The Contribution of Static and Dynamic Load Balancing In A Real-Time Distributed Air Defence Simulation

Mr Bernardt Duvenhage; Mr Jan J. Nel
Council for Scientific and Industrial Research (CSIR)
bduvenhage@csir.co.za, cnel@csir.co.za

Abstract. Simulations with a large number of model instances make use of parallel architectures to improve performance. When using such a parallel architecture a challenge is to effectively distribute the simulation objects across the processing platforms. Load balancing can be static (pre-execution), or dynamic (adaptively performed during execution). In this paper the authors explore the extent to which static load balancing can optimise the performance of a distributed parallel---conservative and 100Hz discrete time---simulation of an air defence system. The measure to which dynamic load balancing could further enhance the performance is then explored. Such knowledge forms the basis for further load balance research.

1. INTRODUCTION

The South African Council for Scientific and Industrial Research has recently been involved in acquisition decision support [8] for a Ground Based Air Defence (GBADS) acquisition program. To this end a synthetic combat environment, dubbed Virtual GBADS Demonstrator (VGD), has been developed.

The synthetic combat environment must often interact, *in real-time*, with human equipment operators or connect to real equipment such as systems facilitating situational awareness. Typical GBADS scenarios are of such a scale though, that more computing resources than are available from a single computing platform are required to reach real-time simulation performance. The simulation of the scenario is therefore parallelised across multiple computing platforms.

For the purpose of this paper, load balancing is defined as the process of optimising the distribution of the simulation of the scenario across a certain number of computing platforms; the goal being to reach the real-time performance threshold. Static load balancing attempts to distribute the simulation of the scenario during simulation setup based on a-priori knowledge of the runtime resources required for the simulation of the scenario.

Dynamic load balancing is able to complement the pre-existing knowledge with dynamically measured metrics of which the behaviour trace may be difficult to estimate at start-up. A metric may be defined as a parameter that can be measured. The dynamically measured metrics can then, at run-time, cause redistribution of the simulation of the elements of the scenario as required.

For static load balancing, this paper assumes that minimising the global execution time of a simulation optimises the local second to second execution as well. This assumption allows the execution efficiency of a distribution to be measured on the execution time of the simulation run. The assumption is not required for dynamic load balancing due to its nature of optimising local execution directly.

The next section of this paper elaborates on the distributed execution and communication architecture of VGD to aid in identifying the set of potential load measuring metrics to consider. Statistical round robin static load balancing is then applied to VGD and the best, average and worst case performance analysed. From the analysis and the available literature the appropriate metrics and a static load balancing heuristic is chosen and its performance compared to that of the statistical static load balancing. The dynamic suitability of the static load balancing schemes is finally evaluated to estimate the performance increase that dynamic load balancing could potentially provide.

2. VIRTUAL GBADS SIMULATOR

This section examines the details of VGD's custom architecture to identify and substantiate the potential loading metrics. The rationale behind the custom architecture and further details may be found in [4] and [7].

VGD incorporates a publish-subscribe communication architecture as the single way for scenario components to communicate. The publish - subscribe communication model is implemented on top of a peer-to-peer message passing architecture on a commercial gigabit Ethernet infrastructure. The synchronisation between processing platforms is performed at the level of the message passing, shown in Figure 1. Each processing platform (node) goes through an increment, publish, gather and synchronise phase. The gather phase includes both the reading of messages and also the network wait times for new messages to be received.

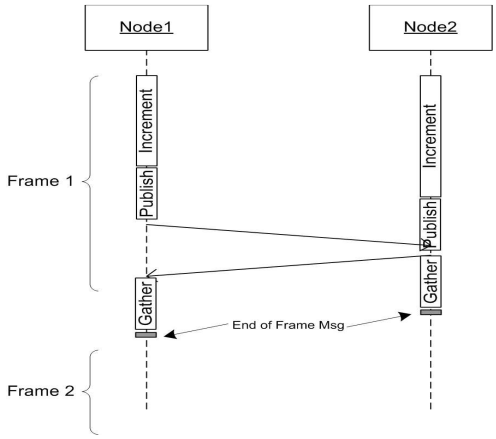

Figure 1: The Layered Simulator Architecture

Each node ends its publish phase by sending an "end-of-frame" message to all other nodes. A node may continue with the next frame once an "end-of-frame" message has been received from every other node. VGD therefore uses conservative time management and implements a 100Hz logical time Discrete Time System Specification (DTSS)[9][4]. The simulator may be run in As Fast As Possible (AFAP) mode or the logical time may be throttled to not execute faster than real-time.

