Dynamic Load Balancing Using Grid Services for HLA-Based Simulations on Large-Scale Distributed Systems

Azzedine Boukerche and Robson Eduardo De Grande
PARADISE Lab - School of Information Technology and Engineering
University of Ottawa - Canada
Email: boukerch@site.uottawa.ca, rdgrande@site.uottawa.ca

***Abstract*—HLA-based simulations, as any distributed computing application, can undergo critical performance issues due to load imbalances on large-scale, heterogeneous, non-dedicated distributed systems. Such imbalances are produced by HLA simulation entities that can dynamically change their computation and communication load during their execution time, so an initial static load deployment is incapable of providing simulations complete and even distributed resources usage. Moreover, because the computing resources are non-dedicated, unknown external applications can generate load for any computing resource, increasing the imbalances' unpredictability. Thus, in order to re-allocate resources for an HLA simulation during its execution time, an hierarchical dynamic load balancing system is introduced. The system manages a simulation's workload by monitoring the distributed load through the MDS Grids' service; by identifying load imbalances according to a load sharing policy; by re-allocating resources according to defined policies; and by migrating federates through the GRAM Grids' service, a migration proxy, and peer-to-peer state transfer. By keeping the load evenly partitioned on the distributed system, such a devised system successfully improved the simulations' performance. The experimental results and comparative analyses between balanced and non-balanced simulations proved the efficiency of the proposed dynamic load balancing system.**

***Index Terms*—Parallel Simulations, High Level Architecture, Dynamic Load Balancing, Performance.**

I. Introduction

Dynamic load re-partitioning is an essential property for HLA-based simulations since the performance of such simulations is affected by uneven distribution of load on shared resources during run-time. Re-partitioning improves the efficiency in which the resources are allocated and consequently increases the performance of a distributed simulation. Since the static partitioning is a limited prediction and presents a deployment that influences positively only the start-up performance, it hardly reaches efficient resource usage due to dynamic changes that might occur during run-time. Moreover, in order for a balancing system to be able to re-distribute load properly on large-scale systems, non-dedicated, heterogeneous resources must be managed. Therefore, to suit the simulation elements according to their resource requirements, a dynamic load balancing of HLA-based simulations is considered, introducing a near optimal usage of resources by re-allocating them on-demand and evenly to HLA federates.

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

The High Level Architecture (HLA) standard [13] was developed to coordinate parallel and distributed simulations. In order to maximize parallelism and reduce execution time, simulations are divided into several smaller independent entities, which are allocated on available shared resources accordingly. Because independent interactions between simulation entities are allowed, simulation causality consistency errors might happen, requiring certain advanced management. The HLA specification presents a framework that provides mechanisms to organize parallel and distributed simulations and consequently avoid such inconsistencies. However, the HLA standard is unaware of issues regarding the distributed resources that are allocated to run simulations. The HLA management services are restricted to deal only with simulation coordination, so the standard lacks means to prevent performance issues and the unreliability of resources. Thus, when simulations are placed in distributed environments, which are mostly composed of unreliable and non-dedicated resources, performance issues are likely to occur, requiring load balancing techniques and resource management mechanisms.

Since there exists a need for coordinating shared resources and scheduling distributed HLA simulation load, a resource management system emerges as vital for dynamically balancing distributed load. Grid is a widely used system to manage distributed applications on shared resources and provides several services. Grid computing was first devised by Foster [16] as a coordinated resource sharing system involving resources, individuals, and institutions, aiming at flexibility and security. Globus Toolkit [9] is a de facto middleware standard for Grid computing and is based on Open Grid Services Architecture [17], which combines Web Services in the Grid architecture. Even though the Grid services range the monitoring of resources and applications, the scheduling and allocation of resources, and the identification of application requirements [14], they do not solve the imbalance issues but provide elemental tools for a load management system.

Due to the critical importance of load balancing to achieve good resources utilization and fast processing for any distributed simulation, several approaches exist that attempt to manage the distributed simulations' load dynamically. Generally, in order to react properly to load changes in a distributed application, a load balancing system is composed of sensing mechanisms to gather monitoring information, load redistribution policies to identify imbalances and correct them, and load transfer techniques to execute the needed deployment changes. The load balancing system is also developed by considering a range of aspects inherent to distributed systems, which involve the resource heterogeneity, background load, and computing and communication workload. Each proposed approach, in its design, considers a different set of aspects depending on the type of system it is managing. However, neither of the solutions in the literature completely concerns the aspects that best suit HLA simulations running on large-scale systems composed of distributed shared resources.

