Dynamic Balancing of Communication and Computation Load for HLA-Based Simulations on Large-Scale Distributed Systems

Robson Eduardo De Grande, Azzedine Boukerche

School of Information and Technology Engineering, University of Ottawa, Ottawa, CA

Abstract

Dynamic balancing of computation and communication load is vital for the execution stability and performance of distributed, parallel simulations deployed on the shared, unreliable resources of large-scale environments. High Level Architecture (HLA) based simulations can experience a decrease in performance due to imbalances that are produced initially and/or during run time. These imbalances are generated by the dynamic load changes of distributed simulations or by unknown, non-managed background processes resulting from the non-dedication of shared resources. Due to the dynamic execution characteristics of elements that compose distributed applications, the computational load and interaction dependencies of each simulation entity change during run time. These dynamic changes lead to an irregular load and communication distribution, which increases overhead of resources and latencies. A static partitioning of load is limited to deterministic applications and is incapable of predicting the dynamic changes caused by distributed applications or by external background processes. Therefore, a scheme for balancing the communication and computational load during the execution of distributed simulations is devised in a scalable hierarchical architecture. The proposed balancing system employs local and cluster monitoring mechanisms in order to observe the distributed load changes and identify imbalances, repartitioning policies to determine a distribution of load and minimize imbalances. A migration technique is also employed by this proposed balancing system to perform reliable and low-latency load transfers. Such a system successfully improves the use of shared resources and increases distributed simulations' performance by minimizing communication latencies and partitioning the load evenly. Experiments and comparative analyses were conducted in order to identify the gains that the proposed balancing scheme provides to large-scale distributed simulations.

Keywords: Parallel Simulations, High Level Architecture, Dynamic Load Balancing, Performance.

1. Introduction

Generally, large-scale High Level Architecture (HLA) based distributed simulations are subject to load imbalances caused by the uneven partitioning of such simulations' load and by the dynamic load changes that might occur during run time. Uneven partitioning stems from a lack of knowledge about distributed applications or shared available resources. This irregular partitioning leads to some resources to be overloaded while others are underloaded. Moreover, distributed applications might present unpredictable run-time load changes. In such cases, a static initial load partitioning barely reaches efficient resource usage because it cannot optimally distribute load based on a deterministic assessment of the simulations' characteristics. Other than the existence of uneven load distribution, factors inherent to large-scale distributed systems must be considered, such as the heterogeneity of resources and the presence of external background load. Therefore, in order to provide a more efficient use of shared resources and to increase the performance of large-scale distributed HLA-based simulations, a dynamic balancing of computation and communication load is devised.

A distributed simulation consists of a set of interactive, independent simulation entities that perform simulation processing over a group of shared resources. The partitioning of such entities may lead to causality simulation inconsistencies, thus corrupting the simulation's result. To organize the interactions between simulation entities, a High Level Architecture (HLA) framework is employed. The HLA standard includes rules and services, which maintain the communication order of simulations. However, the HLA standard is limited to the management of only distributed simulations, and ignores the issues caused by distributing simulations on unreliable, non-dedicated, shared resources; one example being load imbalances and failures. Thus, resource management mechanisms and load balancing schemes are needed in order to coordinate the utilization of available resources and improve the simulation performance.

Resource management systems are essential for administering distributed resources; the Grid system is extensively employed to organize distributed applications that run on shared resources through the provisioning of various services. According to Foster [1], Grid computing involves the organization of resources, individuals, and institutions, establishing a coordinated resource-sharing system aimed at flexibility and security. Furthermore, based on the Open Grid Services Architecture [2], Globus Toolkit [3] is the *de facto* middleware standard for Grid

Email addresses: rdgrande@site.uottawa.ca (Robson Eduardo De Grande), boukerch@site.uottawa.ca (Azzedine Boukerche)

computing. This standard combines stateful web services in the Grid architecture. Grid services encompass the monitoring of resources and distributed applications, the scheduling and allocation of resources, and the identification of application requirements [4]. However, these services do not resolve load imbalance issues although they do provide basic instruments for balancing systems.

In order to maximize resource utilization and consequently increase simulation performance, several dynamic balancing approaches were proposed in the literature. These approaches generally attempt to react effectively to the load changes of distributed simulations. Such solutions are usually composed of monitoring mechanisms to detect imbalances, load repartitioning policies to reconfigure the load distribution, and load migration methods to move load according to necessary redeployment. Moreover, these approaches were designed with the characteristics of distributed systems and applications, such as heterogeneity and non-dedication, in mind. Each approach in its scheme considers different aspects, related to the system and environment being managed. However, such proposed solutions do not provide a balancing scheme that redistributes the communication and computational load according to the needs of large-scale HLA-based simulations.

In order to provide a dynamic load redistribution scheme that can improve HLA-based simulations distributed in large-scale environments, this paper presents a new balancing approach. This scheme considers communication and computational loads in its design and detects heterogeneity and external background load, which are inherent aspects of large-scale shared environments. The proposed balancing scheme consists of a hierarchical design that essentially monitors resources and applications, load reallocation, and load migration. The hierarchical architecture of the scheme minimizes balancing overhead while load redistribution is realized. Balancing involves normalizing load distribution on shared resources and maximizing overall resource utilization. Grid services are employed in the balancing approach to help support the system's proper scaling according to the magnitude of the balanced system. These services also provide tools for monitoring the resources and performing reliable transfers of data for load migration.

The remainder of this paper is organized as follows. Section 2 details the HLA standard and its limitations. Section 3 describes related work on the existing solutions for redistributing load dynamically, as well as their drawbacks. In Section 4, the proposed hierarchical three-phase scheme for balancing communication and computational load and its architecture are introduced. In Section 5, three test case groups of experiments are outlined, and the experimental results are shown and discussed. Finally, Section 6 provides the conclusion and proposes directions for future work.

2. High Level Architecture

High Level Architecture is a standard initially developed by the United States' Department of Defense in order to perform distributed simulations for military purposes. In 2000, the HLA specification became an open IEEE standard [5] for parallel simulations. The specification defines management mechanisms and design rules, where simulation causality inconsistencies are avoided and interoperability and reusability aspects are provided. Interoperability refers to the capacity of many elements to operate together, while reusability stems from the separation of simulations into independent, stand-alone components. Thus, these specification aspects determine how the simulation entities, called federates, must be designed as well as how they interact with one another in simulations, called federations.

According to the HLA specification [5], the framework consists of three main components: the Federation Rules, which delimits the actions of simulations; the Object Model Template, which defines the data transferred in the simulations; and the HLA Interface Specification, which describes Run Time Infrastructure (RTI) services and their interface. Moreover, as shown in Figure 1, RTI software is composed of an RTI Executive process (RTIExec), a Federation Executive process (FedExec), and the libRTI library [5]. The RTIExec manages Federations and FedExecs, while a FedExec organizes the simulation of one federation by managing the life cycle of all federates within this simulation. The libRTI includes the mechanisms used to organize distributed simulations and provides them as management services. These management services are accessed by federates in order to coordinate operations and data exchange. Thus, all communication between federates is ruled by the RTI, enabling federates to access it through the Local RTI (LRC) containing all libRTI library interfaces.

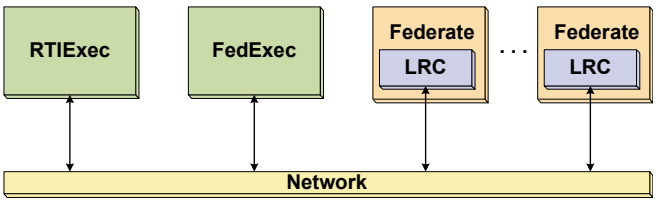

Figure 1: HLA General Architecture.

However, the HLA standard does not define any mechanism or method for manipulating the distributed load deployed on shared resources or managing the underlying shared resources from the distributed system where simulations run. Due to this limited management, HLA simulations rely on the fact that all resources are reliable and dedicated. HLA simulations can be entirely compromised due to load imbalances stemming from only one overloaded resource. Thus, load balancing mechanisms are necessary to HLA simulations so that they can improve execution performance by maximizing the distributed resources' usage.

3. Related Work

Many dynamic load balancing approaches have been proposed, aiming to solve uneven load partition problems for simulations distributed on shared resources and to therefore achieve

certain improvements in performance. In the case of distributed simulations, the existing approaches assess the state of simulation applications or the characteristics of the distributed system where such applications run. These approaches attempt to identify load imbalance, perform load transfers, and decrease simulation execution time. Many aspects are considered in the design of these balancing approaches, such as monitoring metrics, resource heterogeneity, external background load, simulation computing load, simulation entities' interactions, and other simulation-specific characteristics, such as lookahead. However, none of the solutions in the literature address the aspects that best suit HLA simulations running on large-scale distributed systems.