The two initial loading metrics considered for this distributed conservative simulation are computational load and communication overhead. Additional loading metrics are identified in Section 3.4 below from the analysis of the statically distributed statistical runs. The benchmark scenario for the statistical runs a GBAD system performance experiments. It is a full multi-layer GBADS deployment designed to analyse the effectiveness of the air defence system against multiple waves of incoming threats.

3. STATIC LOAD BALANCING

This section elaborates on static load balancing as an optimisation problem of a loading function. The best, average and worst case static load distributions are found experimentally from a statistical sample of all possible distributions. A set of potential loading metrics are applied to these distributions to find a subset of metrics applicable to static load balancing and a static load balancing heuristic.

3.1 The Static Optimisation Problem

A simple way to perform static load balancing is to randomly assign the simulation of scenario components to the available processing nodes in a round robin fashion until all the scenario components have been assigned. Keeping in mind that the scenario components might have very dissimilar weights in computational requirements and information bandwidth, to name but two aspects, the reader might notice the following obvious short-falls with this approach:

- it might happen that some processing platforms are loaded heavier than others in terms of processing,
- some of the peer-to-peer communication channels may be overloaded while others are underutilised, or
- any of the other potential loading metrics may be detrimentally out of balance.

These lead to some processing platforms always finishing discrete simulation frames earlier than others or intermittently waiting for information to reach them. The conservative time management has the drawback that the frame-to-frame simulation execution efficiency is dependent on the slowest (or most heavily loaded) processing platform.

To find the fit static distribution is an NP problem (of time $O(c^n)$ for the number of objects) where a number of, possibly related, metrics have to be optimised. The optimisation problem is demonstrated with processing time and cumulative communication overhead in general. The execution time e_o of each object (scenario component) o excluding communication overhead can be measured, by running a simulation of the scenario once. Assuming a fixed scenario and similar nodes, e_o is fixed and independent of the object's position in the cluster of nodes. Unfortunately though, the communication overhead is a function f_o of, at least, the object's bandwidth requirements and the object's position in the cluster relative to the other objects. This makes it impractical to use empirical solutions of f_o for predictive purposes such as required for load-balancing.

If we rather use the function f_o, assuming that it is known and solvable, then finding the optimal solution for object distribution is equivalent to minimising the global function $F = \max\left(\sum_{o \in O_n} (e_o + f_o) \forall n \in N\right)$ where O_n is the set of objects on node n and N is the set of all nodes. The minimisation of F may be done by means of a computational intelligence paradigm such as particle swarm systems [6] or any other multi-variable optimisation technique.

To find f_o is unfortunately no easy task as it requires the accurate modelling of the latencies and bandwidth capabilities of the node interconnection infrastructure. The following subsections discuss the load balancing literature and analyse statistical distributions of the simulation of the benchmark scenario. This is done

with the goal of finding a static load balancing heuristic, the use of which approximates the optimum solution of *F* rather than finding the exact solution.

3.2 Load Balancing Literature

Static and dynamic load balancing, load metrics and distribution heuristics have been studied extensively. This section looks at some of the work that has been done for conservative peer-to-peer simulators.

Boukerche and Tropper [1] use a 'simulated annealing' algorithm to find good static distribution partitions that balance processor load and minimise inter-partition communication. Each partition is allocated to a processing node. Simulated annealing is a process of starting with many partitions and then gradually increasing the size of partitions until a pre-defined number of partitions is reached. The annealing process tries to gradually grow a good distribution.

A different static partitioning approach is followed by Carlisle and Merkle [3]. They use what they call 'domain knowledge' to pre-partition the scenario components into groups. This exploits pre-knowledge about the communication patterns such that communication between the groups are minimised. They then follow known load distribution strategies to load balance or anneal the pre-partitions.

