Journal of Interconnection Networks

An Adaptive Dynamic Load Balancing Technique for Grid-Based Large Scale Distributed Simulations*

Azzedine Boukerche, Elie El Ajaltouni, Ming Zhang, and Robson Eduardo De Grande
School of Information and Technology Engineering
University of Ottawa
800 King Edward Avenue
Ottawa, K1N 6N5, Canada
{*boukerch,eelaj094,mizhang,rdgrande*}*@site.uottawa.ca*
http://<paradise.site.uottawa.ca>

Dynamic load balancing is a key factor in achieving high performance for large scale distributed simulations on grid infrastructures. In a grid environment, the available resources and the simulation's computation and communication behavior may experience critical run-time imbalances. Consequently, an initial static partitioning should be combined with a dynamic load balancing scheme to ensure the high performance of the distributed simulation. In this paper, we propose a dynamic load balancing scheme for distributed simulations on a grid infrastructure. Our scheme is composed of an online network analyzing service coupled with monitoring agents and a run-time model repartitioning service. We present a hierarchical scalable adaptive JXTA service based scheme and use simulation experiments to demonstrate that our proposed scheme exhibits better performance in terms of simulation execution time. Furthermore, we extend our algorithm from a local intra-cluster algorithm to a global inter-cluster algorithm and we consider the proposed global design through a formalized Discrete Event System Specification (DEVS) model system.

Keywords: Grid; JXTA; Dynamic Load Balancing; DEVS formalism.

1. Introduction

Running a large-scale distributed simulation generally requires a large amount of computing resources that are at geographically dispersed locations. The imbalance of workload in a distributed simulation is one of the key factors that lead to the degradation of the overall simulation execution performance. Considering the hard-to-predict run time behaviors of most simulation applications, static workload partition mechanisms cannot, in many cases, work effectively. Consequently, the dynamic load balancing mechanisms are called on, as these aim to achieve performance improvement by balancing the workload during simulation run-time. Dynamic load balancing schemes use run-time system state information to make decisions on how

*This work is partially supported by Canada Research Chair Program, NSERC, Ontario Distinguished Researcher Award, Ontario Early Researcher Award, EAR Award and ORF Funds.

to move work from one host to another during execution. However, overhead due to the dynamic migration of simulation entities needs to be carefully considered in order to see positive performance improvement. As we know, High Level Architecture (HLA) [2–4] has been the dominant distributed simulation standard for the past few years. However, HLA relies on the Runtime Infrastructure (RTI), which limits its flexibility and scalability. Also, there are no built-in fault-tolerance or load balancing mechanisms in HLA. In order to address these problems, much research has aimed to enhance HLA/RTI by adding load balancing capabilities through federate migration. Some other approaches have focused on using the concept of Service Oriented Architecture (SOA) to encapsulate the core functionalities of HLA as Grid services. In the meantime, peer-to-peer based technology, such as Sun's JXTA, has also been used as middleware to support distributed simulation, as it can overcome some of the existing difficulties in HLA and other distributed simulation frameworks. In this paper, we propose a dynamic load balancing scheme in the context of a JXTA based distributed simulation framework [1]. Our algorithm focuses on several JXTA services, including a time management service, an on-line network analyzing service and a run-time model repartitioning service. We use monitoring agents to collect runtime system state data, and use an on-line network analyzing service to achieve a dynamic load balancing strategy. Finally, we use the run-time model repartitioning service to apply the dynamically generated load balancing strategy to the system. In our proposed algorithm, both computation and communication workloads are considered as sources of potential imbalance, affecting the performance of the distributed simulation. Moreover, our proposed scheme follows the hierarchical design approach to meet the requirement of large-scale grid based distributed simulation applications. As a mater of fact, in this paper, we propose and demonstrate a dynamic load balancing scheme for more efficient distributed simulations. We will verify our idea through simulation experiments conducted on an IBM Linux cluster located at PARADISE Lab. The remainder of this paper is organized as follows: Section 2 presents related and previous work. Section 3 describes our proposed dynamic load balancing scheme in detail. Section 4 shows our simulation experimental results. Section 5 extends our local intra-cluster scheme into a global inter-cluster scheme. Section 6 studies the proposed global design through a formalized Discrete Event System Specification (DEVS) model system. Finally, we conclude this paper and state some future work in Section 7.

2. Related Work

Dynamic load balancing for distributed simulations has received much interest due to its potential ability to enhance performance. Many algorithms [5-22] have been developed to tackle the problem of distributed simulations running on different workstations, clusters and the grid. The authors in [5] propose an algorithm that uses two central metrics: The processor advance time (PAT) and the cluster (of LPs) advance time (CAT) which are the amount of wall clock time required by a processor

and a cluster respectively to advance one unit of simulated time in the absence of rollback. The load balancing algorithm first sorts the usable set of processors based on their PAT values and then traverses the sorted set, migrating clusters of LPs from processors with large PAT values to those with lower PAT values in order to minimize the maximum difference between the PAT values in any two processors. The clusters to migrate are chosen by CAT which defines the amount of computation required to advance a cluster one unit of simulated time in the absence of rollback. Moreover, the authors in [6] extends the algorithm presented in [5] to take communication latencies into account. The authors defines two additional metrics to be considered for the algorithm's migration decision: the Processor-to-Processor Communication and Cluster-to-Processor Communication. These indicate the total number of messages exchanged between two processors (between any LPs) and between a cluster and a processor (any LP on that processor) respectively within a time interval. Most of the research in [9] [10,11] uses the grid services and the HLA / RTI for distributed simulations creating a middleware between the HLA and the grid services. The authors in [13] developed a new dynamic partitioning algorithm based on performance estimates whose calculation is completely platform independent. Moreover, aging functions were defined to increase the robustness of the algorithm, and a set of parameters allowed user configuration to meet specific scenarios. However, their host capacity estimate implicitly assumed that event execution has a uniform cost. The authors in [12] perform dynamic load balancing in two steps. First, a central controller collects information about the pending load distribution and the variance in CPU usage for the participating hosts. Then, it determines the amount of load to transfer between hosts. In a second step, the hosts decide which simulation objects to move according to different policies. A communication-based policy may be selected. However, none of their different policies defined to decide on which simulation object to move consider the combined computation and communication implications. The authors in [14] propose a dynamic load balancing algorithm based on a defined cost model. It showed using different simulation models that to achieve consistent performance imbalances, as well as lookahead between processors need to be considered. Furthermore, the authors in [15] proposed a dynamic load balancing algorithm which assumes no knowledge about the workload parameters based on the CPU-queue length. Two variations of the algorithm were designed: the centralized and multi-level methods. The centralized approach makes use of a central processor to collect and disseminate the loads for each processor while the hierarchical method use a central approach at level one and a distributed strategy at level two, where processors communicate and share LPs' information only with their immediate neighbors. It is worth mentioning that a new metric for measuring the workload is defined in [16] for both optimistic and conservative simulations. In their approach, an architecture for load distribution services over a multi-threaded HLA / RTI is described, and these services aimed to improve the overall simulation performance. The authors in [7] and [8] introduce agents for achieving load balancing over

the grid. [7] uses a combination of intelligent agents and multi-agent approach. The agents are used as representatives of grid resources and utilize predictive application performance data and iterative heuristic algorithms to engineer load balancing across multiple hosts. In [8] the agents are responsible for sensing the context of the Grid, and serve as the scheduler for dynamic load adjustment. The authors in [19] described a dynamic load balancing algorithm consisting of three components: (1) load sensor (2) load evaluator and (3) load adapter. The load sensor computes the Virtual Time Progress(VTP) for each simulation process. The load evaluator decides whether to launch process migration or not, depending on ratio between the actual and the predicted VTP. Process migration is then controlled by the load adapter. The algorithm aims at reducing the lag between the VTPs of the processors for a better simulation performance. Burdorf and Marti [20] initially presents a static balancer that assigns objects to processors according to the load, which is gathered by running a pre-simulation. During the simulation, the balancer records the smallest simulation time of all objects on each machine and seeks to minimize the variance between the objects' simulation times. The work in [21] studies both static partitioning and dynamic load balancing strategies. The static partitioning strategy attempts to minimize load imbalances. The dynamic load balancing scheme works together with the static partitioning strategy and tries to schedule LPs dynamically to threads. The authors in [22] compare the performance of three metrics for use with a token-based dynamic load balancing algorithm. The three defined metrics were processor utilization (PU), processor advance simulation rate (PASR), and a combination of these metrics. We refer to [17] [18] for another approach to load balancing through interest management.