The previous dynamic load balancing schemes proposed for distributed simulations employ different techniques for monitoring, redistribution, and migration. For monitoring, different metrics are used to determine load imbalance. Some approaches that measure the simulation speed of each simulation entity, others collect CPU consumption of a simulation application, and other approaches observe simulation-specific characteristics. The monitoring of simulation pace is used to identify which simulation entities are running slower than others. This metric includes the following factors: simulation advance time [6]; the smallest simulation virtual time in each processor [7]; the virtual time progress of a processor [8]; processor advance time [9]; and simulation advancement [10]. This metric provides an indirect detection of background load, is application-specific – dependent on simulation characteristics – and relies highly upon the simulation code to retrieve the correct system load status. Instead of basing load balancing on a recent computing past, some approaches measure the future simulation workload presented by the simulation processing queue, such as the weighted unprocessed events [11] or the pending load [12] of simulation entities. This technique avoids the use of estimates to predict future loads, relying instead on the accurate assignment of weights to unprocessed events. Other balancing schemes employ CPU or memory consumption as their monitoring metric; these schemes look at simulation CPU consumption [13] [14] or memory consumption [15] of simulation entities. These metrics allow balancing schemes to more precisely identify the load produced by simulation elements more precisely, but they do not detect background load and require weights to overcome heterogeneity issues. Some approaches observe simulations' intradependencies in order to identify imbalances; such dependencies are determined by lookahead [16] and communication dependencies [17], [18], [14], [19]. In this case, the gathering of lookahead relies heavily on simulation characteristics while the communication rate provides a more application-independent solution.

For redistribution, some approaches simply focus on balancing the computational load in their schemes while other approaches consider communication dependencies when repartitioning the load to better reorganize simulation distribution. The scheme proposed by Boukerche [20], [21] joins computation and communication metrics in a formula that is used to determine load imbalances based on a simulated annealing algorithm, which determines execution entropy. Because the relation between computation and communication is highly dependent on a simulation's needs and implementation, this approach cannot be employed in a general balancing system. Some redistribution schemes focus on simulation intradependencies; they attempt to reorganize the simulation entities in order to minimize communication delays [15], [18] and lookahead differences [22], [16]. In this approach, the redistribution of simulation elements helps to decrease simulation delays by minimizing communication overhead and waiting times, but the analysis of lookaheads is highly application-dependent. The analysis observes the characteristics of a simulation in order to repartition its distributed entities. Other balancing schemes perform comparisons between each simulation entity load and the overall system's average [16], [7], [12]. Simulation speed and CPU consumption are usually employed in such balancing methods. With this technique, the goal is to normalize the use of resources by keeping the load properly distributed on the shared resources. Additionally, some balancing schemes use communication dependencies to perform computational load redistribution; this approach considers the communication dependencies between the simulation elements when balancing them [17]. In other schemes, both computation and communication are equally considered as means of redistributing load, such as the methods proposed by Peschlow et. al. [10] and Ajaltouni et. al. [14]. These balancing techniques perform the redistribution of load in alternate computation and communication balancing cycles.

For migration, there exist several techniques designed to transfer load in order to perform the necessary load redistribution. Minimizing the delay generated by the migration procedure is the principal aim of such techniques. Migration delay dictates balancing reactivity by restricting the number of load transfers that a balancing system can perform. Also, minimizing such delay provides better performance gains and allows the balancing scheme to redistribute the load more frequently. Some migration techniques introduce global synchronization into simulations, propagating the migration delay to all simulation entities and causing a large overhead in the simulation. In this case, simulations that are balanced can be stopped regularly in order to perform load redistribution [12] and migrations. With such approaches, the entire simulation is suspended while a simulation entity is transferred [23], [24], [25], or an optimistic simulation is rolled back to the last GVT for migration [9]. Other migration techniques attempt to decrease the delay caused by migration, employing more complex mechanisms to make the migration process more transparent to simulations [26]. Consequently, transferring a simulation entity has a minimal effect on the rest of a simulation, yet it introduces additional processing overhead into the system.

The aforementioned dynamic load balancing schemes are focused on sets of specific parameters. They present different approaches to solving the load redistribution problem for distributed simulations. The approaches provide certain performance gains, but they do not fully consider communication balancing, properly detect heterogeneity of resources and external background load, nor minimize migration delay. Computation and communication load balancing is vital for achieving dis-

tributed simulation performance improvement, and only a few proposed redistribution systems consider both computation and communication load in their balancing schemes. Additionally, when communication balancing occurs, migration moves are performed in order to keep interactive simulation entities in the same resource, thus avoiding network overhead. Even though this load redistribution is successful in decreasing communication overhead, it does not deal with all cases of communication configuration, more specifically, with distributed simulations based on a centralized manager, such as HLA/RTI-based simulations. Moreover, some approaches consider the load produced only by the simulation that is balanced, and these approaches are therefore inadequate for non-dedicated, heterogeneous resources. Other approaches use the simulation entity's processing speed to monitor load imbalances, which indirectly and partially detects the existence of imbalances in these distributed resources. In order to fully overcome the problem of heterogeneity, benchmarks and normalization parameters need to be introduced into the balancing system. To detect background load, tools are necessary in order to measure the total load consumption of shared resources, and not just the load consumption from a determined simulation application. Load migration, which is a critical procedure for balancing simulation's load, is absent in several proposed solutions, and if migration is part of a solution, it introduces considerable latencies, decreasing the performance and balancing system reactivity. A new dynamic load balancing scheme is therefore proposed as means of overcoming such issues and improving the HLA simulations' execution performance in large-scale distributed environments.

4. Dynamic Load Balancing Scheme

A balancing scheme is proposed to provide a simulation-independent, dynamic load redistribution system for large-scale distributed HLA-based simulations. This scheme considers the heterogeneity of resources that compose the distributed system and detects the presence of external background load that runs on non-dedicated shared resources. In order to react to dynamic changes of communication and computation load, the proposed system constantly monitors resources and simulations, using a hierarchical architecture to decrease the monitoring overhead. Moreover, after imbalances are detected, the system defines a load redistribution according to partitioning policies and the reconfigures load through a low-latency federate migration technique.

4.1. Monitoring Phase

The monitoring phase is essential for detecting imbalances that occur unpredictably and dynamically during run time in a distributed system. Monitoring is the initial phase in a reactive or adaptive dynamic load balancing system since the results of environmental analyses indicate the need to reallocate resources and consequently trigger the other balancing phases. After environmental information is gathered, filtering techniques and selection mechanisms are used to retrieve analyze the data in order to identify load discrepancies.

4.1.1. Data Collection

The monitoring phase involves two-level hierarchical data gathering to minimize the amount of collected information that is transmitted and processed. This two-level design is employed because data collection is limited by the trade-off between precision and overhead. The decrease in the amount of data that is collected allows for an increase in the gathering frequency and the amount of relevant data. The gathered information consists of the communication load of federates and of the computational load of resources and federates. Thus, hierarchical monitoring consists of a cluster data collection that focuses on resources, as well as a local data collection that focuses on the federates, and that is performed if required.

Cluster Data Collection. General information about all monitored resources is collected at the cluster monitoring level. The collected information is comprised only of the total resource load that is consumed during a time interval. With load as the monitoring metric, a simulation-independent balancing scheme becomes possible, and external background load is detected. Such information is collected by accessing the Grid Index Service, which provides monitoring information from the Monitoring and Discovery Service (MDS4) [3]. The MDS is a component of Globus Toolkit and collects information by requesting monitoring data from Ganglia [27] through the GlueCE schema [28].

The load information provided by Ganglia is comprised of the average number of ready processes in the queue of certain CPU. The Ganglia system periodically collects this coarse-grained CPU load information. The number of ready processes can be interpreted as a list of waiting processes that will be executed by a CPU; this list therefore represents the accumulated load of a given processor. However, in light of the differences of computing characteristics from one monitored resource to another, a capacity factor is used in order to normalize resource heterogeneity. Thus, this factor is calculated based on the computing capacity of each resource. In order to obtain such capacities, benchmarks determine the processing power of each managed resource. According to Formula 1, the capacity factor corresponds to the ratio between the common normalizing capacity ($nCap$) and the computing capacity (Cap_i) of each resource. Thus, the *relative load* ($rload_i$) of a resource is obtained by multiplying its current load ($load_i$) by the normalization factor. The common normalizing capacity is known by all balancing elements, and it includes the benchmark result of a previously determined resource.

$$rload_i = \frac{load_i * nCap}{Cap_i} \tag{1}$$

Local Data Collection. In the balancing system, a more detailed data collection from resources determines federate migration properly when imbalances are detected in a distributed system. This data collection is performed in order to retrieve information regarding the communication characteristics of each federate. This specific monitoring information is necessary in

order to identify the migration moves that can benefit the system's load balance and prevent the creation of imbalances in the distributed system. The information is also essential for communication balancing because it describes the interaction patterns of a distributed simulation at any given moment. Thus, at the federate monitoring level, each federate is individually monitored, regardless of the external background load.

For load monitoring, information is collected from each resource with regard to the number of federates running on it and the load they produced at any given time. Federates that are running on a resource are ranked according to the load they produce. This resource consumption is detected through the CPU utilization time that a federate receives during a monitoring interval. The ranking of federates helps the load balancing system determine transfer policies and execute them.