The static load distribution approaches discussed have in common that a set of known physically based metrics are applied on a pre-run of the scenario and then used to load balance subsequent runs of the same or similar scenario. Dynamic load balancing has the advantage of not explicitly needing to know which physically based metrics to apply and a pre-run of the scenario is not necessarily required. The disadvantage is of coarse higher runtime overhead in assessing and correcting load imbalances.

Boukerche and Das [2] use the notion of processor queue length to dynamically measure load balance between processors. The load migration then takes place at run-time to optimise load balance. Furthermore the load to migrate is chosen in such a way as to minimise the communication overhead.

From the cited literature it does however seem that the choice and performance difference between static and dynamic, the loading metrics and the distribution heuristics are very application dependent. In particular, the influence on load distribution strategies on the specific type of simulator architecture under investigation is not apparent.

3.3 Statistical Static Load Balancing

This subsection discusses the statistical experiments in static load distribution. The simulator is instrumented with object and network processing load monitors, and a node idle time monitor. The simulator is run in As Fast As Possible Mode (AFAP).

A round robin approach is followed to distribute randomly shuffled scenario components to three separate processing platforms. A sample of thirty scenario distributions is generated in this way. The assumption is made that round robin distribution (approximately equal number of scenario entities per processing platform) creates a manageable, but representative subset, of the *processor balanced* distribution sample space. A second assumption is that the Ethernet throughput, and TCP's flow and congestion control is adequately managing the network data. The second assumption is supported by a measured stable, and real-time performance related, network throughput to and from each processing platform.

A third assumption is that the simulation behaviour is reasonably similar across the different scenario distributions. Differences may arise due to the processing platforms' pseudo random number sequences being applied differently for each unique scenario distribution. The time correlation between the weapon-target kills across the different distributions and the similar shapes of the performance graphs support the behaviour similarity assumption. See the lowered performance spikes encircled on the performance graphs for the best and worst performing statistical distributions in Figure 2 and Figure 3 respectively. Each weapon kill is the result of the air defence control behaviour and then a throw of a pseudo random dice biased by the kill probability. Each low performance spike is due to an audible kill feedback and short pause setup for this purpose.

The results of the performance ranked statistical distributions are shown in Table 1. Note that a negative time indicates being ahead of real-time. Scenario distribution 31 is a manually 'tweaked' distribution that is explained in Section 3.4.2. The ideal performing simulation distribution, according to the original static load balancing assumption[1], has a minimum execution time when run in AFAP mode. The performance histogram is shown in Figure 4. The best case statistical distribution (distribution 3) reached an un-throttled logical time execution performance of 188.39% real-time and the worst case statistical distribution (distribution 4) reached a logical time execution performance of 162.76% real-time.

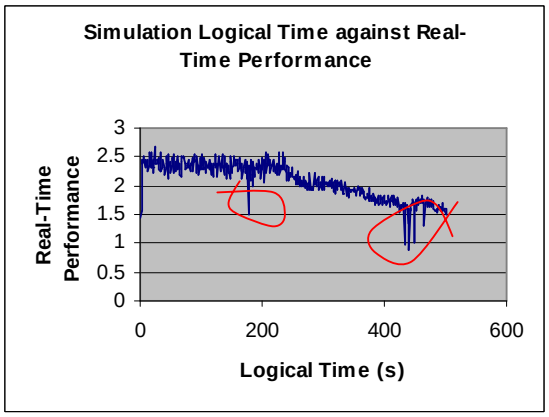

Figure 2: The Performance Graph of the Best Performing Statistical Distribution

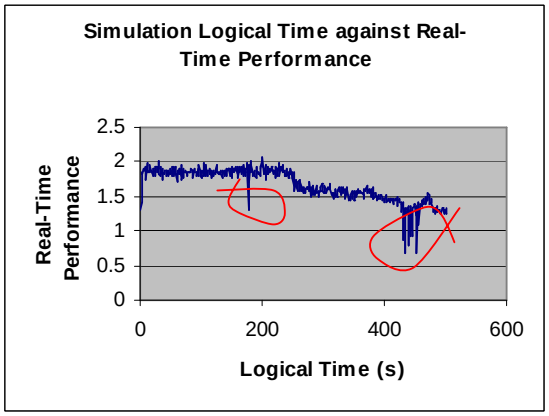

Figure 3: The Performance Graph of the Worst Performing Statistical Distribution

3.4 Finding a Heuristic for Static Load Balancing

This subsection analyses the statistical best, average and worst case simulation distributions according to an initial processor loading metric. Distribution performance anomalies are then explained by looking at additional metrics.