We conclude our related work section by outlining the differences between our scheme and other dynamic load balancing schemes for distributed simulations. We argue that a dynamic load balancing scheme should optimize both computation and communication workloads. The performance approximation metrics we used are completely platform independent and are based on the simulation entities' activity with no priori knowledge needed. As a matter of fact, none of the algorithms we have surveyed provides an adaptive load differentiation mechanism based on the simulation entities' dynamic behavior. Therefore, in our approach, priorities are introduced to the different computation and communication load-balancing phases and are dynamically modified to meet the requirements of the distributed simulation. Indeed, our proposed scheme coupled with the system architecture ensures a scalable hierarchical design: a local intra-cluster load balancing scheme and its extension into a global inter-cluster load balancing scheme. Moreover, our algorithm is JXTA service based, and such modular design provides independence among the different simulation services, favoring any future service or architecture extendibility. These characteristics will be clear after the description of the proposed design in Section 3.

3. Proposed Dynamic Load Balancing Scheme

In this section we present our dynamic load balancing scheme in detail. We first outline our system architecture on which our load balancing scheme is designed. We then present and discuss our intra-cluster algorithm.

3.1. *System Architecture*

Our architecture is illustrated in Figure 1, in which each individual cluster is formed by a number of peer nodes. Every node has one or more simulation entities running with a monitoring agent. Each cluster has two local head nodes to run the key services (time management service, online network analyzing service, run-time model repartitioning service) used by our scheme and a global monitoring agent. The local head nodes are responsible for the load balancing of the intra-cluster distributed simulation. Fault tolerance is realized through the duplicate local head [1]. All the services and monitoring agents are introduced in both local and global levels. Two global head nodes located at one of the participating clusters run the global key services (global time management service, global online network analyzing service, global runtime model repartitioning service). The global head nodes combine with the local head nodes to form the global virtual cluster. The global head nodes are responsible for the overall load balancing of the inter-cluster distributed simulation.

3.2. *Structure of the Algorithm*

Our algorithm assumes that recent "past behavior" reflects that of the near future. Therefore, the monitoring agents introduced will monitor the simulation entities current activity within a certain interval. Based on the analysis of the observed data, a simulation reconfiguration will be decided. The on-line network analyzing service (OLNAS) gathers the monitoring agents observed data, analyzes the intra-cluster computing and communication workload distribution among the simulation entities. The run-time model repartitioning service (RTMRPS) is the key to run-time simulation reconfiguration. It handles the repartitioning of the simulation entities within the cluster. The OLNAS triggers the load balancing by sending a request to all the monitoring agents to collect their observed data upon the elapse of a waiting time interval Δ. The monitoring agents will send their monitored data to the OLNAS, reset their monitoring mechanism and stop their observation policies. Upon receiving the data from the monitoring agents the OLNAS performs a thorough analysis and forwards its analysis results to the RTMRPS. The RTMRPS makes the migration decision based on its analysis. Before starting the migration, the RTMRPS communicates with the STMS to pause the simulation during migration. When the migration is done, the RTMRPS notifies the STMS to resume the simulation and notifies the OLNAS to trigger the monitoring agents back to work. Our algorithm is limited to one migration per session, due to the migration overhead for the whole simulation, and we aim to find the migration that will give us the best possible re-

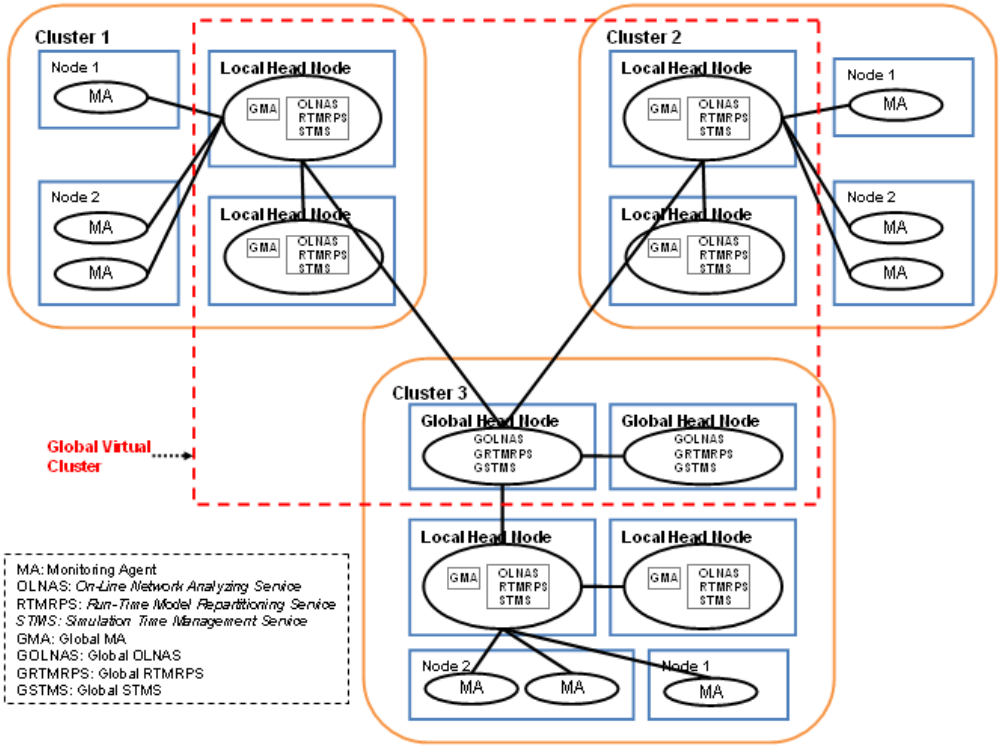

Fig. 1. High Level System Architecture

sult. Figure 2 shows the load balancing sequence diagram. In the following sections, we will describe in detail the three major components of our load balancing algorithm; the monitoring agent, on-line network analyzing service and run-time model repartitioning service. Every component has two phases: a computation phase and a communication phase. Although these phases are described separately, they are interdependent, and the final outcome of our algorithm takes both phases into consideration.

3.2.1. *Monitoring Agent*

In this section we describe in details the monitoring agent's computation and communication monitoring mechanisms and its respective output to the online network analyzing service.

Computation Monitoring Mechanism: The computation workload monitoring mechanism is based on the simulation entity's processing activity. It is important to note that our simulation entities are implemented as multi-threaded components. The processing activity can be approximated by measuring the CPU usage of the simulation entity. To obtain the time spent on the CPU by a specific simulation entity thread, we use ThreadMXBean interface within Java's java.lang.management

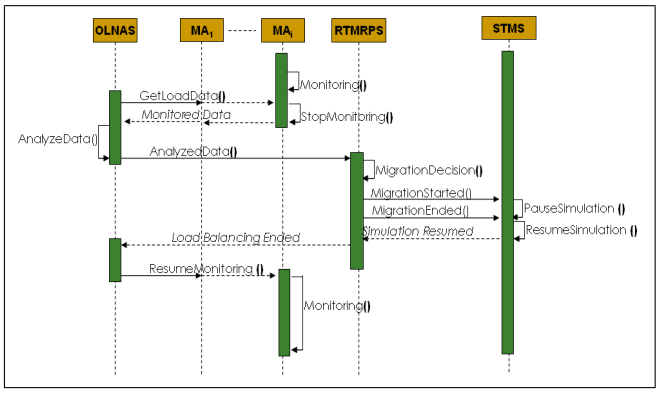

Fig. 2. **Load Balancing Sequence Diagram**

package. The thread identifier and CPU processing time can thus be collected. We use the method getThreadCpuTime() to obtain a good approximation of the observed CPU computation processing time (Φ). To avoid overly abrupt variations, we introduce a smoothing mechanism that balances the previous simulation entity load value with the current simulation load value. The smoothed simulation entity load value ($Load_{SE}$) is calculated in the following way:

$$Load_{SE_i} = (1 - \alpha).Load_{SE_{i-1}} + \alpha.\Phi_i \tag{3.1}$$

Where:

i is the current load balancing phase

$0.1 < \alpha < 1$

$\alpha = \frac{Load_i}{Load_{i-1}}$ if $Load_i < Load_{i-1}$

$\alpha = \frac{Load_{i-1}}{Load_i}$ if $Load_i > Load_{i-1}$

The greater α is, the smaller the value between the past and current load. Moreover, a node workload ($Load_N$) is defined as the sum of all its simulation entities workload:

$$Load_{N_i} = \sum_{SE \in N_i} Load_{SE} \tag{3.2}$$

Communication Monitoring Mechanism: The communication workload monitoring mechanism is based on introducing communication log tables. Every message initiated by the simulation entity updates the communication log table. The table entries store the message length in bytes as well as message's destination. Analyzing the table gives a good approximation of the simulation entity's communication behavior. Basically, it clearly indicates whether the simulation entity communicates more with foreign hosts rather than with its local host. A simulation entity having more foreign communication is considered to have low host binding while a simulation entity with more local communication is considered to have high host binding.

Monitoring Agent Output: Upon receiving the OLNAS request for the observed data, the monitoring agent first calculates the workload of the simulation entity, as defined in (1). Next, it responds to the OLNAS by sending information about the simulation entity (simulation entity id, host name and the size of objects controlled by the simulation entity), the communication log table and the calculated workload.