For communication monitoring, a communication table is kept for each federate. The balancing system locally logs all federate communication by registering the destination address, the number of messages transmitted and received, and the size of the messages. In this case, because of the HLA RTI centralized simulation architecture, the destination is the same in all the log tables because all federates communicate only directly to the RTI. Thus, at every monitoring interval, the logged communication information is retrieved and erased from the table in order to prepare for the registration of new data in the next monitoring interval.

4.1.2. Filtering and Selection

Filtering and selection processes are applied to the gathered data to search for determinant aspects of decision factors. These decision factors show the need for load redistribution, which triggers proper balancing actions. The data-filtering process consists of coarse-grained and fine-grained procedures. The Coarse-grained filtering validates the gathered information that pertains only to the balanced resources. Monitoring data related to computing parameters not considered by the balancing scheme is discarded, with only CPU consumption retained. Also, data that is related to resources not managed by the balancing system is eliminated because it misleads the detection and the redistribution of load. The fine-grained filtering checks the resources that can perform migrations for load repartitioning. Data related to overloaded resources is excluded when such resources do not present any simulation entity running on them. After all irrelevant information is removed from the collected data, selection algorithms are applied so as to properly identify load and communication imbalances.

In the case of CPU load metric selection, the selection algorithm identifies load discrepancies through comparisons between the load of each resource and the overall CPU load average. The discrepancies in the data sample indicate overloaded and underloaded resources, which are potential candidates for load migrations. The average load used to determine such discrepancies is obtained by calculating an arithmetic mean. Moreover, thresholds are incorporated to the mean in order to classify the data sample in load categories and to determine the load imbalances more thoroughly.

The thresholds are required in the selection in order to identify when a resource is overloaded, balanced, or underloaded. Such thresholds depend on a specific policy, which determines the efficiency or the performance of the load balancing system. These thresholds delimit top and bottom boundaries, which establish a tolerance for load variances. Such a tolerance grows proportionally to the mean and is defined according to Function 2. The function delimits the tolerance (bds) through three factors: $mean$, β, and α. Because this proportional growth of tolerance can hide considerable load dissimilarities, the factor β is incorporated into the function. Ranging from 0 to 1, this second factor decelerates the tolerance growth, and according to Function 3, the minimum ($\max(wl_i)$) and maximum ($\min(wl_i)$) loads in the gathered data sample are used to identify the load oscillation in β. Through this function, the relative load oscillation (β) limits the tolerance variation, so overloaded and underloaded resources are selected more accurately. Moreover, because the load imbalances are defined according to a criteria specified by given policy, the limiting factor α is required to narrow the balance interval. This factor is inserted in Function 2 and contains a value that ranges from 0 to 1. The value represents the percentage of the average load that is considered balanced in a given load sample.

$$bds = mean * \alpha * \beta \tag{2}$$

$$\beta = \sqrt{\frac{\max(wl_i) - \min(wl_i)}{\max(wl_i)}} \tag{3}$$

In the case of the selection for communication load metrics, a similar approach to load selection is employed. All the federates' communication rates are compared with an average rate. These comparisons reveal the federates that interact the most within the balanced system. An arithmetic mean is computed in order to retrieve the average value. To more accurately identify the communicative federate candidates, thresholds are also employed in these comparisons. Because the communication rate is particularly application-dependent and varies according to each simulation implementation, the thresholds are determined through a method that is based only on the analysis of the collected data sample. The standard deviation of the gathered data sample is employed in the calculation of the superior boundary, which is used to identify communication imbalances caused by interactive simulation federates. Such federates present a differential communication rate, which indicates that the simulation performance might be limited by their communication latency. Thus, these federates are selected as candidates for migration in the next balancing phase.

4.2. Redistribution Phase

Due to the its role in efficiently balancing a distributed load, the redistribution phase is the main part of a dynamic load balancing scheme. Balancing efficiency depends on the relationship between complexity and time; thus, simple redistribution

algorithms are employed. These algorithms must generate a reasonable redeployment that reflects the real workload status of the distributed system. Moreover, computation and communication characteristics of the distributed environment and simulation are considered in order to determine the proper load modifications for minimizing imbalances. The redistribution algorithm assesses computation and communication aspects individually because they reflect different load characteristics. These are classified either resource-centred or federate-centred respectively. Thus, based on the classification performed in the monitoring phase, the redistribution phase determines the migration moves that are needed for better load balance among the available shared resources and for decreased communication latency among the simulation federates.

4.2.1. Redistribution of Load Regarding Computation

As detailed in Algorithms 1 and 2, the redistribution of computational load re-defines the deployment of a distributed simulation by comparing the gathered load resources and by identifying simulation entities to be migrated. The comparison consists of analyzing the computational load of two resources, one of which is underloaded or overloaded. According to this analysis, a migration move is selected by transferring a simulation entity from an overloaded resource to an underloaded resource. A migration move consumes resources and introduces delays in simulations, and precipitated, unnecessary migration moves do not benefit simulation performance. These migrations are generated from the detection of abrupt, short-term load changes, and are avoided in order to minimize balancing system overhead.

Algorithm 1 Pair-Match Load Redistribution Algorithm

```
Require: resources_data, specific_resources_data
  classify(resources_data)
  order_ascendant(underloaded, balanced, overloaded)
  for all src_rsc IN overloaded do
      dst_rsc ⇐ get_next(underloaded, balanced)
      migration_moves ⇐ evaluate(src_rsc, dst_rsc)
  end for
  for all dst_rsc IN underloaded do
      src_rsc ⇐ get_next(overloaded, balanced)
      migration_moves ⇐ evaluate(src_rsc, dst_rsc)
  end for
  return migration_moves
```

Based on the classification of the data sample in the balanced, underloaded, and overloaded categories, the redistribution algorithm reallocates resources. This reallocation occurs according to the following defined premises: to establish migration moves from the most loaded resource to the least loaded resource, to conduct only one migration move for each resource, and to migrate a federate based on its load consumption of the resource and its communication rates in the simulation. In keeping with the first premise, the data sample is organized in a sorted list in ascendant order. The migration pairs are assembled by matching the extremities of this list. The match is performed while any of the sets of underloaded and overloaded categories contain resources to be analyzed. Thus, the search finishes when it is not possible to identify two resources that are different enough in their computational load to justify an effective load migration according to the adopted redistribution policy.

Algorithm 2 Pair-Match Evaluation Algorithm

```
Require: src_rsc, dst_rsc
  if dst_rsc < min then
      if number_fed(src_rsc) > 1 then
          create_migration_move(src_rsc, dst_rsc)
      else if number_fed(src_rsc) = 1 & src_rsc > (min * φ) then
          create_migration_move(src_rsc, dst_rsc)
      end if
  else if (dst_rsc − src_rsc) < (min * δ) then
      if number_fed(src_rsc) > 1 then
          create_migration_move(src_rsc, dst_rsc)
      else if number_fed(src_rsc) = 1 & (dst_rsc − src_rsc) > (min * φ) then
          create_migration_move(src_rsc, dst_rsc)
      end if
  end if
  return migration_move
```

As described in Algorithm 2, each pair of resources is analyzed by comparing the load of each resource in order to match them for load transfer. The comparisons of the pair-match generates a migration move through the analysis conditions related to the amount of computational load and the number of running federates in a resource. The first analysis condition identifies a potential candidate for receiving a federate; the load of the destination resource is then compared with an established minimum load (*min*). After this comparison, a migration move is determined if the number of federates in the source resource is larger than 1. Similarly, if the source resource's number of federates is one and its computational load is larger than the threshold $min * \phi$, a migration move is also determined. The threshold $min * \phi$ corresponds to load for one process in a CPU queue (*min*), smoothed by the parameter ϕ. The second analysis condition determines the difference of load between the source resource and the destination resource. In this load relation, if the difference is larger than the threshold $min * \delta$ and the source resource has more than one federate running, a migration move is created. The threshold corresponds to a value equal to less than the load produced by a process in a CPU queue. If the source has only one federate and the relation is under the threshold $min * \phi$, a migration move is created.

After the migration pair is identified, a federate in the source resource needs to be selected for the migration move. With regard to the defined premises, the migration of the federate must cause smooth changes in the distributed load configuration and not generate additional communication overhead in the balanced simulation. In order to fulfil such requirements, the balancing algorithm analyzes the migration destination resource and its communication latency in order to choose a federate for migration. If the resource's distance is smaller than the source resource's distance, the algorithm selects the most communicative federate for migration. On the other hand, if the resource's distance is larger than the source resource's distance, the algorithm selects the least communicative federate for migration. If both resources have the same communication latency – for instance, when performing local balancing – or if there is more than one federate with the same interactivity rate, the algorithm selects the federate with the least load consumption.

4.2.2. Redistribution of Load Regarding Communication

As delineated in Algorithm 3, the repartitioning of load for communication purposes aims to decrease or minimize communication delays. The repartitioning process identifies the most interactive federates during the balancing cycle and searches for the most appropriate destination resources for them, given the communication topology of the shared resources. After identifying the need for reallocation of resources to decrease communication latencies, an analysis of dispersion helps determine which federates are the most communicative in the collected data sample. In order to identify these communicative candidates, the federates are grouped according to the destination resource with which they interacted the most. In the scope of this work, the grouping of federates is discarded because the communication destination of all federates is the RTI. Moreover, network capacities are not introduced at this stage of design, so only the communication delays caused by the distances between two communicating parts are considered. Thus, the order is used to identify which federates are communicating the most within a given simulation. Consequently, migrating such federates would decrease delays and increase the simulation performance.