3.4.1 Analysis and Initial Heuristic

The authors reason that initially using a heuristic that purely balances processor usage, ignoring network overhead, gives reasonable performance.

This is due to two factors, nl:

- the network processing and transfer of data takes place in a separate operating system thread controlling a largely independent hardware communication subsystem[1], and
- balancing processor usage balances, on average, the number of scenario components and, consequently, on average the platform bandwidth.

[1]Adequate TCP send and receive buffers are allocated to minimise blocking sends and read wait times [4].

Table 1: Scenario Distribution Performance Results

Rank	Scenario Distribution		Time Behind Real-Time	Real-Time Performance
1	31		-253.453	202.80%
2	3		-234.593	188.39%
3	25		-234.115	188.05%
4	7		-232.445	186.88%
5	27		-231.513	186.23%
6	20		-231.489	186.21%
7	9		-230.827	185.75%
8	26		-230.55	185.56%
9	13		-229.354	184.74%
10	12		-228.497	184.16%
11	24		-227.328	183.37%
12	29		-227.274	183.33%
13	1		-226.167	182.59%
14	28		-224.904	181.75%
15	8		-224.378	181.41%
16	11		-224.288	181.35%
17	17		-222.894	180.44%
18	10		-221.628	179.62%
19	23		-220.989	179.20%
20	22		-220.797	179.08%
21	30		-219.649	178.35%
22	19		-218.939	177.90%
23	16		-217.724	177.13%
24	5		-216.644	176.46%
25	15		-215.114	175.51%
26	21		-209.187	171.93%
27	2		-208.946	171.79%
28	6		-206.334	170.26%
29	18		-205.383	169.71%
30	14		-194.066	163.43%
31	4		-192.79	162.76%
Average Performance				179.45%

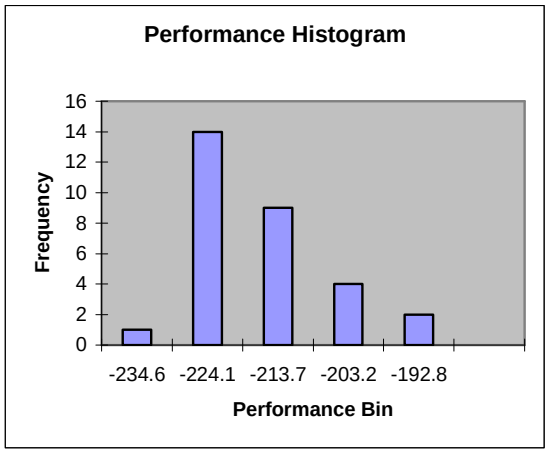

Figure 4: The Binned Performance Histogram

The load *balance* of the processing nodes is calculated by the difference between the nodes' maximum and minimum CPU time divided by the average CPU time.

Graphing (Figure 5a) the node performance balance, the initial heuristic seems to hold. A least squares line fitting process is followed to assess the R^2 value of fit of the metrics to the real-time performance. The R^2 value of fit to a direct proportionality of real-time performance to processor balance is 0.626. Doing load balancing purely on processor loading with a balance of 0.25 and better gets to within 98% of the statistical best real-time performance, which is 4% higher than

the average and 16% higher than the worst case statistical distribution.

From the analysis results it does seem as if the distribution performance is directly related (see Figure 5b, R^2 value of fit of 0.9961) to the frame execution time of the slowest processing platform. Looking at the simulator's time stepped design shown in Figure 2, a platform's time spent on a simulation frame is equal to the platform's combined:

- object increment and publish processing time,
- message send time,
- receive wait time, and
- receive message processing time (Gather) for that frame.

The increment and message processing times have been accounted for in the processor loading already instrumented. The measurable message send time only includes copying the data to the TCP send buffer, as mentioned[1], and has also already been included in the processor loading. A node's 'receive wait time' is the part of the TCP data transportation that could not happen in parallel with the increment and publish processing phase of the node's simulation frame. This is because there is nothing to increment or publish during such a wait time.

