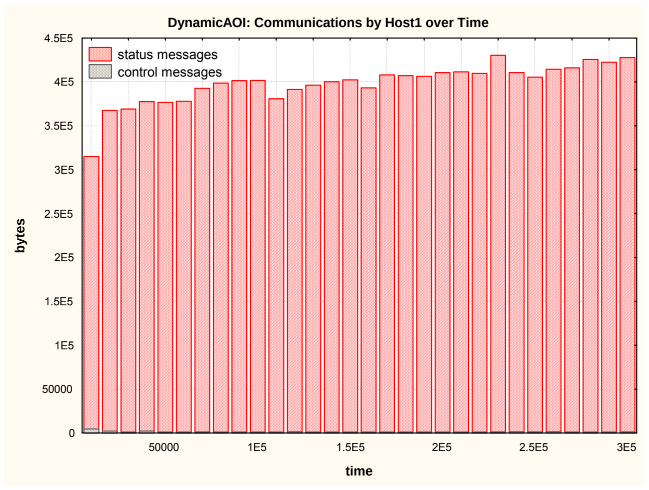


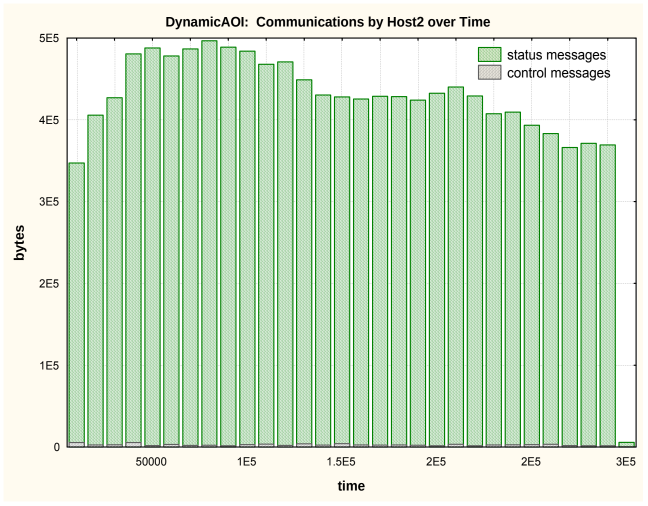
Figures 6a – 6f. Load (a. left, top), Cumulative Communications (b. right, top), and Communications Costs by Host (bottom four) of FixedAOI Policy

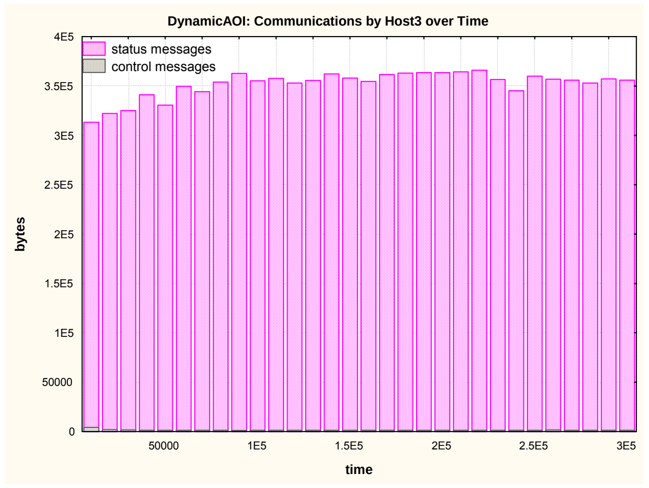












Figures 7a – 7f. Load (a. left, top), Cumulative Communications (b. right, top), and Communications Costs by Host (bottom four) of DynamicAOI Policy

1. Random: As seen in Figure 3a, this policy seems to work well at maintaining a stable load across machines. We attribute this in large part to the system implementation and experimentation paradigm we use. For example, movement of entities is not coordinated amongst entities and the direction vectors are based on random distributions. Given that all machines started with same number of randomly moving entities and all machine applied same random distribution for load balancing policies, it makes sense that this policy would perform well. For similar reasons, growth in cumulative bandwidth used (see Figure 3b) appears consistent across machines, as do numbers of status messages and control messages across time (see Figures 3c – 3f).

2. Lightest: As seen in Figure 4a, this policy is prone to large fluctuations in load. This oscillatory behavior stems from the fact that on each cycle entities get transferred according to which machine has the lowest overall load. Understandably, once entities transfer on one cycle, the machine with overall lowest load changes and then receives the block of entities on the next cycle. This notion is corroborated by high number of control messages relative to number of status messages (see Figures 4c - 4f). However, while this policy results in a high number of transfers, it also requires a lower amount of cumulative bandwidth (see Figure 4b). This is due to the fact that migration control messages for transfers (migration and acknowledge migration) are, in this system, cheaper on average than entity status messages. Thus, while requiring more transfers, this policy generally has a lower communications cost.

3. Closest: As seen in Figure 5a, this policy results in relatively high processor costs in comparison with the other policies. We believe this is due to the computational overhead spent on deriving nearest neighbor' center of mass. In the closest case, the center mass is associated with a processor, thus the entities that make up the group tend to be too. Therefore, as they move around, their center of mass moves, but they stay attached to the same processor. This concept is supported by the relatively small number of control messages as seen in Figures 5c – 5f. Also, as demonstrated in Figures (5c – 5f), this policy tends to perform bulk of transfers in the beginning of the run.