Algorithm 3 Communication Redistribution Algorithm

```
Require: current_loads, spec_loads, federates_loads,
  path_distances
  order(federates_loads)
  cmean ⇐ calc_mean(federates_loads)
  cstd ⇐ calc_STD(federates_loads, cmean)
  comm_federates ⇐ find_comm(federates_loads, mean, std)
  for all federate IN comm_federates do
    while !resource_found and path_distances(next) do
      ring ⇐ get_closest_ring(path_distances, RTI)
      resource ⇐ least_load_resource(ring)
      if !overloaded(resource.load) then
        migration_move.add(federate, resource)
        resource_found ⇐ TRUE
      end if
    end while
  end for
  return migration_moves
```

When the list of communicative federates are determined, the redistribution Algorithm 3 subsequently evaluates each of these federates. The algorithm searches for the best destination resources for a communicative federate based on its current location. This algorithm uses a structure called path distances to determine the closest location to which the federate can be transferred, which means that the communication delay can be decreased or minimized. The optimal destination for a federate transfer is the resource that it communicates with the most because the communication latencies are minimized or eliminated. This solution has been proposed in previous works, and it eliminates networking delays by grouping highly interactive federates in the same resource. However, this solution is limited; cannot be applied to HLA simulations based on a centralized RTI because this procedure easily overloads the resource where the RTI is running and diminishes simulation parallelism. Consequently, federates cannot be migrated to the resource where the RTI is running, so other locations that are closer to the RTI than the federate's resource are determined. Thus, the structure of path distances is used to identify the closest resource to the destination resource for a given federate. Such a structure provides a set of rings, and each ring contains a set of eligible destination candidates for the communicative federate. These ring candidates are ordered according to their computational load, so the resource with smallest load can be identified.

After the best resource candidate is found in the distance ring, load comparisons are performed to determine whether a load transfer overloads the candidate resource. These comparisons determine the load characteristics of the destination in regard to the rest of the resources' loads. The destination resource's load is compared with the overall load average of the shared resources. The difference between these two measurements needs to fall within the load relation parameters defined for computational load balancing, so any computational load imbalance is avoided when the load is redistribute for communication purposes. However, a further search in the distance rings occurs if the chosen destination resource cannot receive the federate due to the candidate resource's current load. The next ring in the path distances is therefore selected, and the same process of comparison is performed in order to identify a destination resource that decreases communication latency and does not harm current computation balancing. Thus, the procedure of selecting the next ring is realized until a destination resource is found or until the ring selection reaches the distance ring where the federate's resource currently resides. When the procedure reaches such a ring, the balancing scheme cannot improve the communication latency of the federate for that system's configuration at that moment.

4.3. Migration Phase

For each load transfer determined at the redistribution phase, a migration move is issued to a resource in order to efficiently achieve the expected load balance. Migration latency dictates the number of times that migrations are called by the balancing system since migrations can add considerable cumulative delay to execution time. Federate migration is employed as the tool for migrating load, and it consists of transferring the code and initialization files of a federate to a destination resource, suspending the federate, saving its execution state, transferring it, and restoring its state from the saved status. Even though federate migration is composed of only a few steps, the data transfers introduce overhead. Such transfers can considerably increase latency when performing migration in large-scale distributed systems. Therefore, in order to minimize migration latency and enable better balancing reactivity, a two-phase migration technique is adopted in the redistribution scheme. The technique is similar to the migration techniques proposed by Boukerche and De Grande [29] and by Zengxiang et. al. [30]. In order to minimize the overhead and latency produced by the migration process, this approach is freeze-free, requires minimal external tools, and avoids unnecessary communication and computing.

In the first migration phase, the balancing system transmits all the information that is required to initialize the federate in the remote resource. This information consists of configuration files and other data. After such information is successfully

transmitted, the federate is initiated in the remote resource, so the federate sets all the start-up steps for the next migration phase. All the required static information is reliably transmitted through a staging process using GridFTP [3], and the Web Service Grid Resource Allocation and Management (WS GRAM) [3] is employed to submit the migrating federate to the remote resource. These third-party data transfers and submission mechanisms cause a considerable delay in the simulation. However, such delays are masked because the migrating federate is not suspended while this first migration phase is accomplished.

In the second migration phase, after the federate finishes its initialization step in the remote resource, the communication channel with the RTI is reconfigured to establish a connection with the federate at the new location. Then, the federate in the local resource has its execution suspended for the transfer of its running status, which is saved and transmitted to the federate in the remote resource. The status is comprised of all the necessary dynamic information that is used to keep the federate's running state and the Local Run Time Controller's state. Moreover, messages received after the communication channel with the RTI is exchanged are transmitted to the remote federate, so they can be processed in the proper order, thus avoiding simulation causality inconsistencies. After all the information is successfully transmitted and restored, the federate in the remote location continues its execution.

4.4. General Architecture

As depicted in Figure 2, the architecture of the balancing system consists of components that periodically communicate with the Grid system; detect load discrepancies and trigger the redistribution algorithm; reallocate resources to correct the detected load imbalances; and migrate load to realize established load repartitioning. Generally, the balancing scheme contains Cluster Load Balancers (CLBs), Monitoring Interfaces, Local Load Balancers (LLBs), Local Monitoring Interfaces, and Migration Mechanisms. Furthermore, in order to provide a cross-platform implementation and ease interaction with the adopted HLA RTI distribution, all the architecture is coded in Java.

4.4.1. Cluster Load Balancer

The Cluster Load Balancer (CLB) performs the main tasks of the dynamic load balancing system, executing the balancing algorithm and managing the load of many clusters of resources. In order to organize all the required balancing actions, the CLB is composed of Monitoring, Redistribution, and Migration components. The Monitoring component accesses the Grid Index Service through the Monitoring Interface, which sends queries requesting information about the managed resources. The Monitoring Interface adds modularity and transparency to the balancing system because any monitoring tool can be easily accessed by the balancing system, without requiring knowledge about it. Moreover, the Monitoring component is connected with Local Load Balancers (LLBs) to collect detailed information or with sub-CLBs to collect information from other clusters of resources. Finally, when the monitoring data is gathered, the Monitoring elements perform filtering and selection tasks and trigger the Redistribution component.

The Redistribution component reconfigures the load distribution for the detected load imbalances and the required adjustments determined by the balancing policies. The repartitioning of load involves the computation and communication aspects of a collected data sample. Consequently, resources are classified according to their load and federates are arranged according to their interaction rates. The structure of path distances is used for topology analysis in order to reallocate federates on the shared resources for decreasing communication latencies. After reallocation is performed and migration moves are generated, the Redistribution component emits migration calls to their respective Local Load Balancer (LLB), which forwards the call to the proper Migration Mechanism.

The Path Distances is a data structure that stores information about the communication topology and is employed by the balancing system in order to search for destination resources and perform communication balancing. Based on an analysis of the communication topology, this data structure classifies the resources in distance rings. Resources that have the same distance are grouped in the same ring in the Path Distances. A search for a destination resource becomes possible when such a structure is employed. The distance corresponds to the sum of the number of hops between two resources, including the networking capacity of each hop. Simply put, the ring structure is created just once in the start up of the balancing system since the HLA-based distributed simulations in this scope are based on a centralized RTI.

In order to facilitated the management of large-scale distributed systems, a multi-layered hierarchical design is adopted for organizing the balancing system, as presented in the Figure 3. In the hierarchical architecture, a CLB can manage a list of LLBs and a list of sub-CLBs and can be managed by a upper CLB. A CLB controls a set of resources, which are managed directly by a list of LLBs. At the bottom layer of the hierarchy, a CLB controls only LLBs and, if a upper CLB exists, reports the collected data to its upper CLB when performing global balancing. At the intermediate layers, a CLB gathers information from a list of sub-CLBs so as undertake global balancing. At the top of the hierarchy, the root CLB is responsible for gathering all the collected data from its LLBs and sub-CLBs, performing the redistribution procedure, and reporting to other CLBs the load moves.

Therefore, the hierarchical structure of the balancing system minimizes the system's communication overhead and facilitates management of resources and simulations. Communication overhead is caused by the periodical data gathering needed from a large number of resources and federates. Filtering and selections performed by each CLB in the hierarchical structure decreases the amount of transferred information to be analyzed. This reduction of transferred information becomes vital for large-scale systems. The hierarchical structure also facilitates the issue of redistribution, or migration, calls to each federate, therefore avoiding the need for a centralized element reaching all simulation parts in order to execute the required load modifications.