Analysing both the node send balance and the maximum 'Node Total Sent' (Figure 6) against real-time performance reveals the expected trend of lowered performance on increased bandwidth, but with a very low R^2 value of fit. This enforces the idea that the real-time performance, and thus the 'network wait time', is related to both processor balancing and network bandwidth and, quite possibly, other aspects as well.

3.4.2 Remaining Performance Anomalies

Scenario distribution 31 is distributed manually by dividing the scenario into object groups that minimise the communication between groups. In effect applying the domain knowledge as Carlisle and Merkle [3] discuss. These groups are processor balanced between the nodes and the distribution tweaked, by trial and error, until an optimum distribution is found.

Distribution 31 and the other performance outliers above the processor balance trend line seem to have a load balance closer to 0.37:0.34:0.29 than 0.33:0.33:0.33. Revisiting Figure 2 and the simulator architecture this behaviour may be linked to the specific conservative and peer-to-peer synchronisation mechanism and its implementation. It seems that to lower the network wait times, the node loading must be staggered. An optimum balance between not overloading the heaviest node and staggering the TCP data transportation might have to be found as mentioned again in the *Future Work* section.

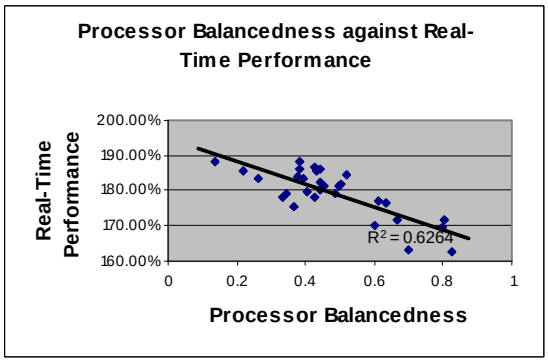

Figure 5a: The Processor Balance against Real-Time Performance

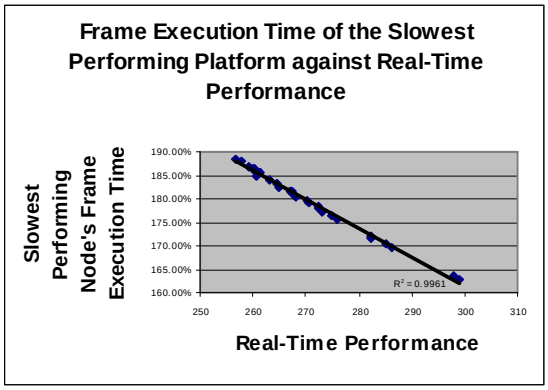

Figure 5b: Frame Execution Time of the Slowest Performing Platform against Real-Time Performance

4. DYNAMIC LOAD BALANCING

The loading metrics identified in the previous two sections are used to find the dynamic suitability of both the statistical best case and the heuristic driven static load balancing. The advantages of dynamic load balancing compared to static load balancing and the potential metrics and dynamic load balancing heuristics are then explored.

4.1 The Dynamic Optimisation Problem

The major difference in doing load balancing dynamically is that a load balancing heuristic is applied to gradually and adaptively optimise the run-time simulation performance. In effect:

- observe current performance,
- decide what load to tentatively migrate to increase the simulation performance, and
- act by intelligently migrating the load.

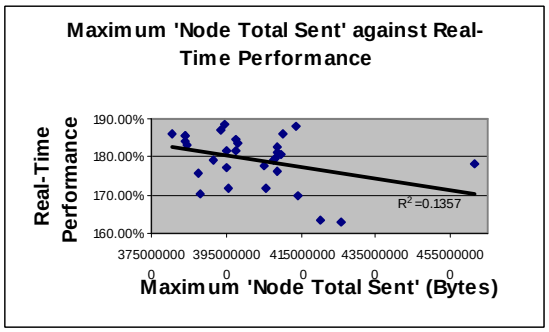

Figure 6: The Maximum 'Node Total Sent" against Real-Time Performance

4.2 The Dynamic Suitability of Static Load Balancing

Figure 7 represents a typical scenario distribution on each of the three processing nodes. The graph indicates cumulative increment and publish times for all the objects. The derivative (gradient) of the graph therefore indicates the instantaneous processing load of the associated object's increment or publish phases. It is clear that the nodes contain objects that adapt their processing load. The dynamic suitability of the static load distribution is analysed by ranking the distributions according to performance over the first half, called phase 1, of the simulation and then again for the second half, called phase 2. These results are shown in Table 2.