3.2.2. *On-line Network Analyzing Service*

In this section we describe in details the online network analyzing service's computation and communication data analysis and its respective output to the run-time model repartitioning service.

Computation Analysis: The OLNAS computation analysis aims to:

- Classify the nodes as overloaded, balanced and under-loaded
- Identify the highest workload generating simulation entities within each overloaded node

For node classification, it calculates every node load as defined in (2), the average μ of all the nodes' load and their respective standard deviation δ. According to Chebyshev's inequality [34], for nearly all random distributions, about 68 % of the values are within a single standard deviation of the mean. Consequently, the node classification is based on the following:

- Overloaded nodes are those whose load is greater $\mu + \delta$
- Under-loaded nodes are nodes whose load is less than μ - δ
- Balanced nodes has a load equal or in between μ - δ and $\mu + \delta$

Then, the OLNAS identifies the simulation entities with highest workload within the set of overloaded nodes.

Communication Analysis: The OLNAS communication analysis aims to:

- Identify simulation entities with high foreign communication

The communication analysis is carried out on the communication log table by calculating the difference between foreign and local calls. Simulation entities having high foreign communication are said to have low host binding while those with high local communication are said to have high host binding. The analysis aims to locate simulation entities that have low host binding and can be possible migration candidates. Nevertheless, a simulation entity may experience a slightly more foreign than local communication, and its migration will have a negative impact on the simulation performance. We therefore consider simulation entities with low host binding those which have foreign calls double or more than local calls. It is worth

mentioning that our selection for the *double* parameter has no major impact on our algorithm behavior and performance since what matters is to have significantly more foreign calls than local calls. Eventually, simulation entities that have foreign calls less than double their local calls will be considered as high host binding.

On-line Network Analyzing Service Output: At the end of the analysis, the OLNAS will have 4 possible outcomes: two possible migration candidates a computation and a communication one, one possible computation migration candidates, one possible communication migration candidates or none. If no possible migration candidates are found, then our algorithm will stop here. However, if a possible set of migration candidates is found then it will be sent to the RTMRPS with all the analyzed data.

3.2.3. *Run-Time Model Repartitioning Service*

In this section we describe in details the run-time model repartitioning service's computation and communication migration decision and the final migration decision.

Computation Migration Decision: The computation migration decision considers the identified possible computation migration candidates by the OLNAS which are basically the high workload generating simulation entities within the overloaded nodes. As we have mentioned earlier our purpose is to have the single migration that will yield the highest gain. We will run a first set of conditions on the possible migration candidates and if more than one possible migration candidate is left we will run another set of conditions. The first set of conditions is the following:

- The simulation entity has more foreign than local communication
- The simulation entity migration must not overload the destination node

The latter condition can be easily calculated by adding the simulation entity workload to the destination node's workload and by checking whether the destination node becomes overloaded. If more than one possible computation migration candidate satisfies the first set of conditions, we have to further reduce our selection by considering a second set of conditions:

- The simulation entity has high communication with one of the under-loaded nodes
- The simulation entity migration will induce the least migration latency(predicted by the size of objects it has)

Consequently, the set of possible computation migration candidates that passes the two set of conditions will be referred to as the computation migration candidates set.

Communication Migration Decision: The communication migration decision considers the identified possible communication migration candidates by the OLNAS which are basically the simulation entities with high foreign communication. With a similar methodology and argument for the computation migration decision we run two set of conditions on the possible communication migration candidates. The first set of conditions is the following:

- The simulation entity migration will not cause the overloading of the destination node
- The simulation entity migration will keep its host node within a balanced state

If more than one possible migration candidate meets the first set of conditions, we try to further reduce our selection by running a second condition:

- The simulation entity migration will induce the least migration latency

Consequently, the set of possible communication migration candidates that passes the two set of conditions will be referred to as the communication migration candidates set.

Run-Time Model Repartitioning Service Output: By running the computation and communication migration decisions the RTMRPS is left with a computation migration candidate set and a communication migration candidate set. There are four possible combination cardinalities for these two sets which we need to carefully consider. Based on each combination a different execution path and set of procedures will be executed by the RTMRPS. The four possible combination cardinalities are:

1. The cardinality of the computation migration candidate set and the communication migration candidate set are both greater than or equal to 1

The set with the higher priority will be selected and if it has more than one migration candidate a random simulation entity will be picked. The priority scheme will be explained in detail in Section 3.3.

2. The cardinality of the computation migration candidate set is greater than or equal to 1 while the cardinality of the communication migration candidate set is 0

A simulation entity will be randomly picked from the computation migration candidate set.

3. The cardinality of the communication migration candidate set is greater than or equal to 1 while the cardinality of the computation mi-

gration candidate set is 0

A simulation entity will be randomly picked from the communication migration candidate set.

4. The cardinality of the computation migration candidate set and the communication migration candidate set are both equal to 0

Reaching such a case means that we have a load imbalance locally however any local migration will not resolve the imbalance. Therefore, in such a case we trigger our global load balancing to help us resolve the local imbalances.

Finally, when the RTMRPS migration decision has been settled on a specific simulation entity, the migration process can be initiated. Before starting the migration, the RTMRPS communicates with the STMS to pause the simulation during the migration. When the migration is complete the RTMRPS notifies the STMS to resume the simulation and notifies the OLNAS to trigger the monitoring agents back to work. The basic concepts of the local intra-cluster algorithm are outlined in Algorithm 1.

3.3. *Algorithm Adaptation*

The worst-case scenarios for our algorithm involve a well-balanced simulation or a simulation that experiences only one kind of imbalance, either a computation or a communication one. Running our algorithm with such scenarios will induce an overhead that is not needed and will potentially degrade the simulation performance. To resolve this issue we add to our design two adaptation mechanisms that can detect such scenarios and adjust accordingly to our algorithm procedure and monitoring policies. Our first adaptation mechanism is for the OLNAS waiting time interval (Δ). Δ is initially set to a small value. If a migration decision is reached it is halved and if none is reached then it is doubled. It is worth mentioning that our selection for the *double* and *half* parameters has no major impact on our algorithm behavior and performance, since what matters is to have a significant decrease of the waiting time interval when there are frequent load imbalances and a significant increase of the waiting time interval when there are no load imbalances. As a result, with a well-balanced simulation, our algorithm waiting time interval will keep on increasing and eventually reducing its unnecessary overhead. Our second adaptation mechanism considers prioritizing the computation and communication imbalances. The RTMRPS besides doing the migration decision ensures an adaptive load differentiation mechanism based on the simulation dynamic behavior. Priorities are introduced to the different computation and communication workload and are dynamically modified to meet the requirements of our simulation. The argument here is that a simulation at run-time can experience either computation or communication imbalances for a long period of time. Consequently, it is preferable during this period to give either one a higher priority over the other. Furthermore, skipping the policies to detect one of the imbalances within this period can reduce unnecessary overhead.

Algorithm 1 Local Intra-Cluster Load Balancing Algorithm

Δ: OLNAS waiting time interval
SE_{ijk}: i^{th} simulation entity group of j^{th} node of k^{th} cluster
MA_{ijk}: i^{th} monitoring agent of j^{th} node of k^{th} cluster
N_{ij}: i^{th} node of j^{th} cluster
$Load(SE_{ijk})$: load of i^{th} simulation entity group of j^{th} node of k^{th} cluster
$Load(N_{ij})$: load of i^{th} node of j^{th} cluster
μ: average of simulation entity loads
σ: standard deviation of simulation entity loads
ON: set of overloaded nodes
BN: set of balanced nodes
UN: set of under-loaded nodes
$PPMC$: set of possible computation migration candidates of simulation entities
$PCMC$: set of possible communication migration candidates of simulation entities
PMC: set of computation migration candidates of simulation entities
CMC: set of communication migration candidates of simulation entities
BEGIN
ON $\leftarrow \emptyset$, BN $\leftarrow \emptyset$, UN $\leftarrow \emptyset$, PPMC $\leftarrow \emptyset$, PCMC $\leftarrow \emptyset$, PMC $\leftarrow \emptyset$, CMC $\leftarrow \emptyset$
wait (Δ)
for *all* MA_ijk **do**
  Send monitored data to OLNAS
end for
OLNAS data analysis:
Calculate the load of all nodes, Calculate μ and σ
for *all* N_{ij} **do**
  if $N_{ij} > \mu + \sigma$ **then**
    ON $\leftarrow$ ON $\cup \{N_{ij}\}$
  else if $N_{ij} < \mu - \sigma$ **then**
    UN $\leftarrow$ UN $\cup \{N_{ij}\}$
  else
    BN $\leftarrow$ BN $\cup \{N_{ij}\}$
  end if
end for
for *every node in ON* **do**
  Identify SE_{ijk} with highest workload
  PPMC $\leftarrow$ PPMC $\cup \{SE_{ijk}\}$
end for
for *all* SE_{ijk} **do**
  Identify SE_{ijk} with highest foreign communication
  PCMC $\leftarrow$ PCMC $\cup \{SE_{ijk}\}$
end for
Send analyzed data to RTMRPS
RTMRPS migration decision:
for *every simulation entity in PPMC* **do**
  If migration of SE_{ijk} is possible,
  PMC $\leftarrow$ PMC $\cup \{SE_{ijk}\}$
end for
for *every simulation entity in PCMC* **do**
  If migration of SE_{ijk} is possible,
  CMC $\leftarrow$ CMC $\cup \{SE_{ijk}\}$
end for
if $card(PMC) \geq 1$ *AND* $card(CMC) \geq 1$ **then**
  Prioritize
end if
if $card(PMC) \geq 1$ *AND* $card(CMC) = 0$ **then**
  pick randomly a simulation entity from PMC
end if
if $card(PMC) = 0$ *AND* $card(CMC) \geq 1$ **then**
  pick randomly a simulation entity from CMC
end if
if $card(PMC) = 0$ *AND* $card(CMC) = 0$ **then**
  trigger global inter-cluster load balancing
end if
Initiate migration
END