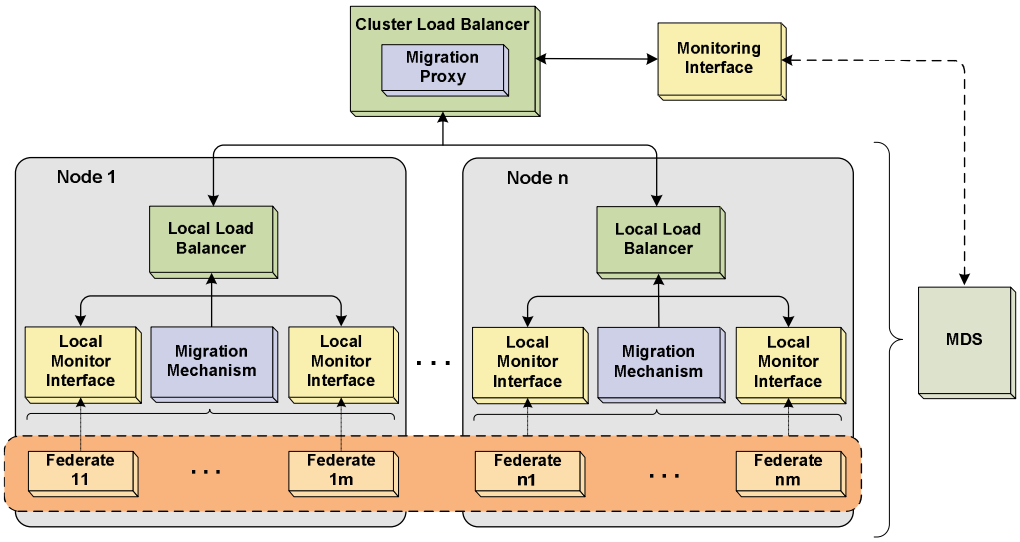

Figure 2: The Dynamic Load Balancing's General Architecture

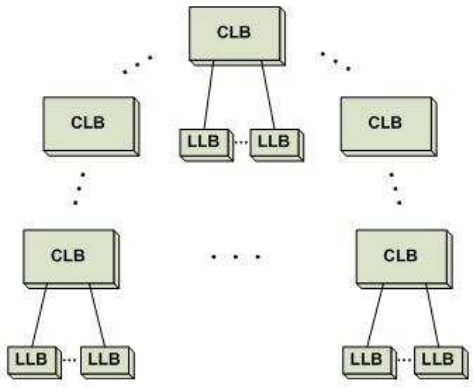

Figure 3: The Dynamic Load Balancing's Hierarchical Architecture

4.4.2. *Local Load Balancer*

The Local Load Balancer (LLB) corresponds to an extension of CLB that acts locally in each resource managed by the load balancing system. The LLB is responsible for requesting load information from a federate, providing this information to the Monitoring component in the CLB, and executing the commands sent by a CLB. The local resource's information is gathered through direct access to the Local Monitor Interface, which monitors each federate individually through a specific java library. The library is a Java monitoring library interface; it is called *ThreadMXBean* and belongs to a Java library. In order to collect a federate's CPU utilization time, the method *getProcessCpuTime()* from the Java library, is called. Also, when a LLB receives a migration command, it informs the Migration Mechanism about the federate that is required to migrate and about the destination resource.

4.4.3. *Migration Mechanism*

The Migration Mechanism is responsible for managing the migration procedure according to the defined migration steps in the Migration Phase. The Simulation Agent is the main element of the Migration Mechanism; it supports all the migration tasks in order to keep simulation causality consistent. Thus, the agent manages communication transference in both parts of a migrating simulation entity, so the migration process is imperceptible to the simulation entities. The agent is also responsible for managing all the required transfers of a federate's code, its initialization files, and its running status data. In order to perform such tasks, the agent consists of a Migration Manager and a Communication Manager. The Migration Manager orchestrates all the migration steps so that it does not loose or disrupt simulation events. The manager triggers the Communication Manager and requests a federate's simulation state by calling an interfacing method named *federateSave*. This method retrieves a federate's running status and LRC. The Migration Manager also launches the simulation manager at the remote resource designated by the LLB and manages the information exchange to restore the migrating federate completely. The Communication Manager organizes all the message exchanges of a federate during migration. The manager stores the incoming messages in a queue, which is transferred to the destination resource.

Furthermore, as depicted in Figure 2, the Migration Proxy is used as an intermediate element in the migration procedure to help transfer the execution state and the incoming messages of migrating federates. The proxy is employed when the destination resource is located outside the source resource's cluster. This means that the destination resource cannot be reached directly by the source resource. This element establishes connections with the two federate migration parts in order to receive, store, and forward the migration data.

Algorithm 4 Global Dynamic Load Balancing Algorithm

```
path_dist ⇐ analysis(resources_topology)
loop
    loads ⇐ query_MDS()
    current_loads ⇐ filter_MDS_data(loads)
    if balancing_globally then
        spec_loads ⇐ request_LLBs(current_loads)
        if root then
            for all sub_CLB IN list_CLB do
                ext_loads ⇐ request_loads()
            end for
            merge(ext_loads, current_loads)
            for all sub_CLB IN list_CLB do
                ext_spec_loads ⇐ request_spec_loads()
            end for
            spec_loads ⇐ merge(ext_spec_loads, spec_loads)
            feds ⇐ communicative(spec_loads, path_dist)
            if feds ≠ ∅ then
                mig_moves ⇐ redistribute_comm()
            end if
            mng_loads ⇐ filter(current_loads, spec_loads)
            mean, bds ⇐ calculate_mean_bds(mng_loads)
            over, under ⇐ select(mng_loads, mean, bds)
            if over ≠ ∅ ∧ under ≠ ∅ then
                mig_moves ⇐ redistribute_comp()
            end if
        else
            for all sub_CLB IN list_CLB do
                ext_loads ⇐ request_loads()
            end for
            merge(ext_loads, current_loads)
            send_upper(current_loads)
            for all sub_CLB IN list_CLB do
                ext_spec_loads ⇐ request_spec_loads()
            end for
            spec_loads ⇐ merge(ext_spec_loads, spec_loads)
            send_upper(spec_loads)
            mig_moves ⇐ retrieve_mig_moves()
        end if
    else
        overload_cand ⇐ select_overloaded(current_loads)
        spec_loads ⇐ request_LLBs(overload_cand)
        mng_loads ⇐ filter(current_loads, spec_loads)
        mean, bds ⇐ calculate_mean_bds(mng_loads)
        over, under ⇐ select(mng_loads, mean, bds)
        if over ≠ ∅ ∧ under ≠ ∅ then
            mig_moves ⇐ redistribute_comp()
        end if
    end if
    for all move IN mig_moves do
        if internal(move) then
            send_migration_move(move)
        else
            forward_migration_move(move)
        end if
    end for
    balancing_globally ⇐ choose_scope()
    wait( Δ )
end loop
```

4.5. Load Balancing Algorithm

The resource monitoring, load redistribution, and federate migration phases are merged in a dynamic load balancing algorithm, as detailed in Figure 4. In every balancing cycle, the algorithm triggers the monitoring of resources in order to detect load imbalances. If any load repartitioning is required, load redistribution is invoked and migration moves are performed. The balancing is initiated periodically, and a balancing cycle occurs in periodic Δ intervals. The interval is limited to the frequency in which the MDS service updates its monitoring database, every 20 seconds, adding minimal overhead to the distributed system and dictating the reactivity of the balancing scheme.

As delineated in Algorithm 4, the distributed load is balanced in local or global cycles. The balancing system performs computation and communication load redistribution in the global scope, considering that the communication balancing is accomplished first. Communication balancing is only realized globally because a global view of the entire system is required for determining communication rate differences. On the other hand, computation balancing is performed both locally and globally. In both communication and computation balancing, local and cluster data collections are necessary for the monitoring phase. The cluster data gathering is triggered as soon as the load balancing cycle starts in order to obtain an overall overview of resources. After collection, the cluster data is filtered, and a gathering of specific data is performed with all resources or with a defined group of resources. This group is based on the median calculated with the cluster data during local balancing. After data is collected, selection is initialized. In the selection step of the monitoring phase, communication balancing identifies the communicative federates while computation balancing determines load imbalances within the shared resources.

The respective redistribution algorithm is triggered to determine the possible migration moves, which depend on the type of balancing that is performed. This occurs if any computation or communication imbalance is detected. When global balancing is realized, the root CLB gathers all migration moves and sends them to their respective destinations: to its LLBs or sub-CLBs. This forwarding occurs with each CLB until all migration moves reach the bottom of the hierarchy: an LLB. Moreover, only one move involving a resource is allowed per balancing cycle; thus, the balancing system reacts gradually to load changes by correcting load distribution and evaluating the effects of the redistribution. After the migration moves are produced, they are forwarded to their respective migration mechanisms, which are responsible for triggering and managing the migration procedure. A migration move is then triggered locally in a source resource. The migration process occurs independently from the load balancing cycle. In order to exclude the load and latencies caused by migrations, the balancing system ignores the data collected from the resources selected for migration in the previous balancing cycle. Consequently, enough time is provided to these resources to stabilize their processing of a new load reconfiguration.