4. Fixed AOI: As suggested by Figure 6a, this policy also shows a relatively high processor cost and a markedly similar load behavior to the Closest policy (see Figure 5a). The interesting comparison in this case, however, is that the similarity behavior was achieved through the more consistent use of transfers (see Figures 6c – 6f).

5. Dynamic AOI: Lastly, as seen in Figure 7a, this policy seems to work fairly well at maintaining an "average" load across the machines. It also results in the highest communication costs in comparison to other policies, as seen in Figure 7b. Further, as supported in Figures 7c – 7f, these costs are attributed to entity status messages, and very few transfers are apparent. Again, we believe that the high communication costs can be explained by the non-uniformity (between) and continuous random movement (within) entities. That is, if entities in our simulation moved in groups with a deliberate direction, this policy would have likely performed more efficiently in comparison to the other policies considered.

To more closely examine the communications costs between policies we considered the numbers and types of messages multicast according to policy. These results are presented below in Figures 8 which

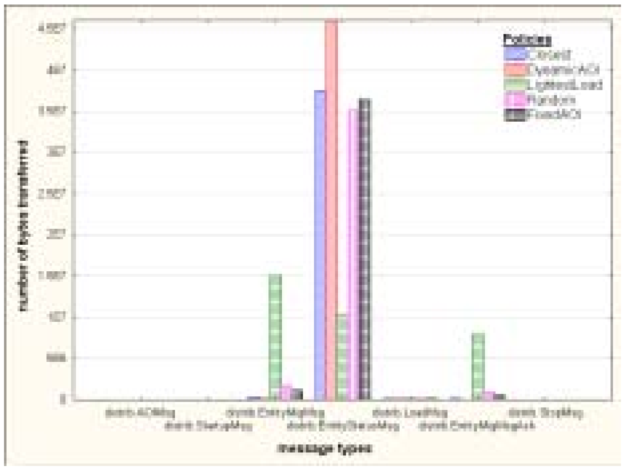


Figure 8. Number of Bytes Transferred by Message Type by Policy Type

corroborates results of preceding analyses and also communicates ratio of control messages resulting from entity transfers relative to control messages relative to exercise control or adjustment of AOI.

FUTURE WORK

Based on our results, there are a number of ways we can extend this work. First, we can implement unit-based movement algorithms in the system that are consistent with military tactical practices. A basic principle of military strategy is to concentrate forces in a small area. In battlefield simulations, this usually leads to a majority of the objects in one small part of the battlefield.

Second, we can implement a communications-based migration policy that attempts to move those objects that communicate frequently with the receiving host. This approach can potentially reduce the communication costs; however, this reduction would come at cost of creating extra overhead during the load migration process as each host must keep a communication lookup table to keep would be required to track of the frequency of communication between each local object and all other hosts and some sorting or searching algorithm must be implemented in order to select the most appropriate object for transferring.

Third, the approach to estimate workload by making use of the busy wait construct was a novel contribution. However, the combination of a time-series model in conjunction with this data may allow us to use it to forecast load for the next Δt interval instead of simply measuring over the just-ended interval. In future experiments we will enhance our information policy to determine whether using this metric in a predictive sense improves results.

Lastly, a number of researchers (Zomaya and The, 2001; Yu, Wu, and Hong, 1997; El-Abd and El-Bendary, 1997) have experimented with the use of intelligent control algorithms (e.g., genetic algorithms, neural networks, etc) to dynamically optimize migration policies. Given that our exercise was centered on use of HPC and that these control algorithms are computationally expensive and perhaps lend themselves to parallelism, it might be an interesting data point to capture the use of parallelism in this dimension as well.

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

Boon P., Yoke, H., Jain, S., Turner, S., Wentong, C., Wen, J., Shell, Y. (2000). Load balancing for conserva tive 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. 20-8.

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.

El-Abd, A., and El-Bendary, M. (1997). A Neural Network Approach for Dynamic Load Balancing in Homogenous Distributed Systems. In *Proceedings Proceedings of the 30th. Annual Hawaii International Conference on System Sciences*. pp. 628-9.

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. 318-27.

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. 725-30.

Nicols, D. (1987). Performance Issues for Distributed Battlefield Simulations. In *Proceedings of the 1987 Winter Simulation Conference*. pp. 624-8.

Niedringhaus, W. P. (1995). Diffusive dynamic load balancing by terrain parcel swaps for event-driven simulation of communicating vehicles. In *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. and Henninger, A. (2003). Load Balancing for Distributed Battlefield Simulations: Initial Results. *Proceedings of the Intersevice/Industry Training, Simulation, and Education Conference (I/ITSEC)*. Orlando, FL.

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. 590-96.

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.

Yu, K., Wu, S., and Hong, T. (1997). A load b alancing algorithm using prediction. *In IEEE Proceedings on Parallel Algorithms/Architecture Synthesis*. pp. 159-65.

Zomaya, A. and Tee, Y. (2001). Observations on Using Genetic Algorithms for Dynamic Load-Balancing. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, no. 9, pp 899-911.