The mechanism is implemented by defining two counters. If the RTMRPS migration decision is settled on a computation migration candidate, then its respective counter will be incremented while the other decremented and vice versa. The monitoring,

analysis and migration decision policies of the imbalance with the lower counter are skipped for the next load balancing phase. For instance, assuming that the communication policies has the lower counter, the monitoring agents will stop updating their communication log tables after being notified by the OLNAS. Furthermore, the OLNAS communication data analysis and the communication RTMRPS migration decision will be skipped. It is worth mentioning that during that phase the RTMRPS computation migration decision will use the last saved communication behavior of the simulation entities. A similar approach is applied when the computation policies are skipped. The advantages of our adaptive scheme will be evaluated in Section 4.

3.4. *Migration Process*

JXTA [23–26] provides a virtual network, which can be used for designing highly dynamic and self-aware heterogeneous virtual networking infrastructure. This type of virtual distributed network aims to support run-time evolution of participating computing entities through a publish/subscribe based peer discovery protocol, as well as pipe based message passing mechanisms.

In our simulation, simulation entities can join a running simulation and the simulation structure evolves dynamically through dynamic simulation entities registration and dynamic pipes creation. The migration is initiated by the RTMRPS. The RTMRPS signals the STMS to pause the simulation while migration is under progress. When the simulation is paused, all the simulation entities save their current state. A copy of the simulation entity is created with the same simulation state and it is set to dynamically lookup the STMS. When it discovers the STMS, it makes a request for registration. After the registration is granted by the STMS, the simulation entity creates pipes and can join the simulation. Moreover, the RTMRPS releases all the resources held by the original simulation entity. After the migration is done, the RTMRPS signals the STMS to resume the simulation.

4. Performance Evaluation of Proposed Algorithm

After assembling all the different components of the design, a clear improvement became apparent. We implemented all the sets of rules and procedures already mentioned, and the results were as promising as we predicted. From what was mentioned earlier, our goal is to have a dynamic load balancing to decrease our simulation execution time. However, an overhead has been noticed. The overhead that was noticed and is supported by all of our experiments can be easily justified as the price of a dynamic load balancing scheme. We can never have dynamic load balancing without incurring some overhead, since this overhead increases our overall simulation performance. Based on the results of our experiments, the execution time of a simulation with high computation and communication imbalances has been enhanced significantly. However, an overhead has been noticed in terms of the simulation execution time. The monitoring agents, data analysis, migration decision and the actual migration are the sources of this overhead. The experiments ran on a cluster of nodes

at the PARADISE Research Laboratory at the University of Ottawa. The nodes' product type is eserver xSeries 336. Each node has two Intel Xeon processors at 3.4 GHz each, with 2 GB DIMM DDR synchronous RAM. Various experiments were performed using 8 and 32 nodes; one node is used to run the local head, which controls the simulation and runs the dynamic load balancing scheme, while the others are used by the remaining simulation entities. Interaction between the simulation entities occurs through the inter-process communication channels available.

The initial static partitioning assigns the local head to one node and distributes the simulation entities evenly on the remaining nodes. As a future work a more intelligent initial static partitioning will be the responsibility of the model static partitioning and deployment service [1]. In order to provide trustworthy results, each plotted point in our graphs represents the average of 33 runs. Thus, the confidence intervals were calculated using the z-distribution at a confidence level of 95%. Our experiments ran with a centralized conservative time management [29]. We simulated a tank battle model. We had a number of tanks running in the battlefield. The tanks are divided into several teams. A team of tanks can fire other tanks in different teams. A tank will fire to its enemy tanks when they are moving into its shooting range. The simulation ends when the simulation loops are over or when there is only one team left. Our simulation entities has the responsibility of simulating a number of tanks moving around in the battle field. The range of tanks for each of the simulation entities was between 2 and 10. Our simulation entity will simulate the tanks within a same team, and a team can consist of several simulation entities. For every simulation loop, the simulation entity requested time advance from STMS, received interactions from other simulation entities, computed its simulated tanks locations and sent interactions to other simulation entities. The simulation ran for 200 simulation loops. The computation load of our simulation is generated by the computation of the simulated tanks' locations. We implanted different hills with different steepness in the battle space. The computation differs whether the tank's location is on a flat surface or a hill, and depending how steep the hill is. As for the communication load, it is generated by the simulation entity location updates and the tank's shooting. Two experimental setups for five different test case groups were devised, and we will refer to them as experimental setups 1 and 2. Both experimental setups use the experimental configurations and environment already presented. However, the experimental setup 1 has 35 simulation entities running on 8 nodes. The OLNAS waiting time interval Δ was initially set to 0.5 seconds. The experimental setup 2 has a range of simulation entities from 60 to 200 running on 32 nodes. The OLNAS waiting time interval Δ was initially set to 5 seconds.

To evaluate our approach, experiments were clustered in five test-case groups. The five test-case groups use the experimental setups outlined here; however some of them may have some modifications. The first group highlighted the benefits of dynamic load balancing on the simulation performance. The second group studied the effect of balancing both computation and communication for different simulation

	Scenario 1	Scenario 2	Scenario 3
Simulation Execution Time (Seconds)	2.819	2.828	2.933
Total Overhead (%)			3.9
Monitoring, Data Analysis, Migration Decision Overhead (%)			0.3
Overhead per Simulation Entity (%)			0.01
Actual Migration Overhead (%)			3.6

scenarios. In the third test-case group, we evaluated our load balancing algorithm overhead. In the fourth test case group, we studied our load balancing algorithm adaptation mechanism and in the fifth we compared our load balancing algorithm to the load balancing algorithm presented in [13]. The first three test case groups use the experimental setup 1 while the last two use the experimental setup 2.

4.1. *Dynamic Load Balancing Benefits*

This test case group shows how dynamic load balancing can benefit our simulation performance. The percentage of computation and communication loads are increased for different experimental runs. The computation percentage load increase is accomplished by an increase the hills' steepness while the communication percentage load increase is accomplished by an increase in the tanks' shooting area range. The results are shown in Figure3. For less than a 30% load increase our dynamic load balancing exhibited more overhead up to 9.3% over actual performance gain. However, as we increased the computation and communication generated load by 100% we achieved a 24.8% decrease in the simulation execution time.

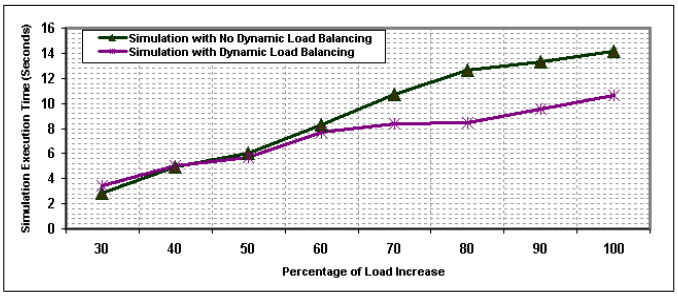

Fig. 3. **Comparison of a Simulation with and without Dynamic Load Balancing for an Increasing Percentage of Load**

4.2. *Effect of Computation and Communication Load Balancing*

In a second test-case group of experiments, we studied the effect of balancing both computation and communication for different simulations. Many existing algorithms takes only either computation or communication imbalances into consideration but not both. However, our algorithm functions best when both imbalances are present,

as is the case in most simulations. The simulation ran with and without dynamic load balancing for three different scenarios. Our load balancing adaptation mechanism was only limited to the waiting time interval. The percentage of computation and communication loads are increased using the same methodology presented in the first experiment. For the first scenario, the simulation ran with computation events only, in the second scenario with communication events only and in the third one with both computation and communication events. To generate only computation events we disabled the tanks' firing ability while to generate only communication events we set an all flat battle space. Figure 4, Figure 5 and Figure 6 illustrates the results of the three scenarios respectively. The dynamic load balancing overhead outweighs its gain for a small percentage of load since not much computation and communication load is generated. For instance, for a 30% load increase the overhead for the first scenario is 14%, the second scenario is 12.3% while the third is 9.3%. As we can observe, our algorithm produces more overhead for a simulation with only one kind of imbalance. However, as the percentage of load increases to 100 the gain for the first scenario becomes 17.9%, the second scenario 15.2% while the third becomes 24.8%. The higher gain in the computation, as compared to the communication, has no significance here, and is caused only by initial configurations. The preceding results verify our argument that our proposed dynamic load balancing performs best when both sources of imbalance are present which is the case in most simulations. Furthermore, our algorithm scales well for an increasing percentage of computation and communication load.