5. Experiments and Results

Experiments have been accomplished in order to evaluate the effectiveness of the proposed dynamic load balancing scheme. The experiments have shown the balancing system's ability to detect and react to load imbalances, thus improving performance and decreasing simulation times. A large-scale, heterogeneous, non-dedicated distributed environment has been used to support the experiments. As summarized in Table 1, this environment has been composed of a Dell cluster consisting of 24 nodes, an IBM cluster consisting of 32 nodes, and a fast-Ethernet link connecting them. In the Dell cluster, each node was comprised of a Quad-core 2.40 GHz Intel(R) Xeon(R) CPU and 8 gigabytes of DIMM DDR RAM memory. All nodes were interconnected through a Myrinet optical network that allowed data transmission up to 2 gigabits per second. In the

Table 1: Physical Environment's Configuration

	IBM Cluster	**Dell Cluster**
Nodes	32	24
CPU	C2D 3.4 GHz	Quad 2.40 GHz
Memory	2 gigabytes	8 gigabytes
Network	gigabyte Ethernet	2-gigabit Myrinet

Table 2: Dynamic Balancing System's Configuration

α	min	δ	ϕ
0.15	150	8	6

Table 3: General Parameters for the Experimental Simulations (Fed=federate, Objs=Objects, TS=time steps)

	Feds	**Objs/Fed**	**TS**
Computation	1 – 1000	1	100
Communication	500	1 – 1000	100
Comp./Comm.	1 – 600	100	100

IBM cluster, each node consisted of a Core 2 Duo 3.4 GHz Intel(R) Xeon(R) CPU and 2 gigabytes of DIMM DDR RAM, and a gigabyte Ethernet network connecting the cluster's nodes. Furthermore, the experiments have been supported by the HLA platform with an RTI version 1.3 and the Globus Toolkit 4.2.1, having Linux as the operating system in both clusters.

As a common simulation scenario in these experiments, simulation federates were deployed on the 56 nodes. It must be taken into consideration that one node was totally dedicated to run the HLA RTI executive. The values presented in Table 2 correspond to the load balancing system's configuration parameters. The LLBs of the balancing system were placed in every cluster node except the node where the RTI was running; moreover, a CLB was placed in one node of each cluster. Additionally, in the experimental scenario, federates coordinated the training operations of two teams of interactive tanks in a two-dimensional routing space in time-stepped simulations. Such federates performed computing-intensive calculations to produce the tanks' movements, published the tanks' positions, and subscribed to interest regions. The HLA simulations were composed of a range of 1 to 1000 federates, running for 100 time steps. Each federate managed 1 to 500 tanks. The parameters of these simulations varied according to three test case groups and are summarized in Table 3.

In the experimental analyses, the proposed dynamic balancing scheme was compared with the base line. The base line is obtained by statically and evenly distributing the simulation federates on the available resources and by running the simulations without any balancing. The resources' heterogeneity generates computation imbalances and the network distances produces communication delays. In the case of the dynamic balancing scheme, the same deployment of federates is used, but the balancing system modifies the load distribution as needed. The dynamic modifications of distributed load can be observed in the improvement in performance, but they are better understood through the number of migrations that were performed by the balancing system. Thus, in the graphs, the right Y axis represents the execution time of the simulations while the left X axis shows the number for migrations. The combination of the information contained in the left Y axis, the Migration curves, and the Dynamic Balancing curves exposes the amount of redistribution effort for achieving a proper balancing.

5.1. Computation Load Balancing

In this test case group, the effectiveness of the dynamic load balancing system in detecting and reacting to computational load was observed as distributed load imbalances occurred. Thus, the first analysis reveals the reactivity of the balancing system to the irregular deployment of computational load and dynamic load variations. The second test case group indicates external background load inserted into the system randomly and dynamically.

5.1.1. Reactivity to Load Imbalances

In this analysis, all the experimental simulations were distributed based on an even initial static partitioning. However, imbalances decreased the simulation's performance due to the heterogeneity of the shared resources and the dynamic computational load changes. The simulations were composed of 1 to 1000 federates, which produced a continuous load or a load that changed periodically, resulting in negligible to intensive computing.

According to the graph in Figure 4a, the balancing scheme reacted to an uneven partitioning of load. The reaction to imbalances normalized the computational load over heterogeneous resources and reduced the simulation time. As shown by the non-existence of migration, the balancing scheme did not redistribute the load for experiments of less than 100 federates because the simulations did not require any load balancing. A noticeable improvement was produced for the experiments with more than 100 federates since the load generated was large enough to produce imbalances. When the experiments were composed of more than 900 federates, the distributed system reached a load saturation point. In this case, all the shared resources were overloaded, and no redistribution of load could improve the simulation's performance. This behaviour is also present in the experiments depicted in figures 4b, 5a, and 5b.

When observing reactivity to dynamic load changes, as presented in Figure 4b, the redistribution scheme detected the load modifications during run time and performed the proper load redeployment. This caused an improvement in the simulation's performance similar to the improvement in the previous analysis. In this experiment, there is a larger variance of simulation times for both unbalanced and balanced curves. This variance is caused by the random, dynamic computational load changes produced by each federate in the simulations.

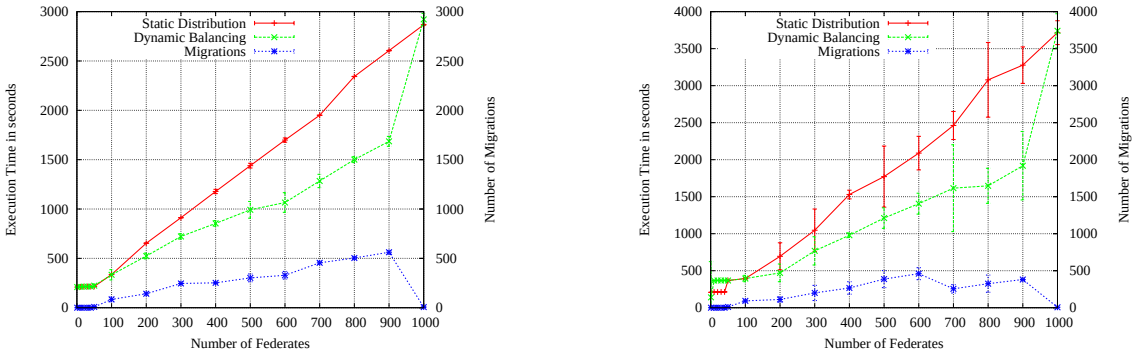

Figure 4: The Dynamic Balancing Scheme versus a Static Partitioning for an Increasing Number of Federates: (a) constant load and (b) dynamic load changes

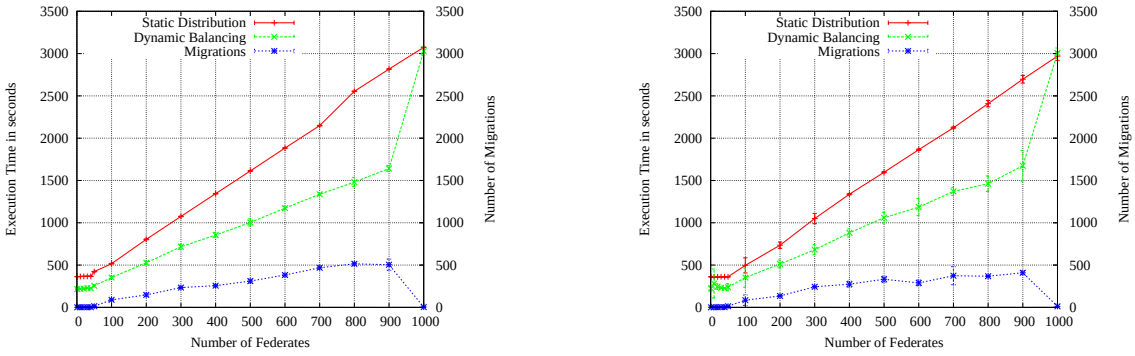

Figure 5: Capacity of the Balancing Scheme to Detect External Load for an Increasing Number of Federates: (a) specific resource with constant external load and (b) random and dynamic move of external load

5.1.2. Detection of External Background Load

In order to measure the capability of the load balancing system in detecting and reacting to an external background load, the previous experiments' scenario was extended by adding an external application in one resource. The application introduced a controlled amount of load to overload a resource. The external application was also dynamically and randomly placed on the shared resources.

As shown in Figure 5a, the introduction of an external load led to increased addition of execution time. This is noticeable in the experiments with 1 to 100 federates. When there were over 100 federates, the load produced by the federates in the system surpassed the external application's overhead, but it continued to produce an offset in the simulation times when compared with Figure 4a. The graph in Figure 5b describes the capacity of the balancing system to detect external load that appeared dynamically and periodically on the shared resources. The results indicate a variance in the simulation times due to the load oscillation caused by the dynamic movement of external load.