The distribution rankings do differ quite a bit between the two phases. This indicates that the relative dynamic suitability of the static distributions does vary. The performance impact is minimal, though. The very good performance of distribution 31, across both phase one and phase two, indicates to the potential dynamic suitability of a static distribution.

4.3 The Potential Performance Increase of Dynamic Load Balancing

The manually balanced distribution 31 is still the best performer, but the best statistical static distribution has changed from 3 to (25+26) for the two phased load balancing shown in Table 2. Even though the ranking between the two phases is considerably different the overall performance increase of the best case phased distribution is a mere 0.66% as shown in Table 3. This indicates to a rather poor potential performance increase for dynamic load balancing.

5. CONCLUSION

Statistical round–robin distribution---with no attempt at applying a load balancing heuristic---results, at worst, in an execution performance that is within 82% of the statistical best case distribution and within 75% of the best case manually tweaked distribution. The current analysis results indicate that a simple CPU load balance heuristic driven static load reaches within 98% of the statistical best case distribution and within 91% of the best case manually tweaked load distribution which is a 10% improvement on random round-robin distribution.

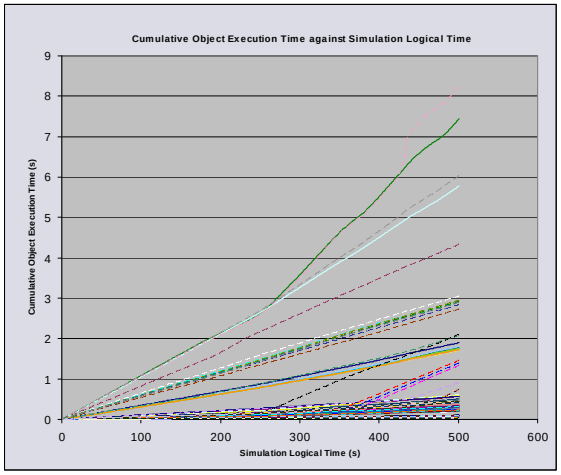

Figure 7: Cumulative Object Execution Times of the Objects on Node 1

Table 2: Dynamic Suitability of Static Load Distributions

Rank	Scenario Distribution	Time Behind Real-Time (Phase 1)	Scenario Distribution	Time Behind Real-Time (Phase 2)
1	31	-152.717	31	-100.736
2	26	-142.104	25	-94.048
3	3	-141.088	3	-93.505
4	7	-140.449	13	-92.513
5	9	-140.278	27	-92.484
6	25	-140.067	7	-91.996
7	20	-139.699	20	-91.79
8	27	-139.029	9	-90.549
9	12	-138.482	12	-90.015
10	29	-138.397	28	-89.519
11	24	-138.099	1	-89.239
12	8	-137.598	24	-89.229
13	1	-136.928	29	-88.877
14	13	-136.841	11	-88.525
15	17	-136.433	26	-88.446
16	15	-136.109	23	-87.932
17	11	-135.763	22	-87.833
18	5	-135.5	30	-87.403
19	28	-135.385	8	-86.78
20	10	-135.066	10	-86.562
21	19	-134.548	17	-86.461
22	16	-134.611	19	-84.391
23	23	-133.057	16	-83.113
24	22	-132.964	5	-81.144
25	30	-132.246	2	-80.357
26	21	-129.028	21	-80.159
27	2	-128.589	6	-79.485
28	18	-126.818	15	-79.005
29	6	-126.849	18	-78.565
30	4	-122.631	14	-71.989
31	14	-122.077	4	-70.159

The performance improvement of the best case statistical distribution over the worst case is 15.75% which is less than the measured parallelisation performance gain of 30% [4] when parallelising the execution of the scenario over four simulator nodes instead of three. The cost of finding the optimum load balancing heuristic must therefore carefully be weighed against the cost of adding another simulator node. In this particular architecture, the performance increase

due to load balancing only exceeds that of adding a node once the simulation reaches a node count of about 8 to 10. At this point the performance of the distributed simulator architecture time management becomes a limiting factor for increasing the node count further.