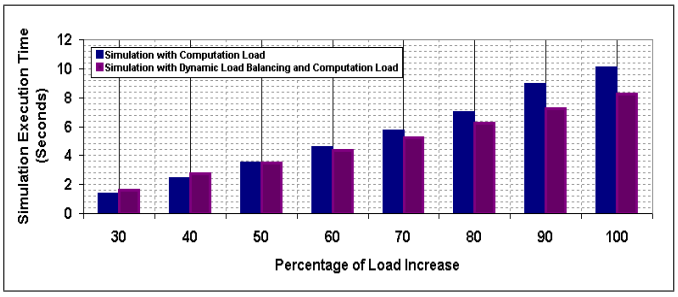

Fig. 4. **Simulation Comparison with and without Dynamic Load Balancing for Computation Load**

4.3. *Evaluation of the Dynamic Load Balancing Algorithm Overhead*

We also investigated the overhead sources of our load balancing algorithm: Monitoring agents, data analysis, migration decision and actual migration. We ran three different scenarios of our simulation in order to isolate the overhead generated from the monitoring policies, data analysis and migration decision from that generated by the migration. In the first scenario, we ran the simulation with no dynamic load bal-

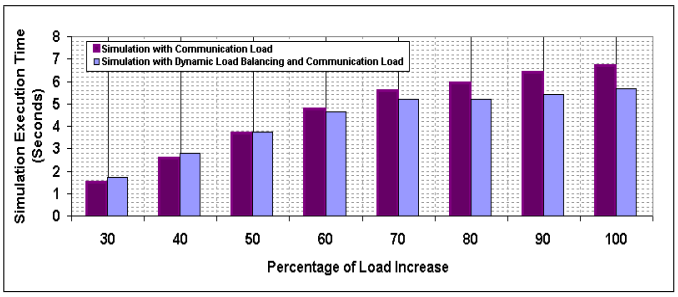

Fig. 5. **Simulation Comparison with and without Dynamic Load Balancing for Communication Load**

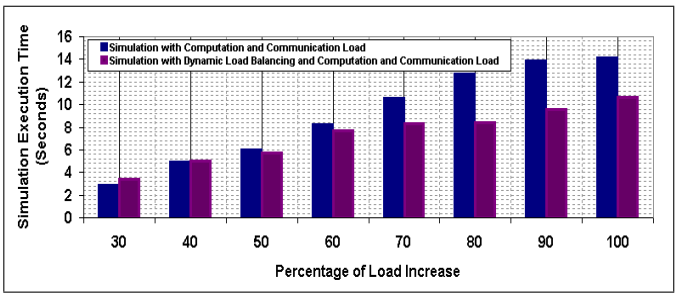

Fig. 6. **Simulation Comparison with and without Dynamic Load Balancing for Computation and Communication Load**

ancing. In the second, we added the dynamic load balancing algorithm but disabled the migration procedure, while in the third scenario; the migration was enabled and was limited to one. Table 1 summarizes our results; the overhead from the migration (3.6%) outweighs the overhead from the monitoring policies (0.3%). This validates our preceding argument of the one migration per session restriction.

4.4. *Evaluation of our Dynamic Load Balancing Algorithm Adaptation Mechanism*

In this fourth test-case group, we evaluate our algorithm adaptation mechanism specifically, our prioritization scheme aimed at reducing unnecessary overhead. An increasing number of simulation entities are used and are set to generate only one type of load imbalance which is a computation load in our case. We ran our simulation with the dynamic load balancing once with the adaptation mechanism functioning and once without it. As shown in figure 7, a decrease in the simulation run-time was noticed when using the adaptation mechanism. This is due to the fact that our algorithm considers both computation and communication loads as sources of imbalances. By integrating our adaptive prioritization scheme we were also able to get rid of the unnecessary overhead generated by the communication monitoring, analysis and migration decision. Our performance gain reached up to 9.8% for 200 simulation entities.

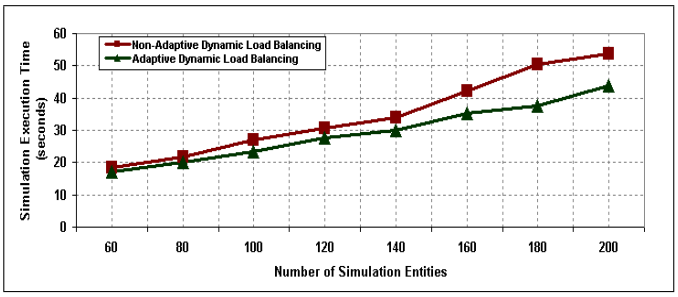

Fig. 7. **Comparison of a Simulation with and without Adaptive Dynamic Load Balancing**

4.5. *Comparison of Load Balancing Algorithms*

Finally, in this test-case group we compared our load balancing algorithm to the dynamic load balancing algorithm proposed in [13] and to a simulation running with no dynamic load balancing. As shown in figure 8, our algorithm had a significant improvement over the algorithm presented in [13] and over the simulation with no dynamic load balancing. The improvement noticed for 60 simulation entities was around 4.5% in comparison to [13] and around 16.6% in comparison to the simulation with no dynamic load balancing. Our algorithm scaled well and the improvement reached up to 21.9% in comparison to [13] and around 31.4% in comparison to the simulation with no dynamic load balancing for 200 simulation entities. The preceding results demonstrates our algorithm scalability with an increasing number of simulation entities. Moreover, most of our improvement is due to our low migration latency and our restriction to one migration per load balancing phase as opposed to many migrations and movements per a load balancing phase suggested in [13].

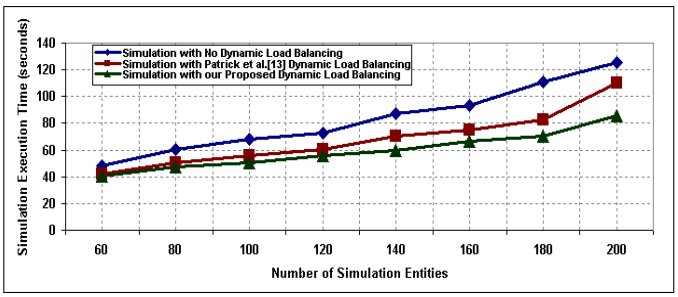

Fig. 8. **Comparison of our Proposed Load Balancing Algorithm to the Load Balancing Algorithm in**

The data and observations from our simulation experiments indicate that the benefits from the dynamic load balancing are significant. These experiments justified what has been claimed in the previous sections about the trustworthiness of our proposed dynamic load balancing scheme.

5. Algorithm Inter-Cluster Extension

The global inter-cluster load balancing scheme is an extension of the local intra-cluster load balancing scheme. However, few modifications need to be introduced in order to handle new challenges such as heterogeneity and scalability. The inter-cluster load balancing algorithm can be triggered in two different cases, either periodically or upon the failure of the intra-cluster balancing algorithm. The waiting time interval (Π) of the global inter-cluster algorithm will be greater than the largest participating cluster waiting time interval (Δ). This will ensure that enough effort has been made to balance the clusters locally before any global costly intervention. Moreover, in the intra-cluster load balancing scheme, the use of the average and standard deviation to identify the overloaded, balanced and under-loaded nodes is well justified since the nodes within one cluster are homogeneous. However, in a grid environment, most probably we have heterogeneous clusters. Therefore our algorithm needs to be slightly modified to meet this requirement. A benchmark test is necessary to identify the capacity of every cluster in terms of its nodes workload defined in (2).

$$Cap_{C_i} = \sum_{N \in C_i} Load_N \tag{5.1}$$

The benchmark test can be executed on one node and the capacity can be deduced by multiplying the node's capacity to the number of nodes in the cluster. The capacity of the node will be calculated by executing a sequence of interactions and computations while monitoring the node load using well known commands for different platforms (such as top command for Linux).