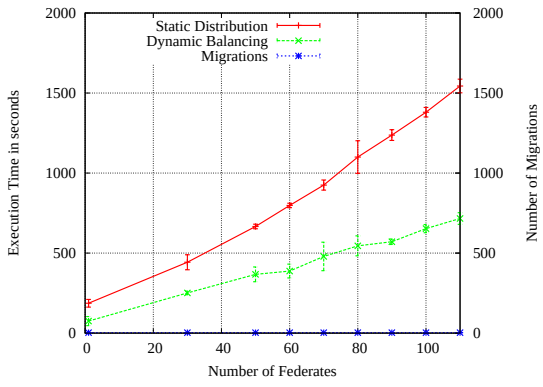

Figure 7: Communication Balancing with Dynamic Communication Load

5.2. Communication Load Balancing

The ability of the balancing scheme to react to communication load imbalances is evaluated in this test case group. The analysis examines the reaction of the dynamic balancing system when redistributing the load of simulations containing spe-

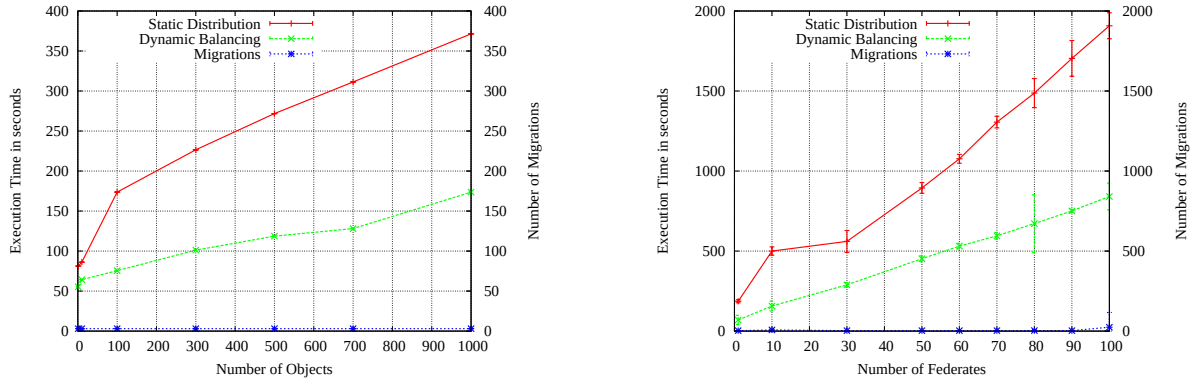

Figure 6: Communication Balancing of Increasing Communication Load in Simulations with 500 Federates: (a) Increasing number of objects and (b) Increasing number of communicative federates

cific federates with more communication or with an increasing amount of communication. The experiments observed the execution time of simulations composed by 500 federates. Among these federates, some published and subscribed to special objects with large amounts of data to produce the required intensive communication.

According to the graph in Figure 6a, the proposed balancing system identified communication load imbalances in HLA simulations. These simulations contained one federate with additional communication, an increasing the number of object updates. Such an experiment demonstrated the capacity to detect communication overhead caused by a highly interactive federate. This detection resulted in the constant number of federate migrations per simulation. Moreover, in a scenario with an increasing number of federates with 100 objects, as shown in Figure 6b, the balancing system successfully detected the communicative federates in the simulation. The balancing reduced the network latency of simulations with intensive communication and also decreased the simulation times. As shown by the curve, the improvement presents a linear trend resulting from the communication overhead of the RTI. Finally, the graph in Figure 7 shows successful repartitioning according to dynamic communication load changes in the experimental simulations. Such simulations were composed of 100 federates that produced a random amount of interactions periodically.

In the three graphs, the dynamic load redistribution for minimizing of communication delay effectively reduced the simulation time of the experiments. However, as observed in the graphs, the balancing system employed a constant, small number of migrations to perform such a redistribution of load, even for large-scale experiments. This characteristic results from the use of standard deviation to detect communicative federates. The detection of communication balancing does not identify all the communicative federates when some federates with extremely high communication rates are present in the simulations. This technique reacts properly to imbalances but is limited in specific simulation cases, such as the described communication experiments.

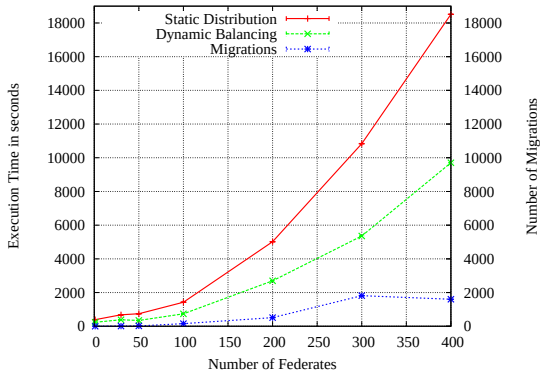

Figure 8: Dynamic Balancing of Constant Communication and Computational Load

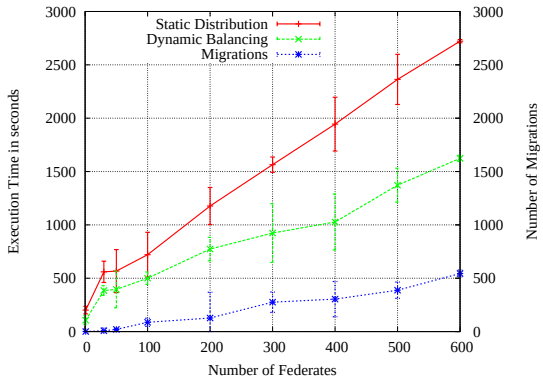

Figure 9: Dynamic Balancing of Communication and Computational Load with Dynamic Changes

5.3. Computation and Communication Balancing

In this test case group, the dynamic balancing of both computational and communication loads is observed in two scenarios. The test case meant to analyze the balancing system's efficiency when computation and communication imbal-

ances were present in distributed environments and simulations. For both scenarios, federates that perform computation- and communication-intensive tasks are evenly placed on the system. Moreover, an external background process is inserted in determined resources. In the dynamic scenario, the external application was moved periodically and randomly on the shared resources, producing a considerable load, and the federates' computational and communication load changed periodically and independently.

In the first scenario, as described in Figure 8, the experiment shows that the proposed dynamic load balancing system successfully reorganizes an imbalanced load. The decrease of execution time indicates that the balancing system detects the load differences produced by external applications, computational load, and communication latencies. However, experiments with over 400 federates show a decrease in performance gain due to the saturation of load in the distributed system, which is confirmed by a reduction in the number of migrations. In the second scenario, as depicted in Figure 9, the observed distributed system produces dynamic load changes, which are successfully detected by the balancing system. The dynamic variations can be identified in the graph by the increase in confidence intervals and by the decrease in execution times. As a result, the dynamic balancing system proved able to determine load imbalances and to redistribute the load accordingly, so better utilization of resources was reached.

6. Conclusions and Future Work

In order to evenly distribute the load of large-scale HLA-based simulations on non-dedicated, heterogeneous environments when computational and communication imbalances are present, a hierarchical dynamic load balancing scheme is proposed in this paper. In order to support such simulations, the balancing approach is composed of three sequential phases with a tree structure that detects imbalances, re-partitions the distributed load, and migrates federates. The system also accesses Grid services to perform reliable load transfers, as well as monitoring that detects external background in shared resources. The hierarchical architecture minimizes the overhead produced by the balancing system, and benchmarking is employed in order to overcome the heterogeneity issues present in shared resources. As a result, the proposed balancing scheme was able to decrease the execution times in the experimental simulations by improving the use of available resources.

In future work, additional analysis will be performed to develop an improved technique for detecting communicative federates, thus minimizing the detection aspect of the communication balancing's limitations. Moreover, further studies regarding different communication and computation balancing techniques will be performed in order to achieve better detection of and reactivity to load imbalances. Such schemes involve the use of monitoring tools other than Grid MDS, the incorporation of federate migration latencies in the load redistribution's decision-making mechanism, and the development of other communication rearrangement policies, leading to greater performance in HLA-based simulations.

Acknowledgments

This work is partially supported by NSERC, the Canada Research Chair program, ORF funds, and EAR Research Award.

References

[1] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: Enabling scalable virtual organizations, International Journal of High Performance Computing Applications 15 (3) (2001) 200–222.

[2] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke, Grid services for distributed system integration, Computer 35 (6) (2002) 37–46.

[3] Globus, University of Chicago, 7 Feb. 2008.
URL `http://www.globus.org/`

[4] J. Nabrzyski, J. M. Schopf, J. Weglarz, Grid Resource Management: State of the Art and Future Trends, Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[5] S. I. S. C. (SISC), Ieee standard for modeling and simulation (m&s) high level architecture (hla) framework and rules, IEEE Computer Society (September 2000).

[6] D. W. Glazer, C. Tropper, On process migration and load balancing in time warp, IEEE Transactions on Parallel and Distributed Systems 4 (3) (1993) 318–327.

[7] C. Burdorf, J. Marti, Load balancing strategies for time warp on multi-user workstations, The Computer Journal 36 (2) (1993) 168–176.

[8] R. Schlagenhaft, M. Ruhwandl, C. S. H. Bauer, Dynamic load balancing of a multi-cluster simulator on a network of workstations, in: Proc. of the 9th workshop on Parallel and distributed simulation, IEEE Computer Society, 1995, pp. 175–180.

[9] C. D. Carothers, R. M. Fujimoto, Background execution of time warp programs, in: Proc. of the 10th Workshop on Parallel and Distributed Simulation, IEEE Computer Society, 1996, pp. 12–19.