Table 3: Dynamic Suitability of Two Phase Static Load Distributions

Scenario Distribution (Phase 1)	Scenario Distribution (Phase 2)	Combined Two Phase Real-Time Performance	Improvement over Single Phase Distribution
31	31	-253.453	100.00%
26	25	-236.152	100.66%
3	3	-234.593	100.20%
7	13	-232.962	100.22%
9	27	-232.762	100.54%
25	7	-232.063	100.25%
20	20	-231.489	100.29%
27	9	-229.578	99.58%
12	12	-228.497	99.63%
29	28	-227.916	99.75%
24	1	-227.338	100.00%
8	24	-226.827	99.80%
1	29	-225.805	99.84%
13	11	-225.366	100.21%
17	26	-224.879	100.22%
15	23	-224.041	99.89%
11	22	-223.596	100.31%
5	30	-222.903	100.58%
28	8	-222.165	100.53%
10	10	-221.628	100.38%
19	17	-221.009	100.62%
16	19	-219.002	100.03%
23	16	-216.17	99.29%
22	5	-214.108	98.83%
30	2	-212.603	98.83%
21	21	-209.187	100.00%
2	6	-208.074	99.58%
18	15	-205.823	99.75%
6	18	-205.414	100.02%
4	14	-194.62	100.29%
14	4	-192.236	99.71%

The overhead associated with the dynamic load balancing measurements and the run-time migration of load may nullify the potential performance increase of dynamic load balancing over a reasonably good static load distribution. The different systems of the GBADS battery goes through increased loading phases at similar times as waves of targets come in. The main argument for using only static load balancing may then be that there are not many systems being spawned dynamically and that the existing distribution stays balanced.

It is however the case that through analysis it was discovered that staggering the processor loading could be advantageous. Making use of dynamic load balancing---which is not based on a pre-conceived notion of what impacts performance---therefore still has an advantage. When making use of a large variety of metrics, dynamic load balancing might create the opportunity to automatically explore and exploit other such, possibly as yet unknown, performance drivers.

6. FUTURE WORK

It is hypothesised that the nodes must be arranged in the staggered loading ratio to increase the performance. This should be investigated further. One suggestion is to run the same experiments, but with a statistical sample of scenarios that is better representative of non-processor balanced distributions as well.

Another important future task is to do research on bringing the network wait time down, possibly through interleaving the messages with the execution in an innovative way. Currently the network is mostly utilised in spikes of activity.

Similar performance experiments should also be run on simulations which require distribution over 8 or more nodes such that the effect of load balancing may be measured when the network is stressed.

REFERENCES

1. Boukerche, A & Tropper, C. (1994) "A Static Partitioning and Mapping Algorithm for Conservative Parallel Simulations," *Proceedings of the eighth workshop on Parallel and distributed simulation*, pp 164-172.
2. Boukerche, A & Das, S. (1997) "Dynamic Load Balancing Strategies for Conservative Parallel Simulations," *Proceedings of the eleventh workshop on Parallel and distributed simulation*, pp 20-28.
3. Carlisle, M & Merkle, L. (2004) "Automated Load Balancing of a Missile Defence Simulation Using Domain Knowledge," *JDMS*, Vol. 1, Issue 1, April, pp 59-68.
4. Duvenhage, B & Kourie, D. (2007) "Migrating to a Real-Time Distributed Parallel Simulator Architecture," *Proceedings of the 2007 Summer Computer Simulation Conference*.
5. El-Khatib, K & Tropper, C. (1999) "On Metrics for the Dynamic Load Balancing of Optimistic Simulations," *HICSS*.
6. Engelbrecht, A. (2002) "Computational Intelligence, An Introduction," John Wiley.
7. le Roux, W. (2006) "Implementing a low cost distributed architecture for real-time behavioural modelling and simulation," *Proceedings of the 2006 European Simulation Interoperability Workshop.*
8. Naidoo, S & Nel, J. (2006) "Modeling and Simulation of a Ground Based Air Defense System and Associated Tactical Doctrine as Part of Acquisition Support," *Proceedings of the 2006 Fall Simulation Interoperability Workshop.*
9. Zeigler, B & Kim, T & Praehofer, H. (2000) "Theory of Modeling and Simulation, Second Edition" Academic Press.