In this paper, a new load balancing scheme is proposed, aiming to provide a solution that can lead to a performance improvement for unbalanced HLA-based simulations running on large-scale, non-dedicated, heterogeneous distributed environments. As with other load balancing approaches, the scheme basically consists of monitoring, load scheduling, and migration, but it employs mechanisms to allow scalability, detect external load, consider the differences of resource capacities, and provide low latency migration. The scheme minimizes the overhead caused by the balancing system through an hierarchical architecture and normalizes the workload of all monitored resources instead of attempting to decrease the load of an overloaded resource individually. Moreover, due to their benefits, Grid services are introduced in the scheme's architecture to support the balancing system to scale accordingly, to overcome most of the heterogeneity issues faced by other balancing schemes, and to treat the unreliability of resources during data transfer for migration.

The remainder of the paper is organized as follows. In section 2, the related work is described and existing issues are identified. In section 3, the proposed three-phase dynamic load balancing scheme, as well as its architecture, is introduced. In section 4, two test-case groups of experiments are outlined, and the experimental results are shown and discussed. Section 5 presents the conclusion and directions for future work.

II. Related Work

Since dynamic load balancing improves systems' performance, many approaches have been proposed, focusing on solving uneven load partition problems for applications distributed on shared resources. For distributed simulations, the existent approaches observe simulation specific characteristics and running conditions to re-partition the load and to minimize simulation execution time. In the design of these approaches, many aspects are considered, such as monitoring metrics, resource heterogeneity, external background load, simulation computing load, simulation entities' interactions, and other simulation specific characteristics. However, neither of the solutions in the literature concerns the aspects that best suit HLA simulations running on large-scale distributed systems. Such dynamic load balancing approaches have been proposed for optimistic and conservative simulations, and some solutions have been designed specifically for HLA-based simulations.

For optimistic simulations, the proposed approaches attempt to decrease the number of simulation rollbacks by keeping simulation entities at the same simulation pace and by considering the simulations interactions. Following this approach, Glazer and Tropper [6] introduced the simulation advance time to identify imbalances and re-partition the simulation, and Jiang et al. [7] added weights to the same scheme in order to support heterogeneity. In the balancing scheme proposed by Burdorf and Marti [25], the least virtual time of each processor is employed to determine load imbalances. The scheme proposed by Schlagenhaft et. al. [26] groups interactive simulation entities and balances the load through the virtual time progress of a processor. Avril and Tropper [27] used communication dependencies and simulation throughput in their balancing system. The scheme devised by Carothers and Fujimoto [21] employs the processor advance time to balance distributed simulation loads, and Jiang et. al. [22] extends the scheme by adding communication metrics. The scheme proposed by Wilson and Shen [3] uses CPU consumption and pending load of simulation entities to perform communication-aware computation balancing. Deelman and Szymanski [28] used the number of weighted unprocessed events to determine load redistribution. In the balancing scheme designed by Choe and Tropper [29], a flow control is used to speed up the simulation and the balancing is performed based on memory consumption and the least LVT in a processor. Low [4] proposed a cost model that considers communication rate, computation load, and the number of lookaheads. In the scheme devised by Peschlow et. al. [2], re-distributions are determined by observing simulation advancements in alternate communication and computation balancing cycles. Even though the use of simulation pace to detect load balances indirectly considers external load and heterogeneity of resources, such approach is application-oriented and misleads the identification of imbalances in some scenarios.

For conservative distributed simulations, the approaches aim to minimize the simulation time by decreasing the communication rate and by maximizing the utilization of resources. Focusing on these objectives, the balancing schemes base their load re-distribution in CPU consumption, simulation entity's speed, and communication dependencies. Boukerche and Das [1] employ the CPU-length for arriving messages in their scheme to detect load imbalances, and communication is considered when determining load moves. In the balancing system proposed by Xiao et al. [30], a critical channel traversing is used to identify dependencies among simulation entities and minimize communication and computational delays. The balancing scheme proposed by Gan et. al. [23] considers lookaheads to distribute simulation entities and employs a thread pool to schedule simulation entities for execution.

Boukerche and Tropper [31] [24] employed entropy to detect load imbalances, considering computation load and communication rate. The scheme devised by Ajaltouni et. al. [8] works in alternate computation and communication balancing cycles by collecting CPU consumption and communication load of each simulation entity from JXTA grid-based [18] simulations. In these approaches, CPU load provides an application-independent metric to detect load imbalances, but the collected CPU regards only the simulation entities, neglecting external load and heterogeneity of resources.