The cluster workload is calculated by the GOLNAS by summing the simulation entity group workloads provided by their global monitoring agents. The clusters classification is done by first normalizing the current cluster load by dividing it by its respective capacity. Consequently, the average, standard deviation and classification will be conducted on the normalized loads. Our global load balancing scheme will incur a large overhead compared to the local one. An inter-cluster simulation entity group migration can lead to significant network latency as opposed to a single intra-cluster simulation entity migration. Also, a benchmark test execution is needed to identify each cluster's capacity. Consequently, we configure our algorithm to provide the user the flexibility of switching off the global load balancing scheme. The basic concepts of the global inter-cluster algorithm are outlined in Algorithm 2. In the following section, we introduce Discrete Event System Specification (DEVS) and explain its conceptual framework, formalization and validation approach for simulation systems specification. Then we formalize and validate our inter-cluster dynamic load balancing and realize it in DEVSJAVA.

Algorithm 2 Global Inter-Cluster Load Balancing Algorithm

```
Π: GOLNAS Triggering interval
GSE_ij: i^th simulation entity group of j^th cluster
GMA_ij: i^th global monitoring agent of j^th cluster
C_i: i^th cluster
Nor(C_i): Normalized load of i^th cluster
μ: average of normalized cluster loads
σ: standard deviation of normalized cluster loads
OC: set of overloaded clusters
BC: set of balanced clusters
UC: set of under-loaded clusters
GPPMC: set of possible computation migration candidates of simulation entity groups
GPCMC: set of possible communication migration candidates of simulation entity groups
GPMC: set of computation migration candidates of simulation entity groups
GCMC: set of communication migration candidates of simulation entity groups
BEGIN
OC ← ∅, BC ← ∅, UC ← ∅, GPPMC ← ∅, GPCMC ← ∅, GPMC ← ∅, GCMC ← ∅
wait (Π) or upon intra-cluster load balancing failure
for all GMA_ij do
   Send monitored data to GOLNAS
end for
GOLNAS data analysis:
Calculate current load of all clusters, Normalize the loads by dividing
by the cluster capacity, Calculate μ and σ
for all C_i do
   if Nor(C_i) > μ + σ then
      OC ← OC ∪ {C_i}
   else if Nor(C_i) < μ − σ then
      UC ← UC ∪ {C_i}
   else
      BC ← BC ∪ {C_i}
   end if
end for
for every cluster in OC do
   Identify GSE_ij with highest workload
   GPPMC ← GPPMC ∪ {GSE_ij}
end for
for all GSE_ij do
   Identify GSE_ij with high foreign communication
   GPCMC ← GPCMC ∪ {GSE_ij}
end for
GRTMRPS migration decision:
for every simulation entity group in GPPMC do
   If migration of GSE_ij is possible,
   GPMC ← GPMC ∪ {GSE_ij}
end for
for every simulation entity group in GPCMC do
   If migration of GSE_ij is possible,
   GCMC ← GCMC ∪ {GSE_ij}
end for
if card(GPMC) ≥ 1 AND card(GCMC) ≥ 1 then
   Prioritize
end if
if card(GPMC) ≥ 1 AND card(GCMC) = 0 then
   pick randomly a simulation entity group from GPMC
end if
if card(GPMC) = 0 AND card(GCMC) ≥ 1 then
   pick randomly a simulation entity group from GCMC
end if
if card(GPMC) = 0 AND card(GCMC) = 0 then
   return
end if
Initiate migration
END
```

6. Global Inter-Cluster Algorithm Formalization and Verification using DEVS

6.1. *DEVS Formalism*

DEVS formalism [27] is one of the most important components of the modeling and simulation theory. It provides a conceptual framework and an associated computa-

tional approach for solving methodological problems in Modeling and Simulation (M&S) communities. This computational approach is based on the mathematical theory of systems including the hierarchy of system specifications and specification morphisms. It manipulates a framework of elements to determine their logical relationships. As a formal system specification language, DEVS formalism and its associated modeling and simulation environment is a very useful tool for helping with complex system design and verification. DEVS is one of the best tools for low-level system specifications due to its ability to represent real world components in a very detailed manner through its discrete event favor, which is generally required for designing large-scale distributed systems. It is worth mentioning that the operation of the discrete event modeled systems is basically represented by a chronological set of events that change the system's state. Thus, for any discrete-event system, the event that occurs at a specific point in time changes the state of the system. While in continuous event systems, time is a continuous function and events occur at any point in time with no specific expectations for events timing, and here appears the difference clearly in the time function.
The DEVS formalism is as follows(i.e. an atomic DEVS):

$$\textbf{\textit{Atomic Model (AM)}} = <\boldsymbol{X},\ \boldsymbol{S},\ \boldsymbol{Y},\ \delta_{int},\ \delta_{ext},\ \lambda,\ \boldsymbol{ta}>$$

Where:
X: set of external input events,
S: set of sequential states,
Y: set of outputs,
δ_{int}: S $\Longrightarrow$ S: internal transition function,
δ_{ext}: Q * X^b $\Longrightarrow$ S: external transition function
Where,
Q = {(s, e) | s $\in$ S, 0 $\leq$ e $\leq$ ta(s)}
X^b is a set of bags over elements in X,
λ: S $\Longrightarrow$ Y^b: output function,
ta: time advance function,

6.2. *Formalized System Design using DEVS*

Design of software or embedded systems is traditionally non-formalized, which often results in low performance and error-prone system architecture. It is worth mentioning that such non-formalized design approaches are still being used by modern researchers. Some researchers may argue that it is not necessary to use complex formal languages to describe their designs because they believe they have enough experience. However, questions such as: "How can you verify that you have reached an optimal system design? and Is your designed system scalable? If so, can we verify the efficiency of this design?" are definitely difficult to answer. From another point of view, it is actually impractical to use a non- formalized design for large-scale and

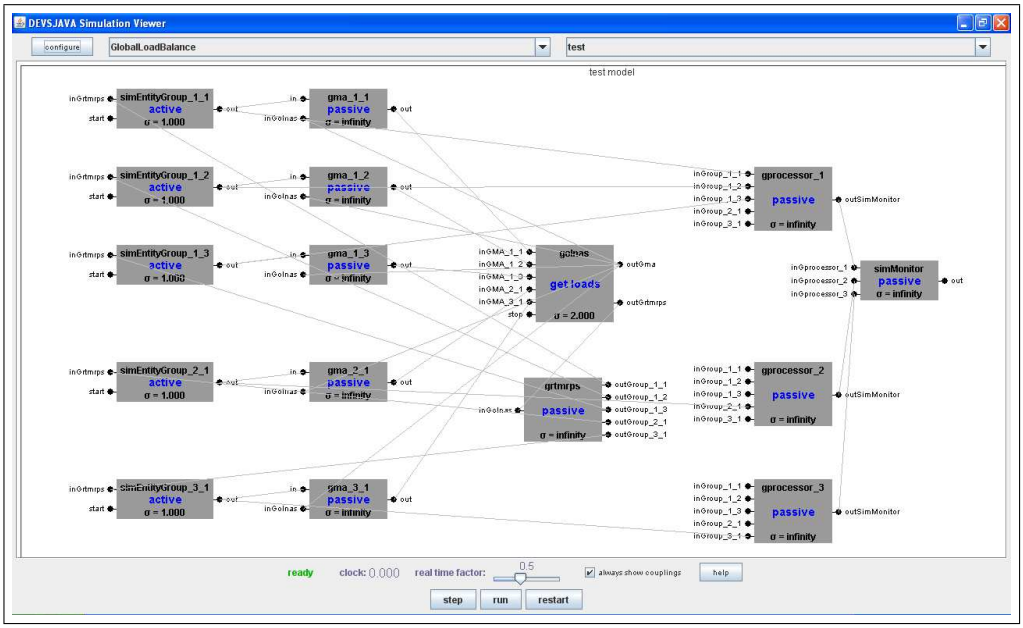

Fig. 9. **Proposed Design Realized in DEVSJAVA**

dynamic systems since these systems generally require stage-by-stage realizations, verifications and a final integration. Design formalization becomes a necessity for designing large-scale and complex dynamic systems, especially real-time distributed systems, where time constraints need to be considered carefully at a very early stage. One of the key advantages of formalized design approach, such as using DEVS, is that critical system design problems can be identified at earlier stages before the costly overhead of realizing the implementation. Such an approach also supports design reusability by using a model repository. Another advantage is that the design validation can be done before its realization. DEVS model-based design has been used by many researchers for solving complex design problems. For instance, Schulz and Rozenblit [30] have proposed a novel co-design approach that uses a formal specification language to describe embedded systems. In addition, Hu [31] has proposed a DEVS model-based methodology to be applied in designing dynamic distributed real-time systems with a particular focus on model continuity.

6.3. *Proposed Design Formalization using DEVS*

In this section, we formalize our design by using DEVS, and then realize the design in the DEVSJAVA environment. The design model components presented here are all reusable ones that can also be used to construct alternative design systems.