[10] P. Peschlow, H. Honecker, P. Martini, A flexible dynamic partitioning algorithm for optimistic distributed simulation, in: Proc. of the 21st Workshop on Parallel and Distributed Simulation (PADS07), IEEE Computer Society, 2007, pp. 219–228.

[11] E. Deelman, B. K. Szymanski, Dynamic load balancing in parallel discrete event simulation for spatially explicit problems, in: Proc. of the 12th workshop on Parallel and distributed simulation, IEEE Computer Society, 1998, pp. 46–53.

[12] L. F. Wilson, W. Shen, Experiments in load migration and dynamic load balancing in speedes, in: Proc. of the 1998 Winter Simulation Conference, IEEE Computer Society, 1998, pp. 483–490.

[13] A. Boukerche, S. K. Das, Dynamic load balancing strategies for conservative parallel simulations, in: Proc. of the 11th Workshop on Parallel and Distributed Simulation (PADS97), IEEE Computer Society, 1997, pp. 32–37.

[14] E. E. Ajaltouni, A. Boukerche, M. Zhang, An efficient dynamic load balancing scheme for distributed simulations on a grid infrastructure, in: Proc. of the 12th 2008 International Symposium on Distributed Simulation and Real-Time Applications, IEEE Computer Society, 2008, pp. 61–68.

[15] M. Choe, C. Tropper, On learning algorithms and balancing loads in time warp, in: Proc, of the 13th workshop on Parallel and distributed simulation, IEEE Computer Society, 1999, pp. 101–108.

[16] B. P. Gan, Y. H. Low, S. Jain, S. J. Turner, W. C. W. J. Hsu, S. Y. Huang, Load balancing for conservative simulation on shared memory multiprocessor systems, in: Proc. of the 14th workshop on Parallel and distributed simulation, IEEE Computer Society, 2000, pp. 139–146.

[17] H. Avril, C. Tropper, The dynamic load balancing of clustered time warp for logic simulation, in: Proc. of the 10th Workshop on Parallel and Distributed Simulation, IEEE Computer Society, 1996, pp. 20–27.

[18] Z. Xiao, B. Unger, R. Simmonds, J. Cleary, Scheduling critical channels in conservative parallel discrete event simulation, in: Proc. of the 13th workshop on Parallel and distributed simulation, IEEE Computer Society, 1999, pp. 20–28.

[19] L. Bononi, M. Bracuto, G. D'Angelo, L. Donatiello, A new adaptive middleware for parallel and distributed simulation of dynamically interacting systems, in: Proc. of the 8th International Symposium on Distributed Simulation and Real-Time Applications, IEEE Computer Society, 2004, pp. 178–187.

[20] A. Boukerche, C. Tropper, A static partitioning and mapping algorithm for conservative parallel simulations, in: Proc. of the 8th workshop on Parallel and distributed simulation, IEEE Computer Society, 1994, pp. 164–172.
[21] A. Boukerche, An adaptive partitioning algorithm for conservative parallel simulation, in: Proc. of the 15th International Parallel and Distributed Processing Symposium, IEEE Computer Society, 2001, pp. 133–138.
[22] M. Y. H. Low, Dynamic load-balancing for bsp time warp, in: Proc. of the 35th Annual Simulation Symposium (SS02), IEEE Computer Society, 2002, pp. 267–274.
[23] J. Luthi, S. Grossmann, The resource sharing system: dynamic federate mapping for hla-baseddistributed simulation, in: Proc. of the 15th Workshop on Parallel and Distributed Simulation, IEEE Computer Society, 2001, pp. 91–98.
[24] W. Cai, S. J. Turner, H. Zhao, A load management system for running hla-based distributed simulations over the grid, in: Proc. of the 6th International Workshop on Distributed Simulation and Real-Time Applications, IEEE Computer Society, 2002, pp. 7–14.
[25] K. Zajac, M. Bubak, M. Malawski, P. Sloot, Towards a grid management system for hla-based interactive simulations, in: Proc. of the 7th International Symposium on Distributed Simulation and Real-Time Applications, IEEE Comp. Society, 2003, pp. 4–11.
[26] G. Tan, K. C. Lim, Load distribution services in hla, in: Proc. of 8th IEEE Distributed Simulation and Real-time Applications, IEEE Computer Society, 2004, pp. 133–141.
[27] M. L. Massie, B. N. Chun, D. E. Culler, The ganglia distributed monitoring system: design, implementation, and experience, Parallel Computing 30 (7) (2004) 817–840.
[28] Ws mds: Cluster monitoring information and the glue resource property, The Globus Toolkit, 7 Feb. 2008.
URL `http://www.globus.org/toolkit/docs/4.0/info/key/gluerp.html`
[29] A. Boukerche, R. E. D. Grande, Optimized federate migration for large-scale hla-based simulations, in: Proc. of the 12th International Symposium on Distributed Simulation and Real-Time Applications, IEEE Computer Society, 2008, pp. 227–235.
[30] Z. Li, W. Cai, S. J. Turner, K. Pan, Federate migration in a service oriented hla rti, in: Proc. of the 11th International Symposium on Distributed Simulation and Real-Time Applications, IEEE Computer Society, 2007, pp. 113–121.

Robson Eduardo De Grande received his B. Sc. and M.Sc. degrees in Computer Science from the Federal University of Sao Carlos, Brazil in 2004 and 2006, respectively. He is now a Ph.D. candidate at PARADISE Research Laboratory at University of Ottawa, Canada. His research interests include load balancing, distributed simulations, and performance evaluation.

Azzedine Boukerche is a full professor and holds a Canada Research Chair position at the University of Ottawa (uOttawa). He is the founding director of the PARADISE Research Laboratory, School of Information Technology and Engineering (SITE), Ottawa. Prior to this, he held a faculty position at the University of North Texas, and he was a senior scientist at the Simulation Sciences Division, Metron Corp., San Diego. He was also employed as a faculty member in the School of Computer Science, McGill University, and taught at the Polytechnic of Montreal. He spent an year at the JPL/NASA-California Institute of Technology, where he contributed to a project centered about the specification and verification of the software used to control interplanetary spacecraft operated by JPL/NASA Laboratory. His current research interests include wireless ad hoc and sensor networks, wireless networks, mobile and pervasive computing, wireless multimedia, QoS service provisioning, performance evaluation and modeling of large-scale distributed systems, distributed computing, large-scale distributed interactive simulation, and parallel discrete-event simulation. He has published several research papers in these areas. He served as a guest editor for the Journal of Parallel and Distributed Computing (special issue for routing for mobile ad hoc, special issue for wireless communication and mobile computing, and special issue for mobile ad hoc networking and computing), ACM/Kluwer Wireless Networks, ACM/Kluwer Mobile Networks Applications, and Journal of Wireless Communication and Mobile Computing. He serves as an Associate Editor of IEEE Transactions on Parallel and Distributed systems, IEEE Transactions on Vehicular Technology, Elsevier Ad Hoc Networks, Wiley International Journal of Wireless Communication and Mobile Computing, Wileys Security and Communication Network Journal, Elsevier Pervasive and Mobile Computing Journal, IEEE Wireless Communication Magazine, Elseviers Journal of Parallel and Distributed Computing, and SCS Transactions on Simulation. He was the recipient of the Best Research Paper Award at IEEE/ACM PADS 1997, ACM MobiWac 2006, ICC 2008, ICC 2009 and IWCMC 2009, and the recipient of the Third National Award for Telecommunication Software in 1999 for his work on a distributed security systems on mobile phone operations. He has been nominated for the Best Paper Award at the IEEE/ACM PADS 1999 and ACM MSWiM 2001. He is a recipient of an Ontario Early Research Excellence Award (previously known as Premier of Ontario Research Excellence Award), Ontario Distinguished Researcher Award, and Glinski Research Excellence Award. He is a cofounder of the QShine International Conference on Quality of Service for Wireless/Wired Heterogeneous Networks (QShine 2004). He served as the general chair for the Eighth ACM/IEEE Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, and the Ninth ACM/IEEE Symposium on Distributed Simulation and Real-Time Application (DS-RT), the program chair for the ACM Workshop on QoS and Security for Wireless and Mobile Networks, ACM/IFIPS Europar 2002 Conference, IEEE/SCS Annual Simulation Symposium (ANNS 2002), ACM WWW 2002, IEEE MWCN 2002, IEEE/ACM MASCOTS 2002, IEEE Wireless Local Networks WLN 03-04; IEEE WMAN 04-05, and ACM MSWiM 98-99, and a TPC member of numerous IEEE and ACM sponsored conferences. He served as the vice general chair for the Third IEEE Distributed Computing for Sensor Networks (DCOSS) Conference in 2007, as the program cochair for GLOBECOM 2007-2008 Symposium on Wireless Ad Hoc and Sensor Networks, and for the 14th IEEE ISCC 2009 Sym-

posium on Computer and Communication Symposium, and as the finance chair for ACM Multimedia 2008. He also serves as a Steering Committee chair for the ACM Modeling, Analysis and Simulation for Wireless and Mobile Systems Conference, the ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks, and IEEE/ACM DS-RT.