As shown in Figure 9, the design model is composed of the proposed dynamic load balancing design. The model is composed of *simEntityGroup* (simulation entity group), *gma* (global monitoring agent), *golnas* (global online network analyzing

service), *grtmrps* (global run time model repartitioning service), *gprocessor* (global processor), and *simMonitor* (simulation monitor). Each of these model components is described using DEVS formalism and then realized as DEVSJAVA atomic models. In Figure 9, the illustrated model is composed of 3 clusters. However, our implementation can handle any number of clusters with any number of simulation entity groups. *simEntityGroup_1_1*, *simEntityGroup_1_2* and *simEntityGroup_1_3* form cluster 1 while *simEntityGroup_2_1* forms cluster 2 and *simEntityGroup_3_1* forms cluster 3. Each *simEntityGroup* has its respective *gma* and each cluster has its respective *gprocessor*. The *golnas* and *grtmrps* are responsible for the dynamic load balancing while the *simMonitor* monitors the simulation and signals its end.

In the following paragraphs, we present each design model component in detail, and several key model components are presented using DEVS formalism as representatives.

1. *simEntityGroup* model: The *simEntityGroup* generates events within every simulation loop. It outputs to its respective *gprocessor* and *gma*. The generated events can be either computation or communication loads.

2. *gma* model: The *gma* model calculates and stores the load of the *simEntityGroup* generated events. It stores the load of the generated events and, upon the *golnas* periodic request, sends the saved load to the *golnas*.

3. *gprocessor* model: The *gprocessor* model is responsible for processing the events generated by all the *simEntityGroup* belonging to the same cluster. The computation events processing by the *gprocessor* represents the computation processing done by the different cluster processors while the communication events processing represents the latency generated by the message passing over the network. After processing all the events it signals to the *simMonitor* that it has finished. The gprocessor has a predefined capacity for processing the *simEntityGroup* generated events which represents the cluster's capacity.

4. *golnas* model: The *golnas* model is the key control unit for triggering dynamic load balancing. The *golnas* sends out a request to each *gma* upon the elapse of its waiting time interval (Π) which is calculated as described in Algorithm 2. After getting the load generated by each *simEntityGroup*, it then classifies the clusters as overloaded, under-loaded and balanced and identifies within the overloaded cluster the highest computation load generating simulation entity group. Moreover, it analyzes the simulation entity group communication behavior whether it has high or low host binding. Finally, it sends to the *grtmrps* the simulation entity groups which are candidates for migration.

5. *grtmrps* model: The *grtmrps* model decides whether a simulation entity migration is possible. If a migration is possible it sends a message to the selected *simEntityGroup* in order to migrate to the specified cluster. This model has a con-

figurable delay for outputting its result to the selected *simEntityGroup* model. The delay in here represents the actual migration latency which can be approximated by having the clusters bandwidth connections and the simulation entity groups size.

6. *simMonitor* model: The *simMonitor* model monitors the simulation and upon receiving messages from all the *gprocessor* of the clusters participating in the simulation signals the end of the simulation.

As we can see from the above descriptions of the model components, they have some common characteristics. We therefore select only two key models to describe in DEVS formalism as representatives.

The *gma* model is described with DEVS as

X={computation load, communication load, fetch load}
S={passive, active, outLoad}
Y={monitored load}
δ_{int}{passive, active, outLoad} = {passive}
δ_{ext}{computation load,{passive,0<e<infinite}} = {active}
δ_{ext}{communication load,{passive,0<e<infinite}} = {active}
δ_{ext}{fetch load,{passive,0<e<infinite}} = {outLoad}
λ{outLoad} = {monitored load}
ta(outLoad) = 0 ; ta(active) = 0 ; ta(passive) = infinite

The *golnas* model is described with DEVS as

X={monitored load}
S={passive, fetchLoad, analyzeData}
Y={fetch load, migration candidates}
δ_{int}{passive, fetchLoad, analyzeData} = {passive}
δ_{ext}{monitored load,{passive,0<e<infinite}}={analyzeData}
λ{fetchLoad} = {fetch load}
λ{analyzeData} = {migration candidates}
ta(fetchLoad) = 0 ; ta(analyzeData) = 0 ; ta(passive) = infinite

In the following section, we will present some precisely designed simulation experiments that aim to test and verify the advantages of the proposed dynamic load balancing design when compared with a design with no load balancing. Our particular focus is on looking at two key characteristics of our design; The importance of balancing both computation and communication loads, and the effect of runtime model repartitioning through dynamic migration on the performance of the simulation execution time.

Table 2. Comparison of acoustic for frequencies for piston-cylinder problem.

Simulation Loops	200
Waiting Time Interval (Δ) for Cluster 1	3
Waiting Time Interval (Δ) for Cluster 2	2
Waiting Time Interval (Δ) for Cluster 3	4
Processing Capacity of Cluster 1	0.3
Processing Capacity of Cluster 2	0.6
Processing Capacity of Cluster 3	0.2
Initial Generated Computation Events Processing Time Random Range	0.1 - 0.3
Initial Generated Local Communication Events Message Latency Random Range	0.1 - 0.2
Initial Generated Foreign Communication Events Message Latency Random Range	0.2 - 0.3
Estimated Migration Latency	5

6.4. *Design Verification through Simulation Experimental Frames*

In this section, we follow the experimental frame concepts commonly used in DEVS based modeling and simulation. We analyze our design using previously defined DEVS models, and we focus our experiments on the effects of dynamic migration and the simultaneous computation and communication load balancing. For the following simulation experiments, the design models are simulated in a DEVSJAVA environment. Our simulation experiments are designed to verify the key aspects of the proposed inter-cluster algorithm. Therefore, the following simulation experiments presented will focus on two of the most important characteristics of our new design: run time model repartitioning through dynamic migration and the effect of balancing both computation and communication loads.

It should be noted that, for the following simulation experiments, all the models are set to depict the same behavior as illustrated in Algorithm 2. Table 2 summarizes the parameter settings used for our experiments. Moreover, the linkages between the *simEntityGroup* and *gprocessor* can be altered dynamically according to the load balancing requirements, and such dynamic linkage changes are realized in DEVSJAVA by using a technique called variable structure. This technique has the ability to change the model system structure and couplings during simulation run-time.

6.4.1. *Run-Time Model Repartitioning in the Modeled Design*

In this simulation experiment, we study how dynamic migration improves the overall simulation performance. The simulation is set to run with an increasing percentage of load by increasing the computation and communication events processing time random range generated by the *simEntityGroup*. The dynamic migration is realized in DEVS models by a request from the *grtmrps* model to a *simEntityGroup* model. Consequently the coupling between the *simEntityGroup* model and the *gprocessor* model are changed accordingly.

In this experiment, we ran two different scenarios; In the first one, the simulation is set to run without dynamic load balancing while in the second with dynamic load

balancing. As shown in Figure 10, the simulation running with dynamic load balancing and run-time model repartitioning through dynamic migration had a significant improvement on the simulation performance. The improvement on the simulation execution time varied between 14.5% with a 20% load increase up to 33.1% for a 100% load increase. This result verifies and validates the effectiveness and worthiness of our proposed inter-cluster algorithm. However, it is worth mentioning that the preceding results are based on our initial configurations and estimated migration latency. Therefore, on a real implementation the migration overhead should not count more than the predicted gain.

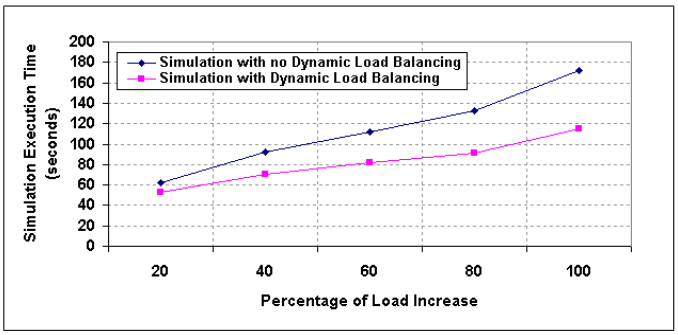

Fig. 10. **Simulation Comparison with and without Dynamic Load Balancing for an Increasing Percentage of Load using DEVS**

6.4.2. *Effect of computation and communication load balancing in the Modeled system*

In this simulation experiment, we use the same experimental setup and parameters used in the earlier experiment. However, we focus on studying another key characteristic of our algorithm which is the effect of computation and communication load balancing. A lot of existing algorithms takes only either computation or communication imbalances into consideration but not both. However, our algorithm functions best when both imbalances are present which is the case in most simulations. We simulated three different scenarios; A scenario where the *simEntityGroup* model generates only computation events another where it generates only communication events and a last one where we have both computation and communication events. As shown in Figure 11, the gain in performance is at most when the simulation encounters both communication and computation load imbalances. The average gain in terms of execution time where only computation events were generated was around 18.6%, where only communication events were generated was around 14.1% and where both loads were generated was around 26%. The higher gain in the computation part as compared to the communication one has no significance here and is caused only by initial configurations.

This experimental result further verifies our proposed design in terms of the importance of considering both computation and communication load imbalances. Therefore, the computation and communication load-balancing capability is an important characteristic and advantage for our proposed design.

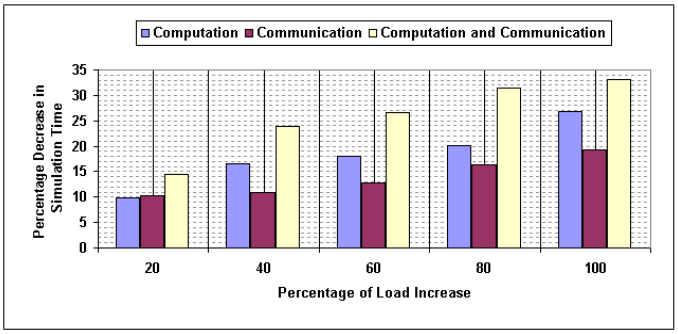

Fig. 11. **Percentage Decrease in Simulation Time with Different Load Imbalances and an Increasing Percentage of Load using DEVS**

In this section, we have demonstrated the powerfulness of the simulation based modeling design approach for the proposed dynamic load balancing algorithm. The DEVS model simulated in DEVSJAVA provides the key design parameters for predicting the crucial performance characteristic of the real system. It is also necessary to mention that the simulation experiments used here only examine some of the key interests for our design, and further more experiments can be done by changing the experimental frames. The purpose of the above simulation experiments is to verify and validate our design through formalized simulation based modeling using DEVS.

7. Conclusion and Future Work

In this paper, we proposed an adaptive and efficient dynamic load balancing scheme. It is generally applicable to different simulation requirements and adapts to meet the simulation dynamic behavior. The JXTA service based modular implementation favors our hierarchical scalable design and any future service or architecture extendibility. We verified our proposed dynamic load-balancing scheme by a series of simulation experiments, and our scheme demonstrated a better performance in terms of the simulation execution time. Furthermore, we extended our proposed algorithm to an inter-cluster algorithm and presented a DEVS based formal language approach for our design.

As a future work we plan to implement our global load balancing scheme using a set of clusters at the PARADISE lab.

References

1. A. Boukerche and M. Zhang. Towards Peer-to-Peer Based Distributed Simulations on a Grid Infrastructure. In Proceedings of the 41st Annual Simulation Symposium (anss-41 2008), pages 212-219, April 2008.
2. IEEE Standard. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules. 1516 2000, September 2000.
3. IEEE Standard. Draft Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Federation Interface Specification. 1516.1-2000, April 2000.
4. IEEE Standard. Draft Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template (OMT) Specification. 1516.2-2000, April 2000.
5. C. D. Carothers, Richard Fujimoto. Background Execution of Time Warp Programs. Workshop on Parallel and Distributed Simulation ,pages 12-19, 1996
6. M. Jiang, R. Anane, G. Theodoropoulos. Load balancing in distributed simulations on the grid. 2004 IEEE International Conference on Systems, Man and Cybernetics, Volume 4, pages 3232 - 3238, 10-13 Oct. 2004
7. J. Cao, D. P. Spooner, S. A. Jarvis, G. R. Nudd. Grid load balancing using intelligent agents. Future Generation Comp. Syst. 21 (1), pages 135-149, 2005
8. J. Wang, Q. Yuan Wu, D. Zheng, Y. Jia. Agent based Load Balancing Model for Service based Grid Applications. International Conference on Computational Intelligence and Security, 2006 Volume 1, pages 486 - 491, 2006
9. W. Cai, S. J. Turner, and H. Zhao. A load management system for running hla-based distributed simulations over the grid. In DS-RT '02: Proceedings of the Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications, page 7, Washington, DC, USA, 2002.
10. K. Zajac, M. Bubak, M. Malawski. Towards a Grid Management System for HLA-based Interactive Simulations. In Proceedings of The Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications, Delft, The Netherlands, pages 4-11, 2003.
11. Z. Yuan, W. Cai, and M. Yoke H. Low. A framework for executing parallel simulation using rti. In Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications, page 12, Washington, DC, USA, 2003.
12. L. F. Wilson and W. Shen. Experiments in load migration and dynamic load balancing in speedes. In Proc. of the 1998 Winter Simulation Conference, pages 483-490, 1998.
13. P. Patrick, Tobias H., P. Martini. A Flexible Dynamic Partitioning Algorithm for Optimistic Distributed Simulation. In Proceedings of the 21st Workshop on Parallel and Distributed Simulation (PADS'07), pages 219-228, 2007.
14. M. Y. H. Low. Dynamic load-balancing for BSP time warp. In Proceedings of the 35th Annual Simulation Symposium (SS'02), pages 267-274, 2002.
15. A. Boukerche and S. K. Das. Dynamic load balancing strategies for conservative parallel simulations. In Proceedings of the 11th Workshop on Parallel and Distributed Simulation (PADS'97), pages 32-37, 1997.
16. G.Tan, and K.C.Lim. Load Distribution Services in HLA. In proceedings of 8th IEEE Distributed Simulation and Real-time Applications, pages 133-141, Budapest, Hungary, October 2004.
17. G. Theodoropoulos and B. Logan. An approach to interest management and dynamic

load balancing in distributed simulation. In Proceedings of the 2001 European Simulation Interoperability Workshop, pages 565-571, July 2001.
18. Morse, K. L. (1996). Interest management in large scale distributed simulations. Technical Report 96-27, Department of Information and Computer Science, University of California, Irvine.
19. R. Schlagenhaft, M. Ruhwandl, C. Sporrer, and H. Bauer. Dynamic load balancing of a multi-cluster simulation on a network of workstations. In Proc. of 9th Workshop on Parallel and Distributed Simulation. Society for Computer Simulation, pages 175–180, 1995.
20. C. Burdorf and J. Marti. Load Balancing Strategies for Time Warp on Multi-User Workstations. The Computer Journal, 36(2), pages 168-176, 1993.
21. B. P. Gan, Y. H. Low, S. Jain, S. J. Turner, W. Cai, W. J. Hsu, S. Y. Huang. Load balancing for conservative simulation on shared memory multiprocessor systems. In proceedings of the 14th Workshop on Parallel and Distributed Simulation, pages 139-146, 28-31 May 2000.
22. K. El-Khatib and C. Tropper. On metrics for the dynamic load balancing of optimistic simulations. In proceedings of the 32nd Hawaii International Conference on System Sciences. pages 13-26, 1999.
23. JXSE 2.5 Programmers Guide:JXTA Concepts, https://jxta.dev.java.net/.
24. JXTA Protocols Specification, http://jxtaspec.dev.java.net.
25. E. Halepovic, R. Deters. The costs of using JXTA. Proceedings of the Third International Conference on Peer-to- Peer Computing, pages 160-167, 2003.
26. G. Antoniu, M. Jan, D. A. Noblet. Enabling the P2P JXTA Platform for High-Performance Networking Grid Infrastructures. In proceedings of the first Intl. Conf. on High Performance Computing and Communications (HPCC'05), Sorrento, Italy, Lect. Notes in Comp. Science, Springer-Verlag, September 2005, no3726, p. 429440.
27. B. P. Zeigler, T.G. Kim, and H. Praehofer. Theory of Modeling and Simulation. 2 ed. New York, NY: Academic Press, 2000
28. B. Zeigler H. Sarjoughian. Introduction to DEVS Modeling & Simulation with JAVATM: Developing Component-based Simulation Models. Arizona Center for Inte- grative Modeling and Simulation, Arizona State University, March, 2003
29. Azzedine Boukerche, Ming Zhang. Time Management Service in JXTA Based Distributed Simulation. In proceedings of 12^{th} IEEE International Symposium on Distributed Simulation and Real Time Application, pages 167-173, 2008.
30. Schulz, S., J.W. Rozenblit, M. Mrva and K. Buchenrieder. Model-Based Codesign. IEEE Computer, 31(8), pages 60–67, 1998.
31. Hu, X. and B.P. Zeigler. Model Continuity in the Design of Dynamic Distributed Real-Time Systems. IEEE Transactions on Systems,Man And Cybernetics – Part A: Systems And Humans, 35(6), pages 867–878, 2005.
32. R. M. Fujimoto. Parallel simulation: parallel and distributed simulation systems. In Proceedings of the 2001 Winter Simulation Conference, pages 147–157 , 2001.
33. I. Foster, C. Kesselman and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In International J. Supercomputer Applications 15(3), 2001.
34. Donald Knuth. The Art of Computer Programming. 3rd edition, volume 1, 1997, p.98