For HLA-based simulations, the load balancing schemes consider the HLA simulation characteristics when performing federate migration in order to avoid causality inconsistencies. Lüthi and Grossmann [20] devised a resource sharing system using a high-latency federate migration. Cai et. al. [12] developed a Load Management System that performs high-latency federate migrations and uses Grids services to gather monitoring data. The resource management system proposed by Zajac et. al. [19] employs Grids services to perform monitoring and high-latency federate migration to move load. Tan and Lim [5] presented a load distribution scheme based on time advance calls and on a freeze-free federate migration. The focus of all these schemes regards federate migration, avoiding lost and disordered events and minimizing latency.

Even thought the aforementioned approaches present certain performance gains for distributed simulations, they either do not consider the heterogeneity of resources and background load fully or are not designed to support HLA simulations completely. Some approaches consider the load regarding only the simulation that is balanced, being inadequate for non-dedicate, heterogeneous resources. Other approaches use the simulation entity's processing speed to monitor load imbalances, which detects indirectly and partially the existence of imbalances in such distributed resources. Moreover, load migration, which is a critical procedure for balancing simulation's load, is not mentioned by several proposed solutions, or when migration is mentioned, it introduces considerable latencies. Thus, a dynamic load balancing scheme is proposed to overcome such drawbacks and improve the HLA simulations' execution performance.

III. Dynamic Load Balancing Scheme

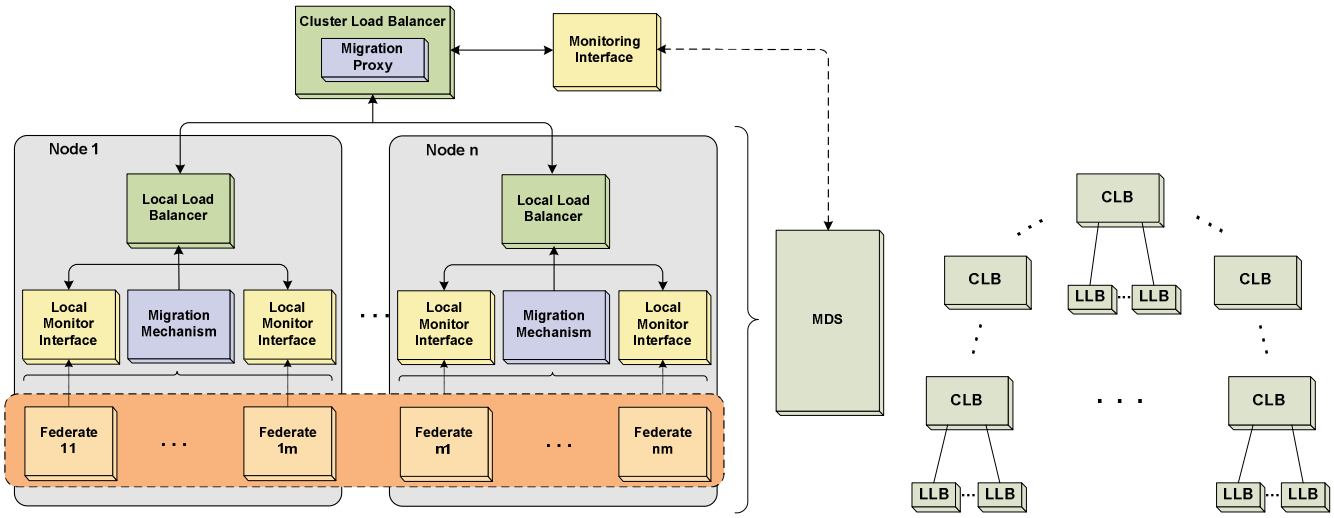
A new load balancing scheme is proposed in order to have a simulation-independent load management system that best fits with the characteristics of HLA simulations running on shared resources. Such scheme also aims to be aware of heterogeneity aspects of large-scale distributed systems to re-partition load accurately, to observe external background load when simulations run on non-dedicated resources, and to consider HLA simulation characteristics to avoid simulation casuality inconsistencies when transferring load. According to the design presented in the figure 1a, the dynamic load balancing system as a whole is responsible for communicating with the Grid system periodically, identifying load imbalances, re-allocating resources, and migrating load. The system is basically composed of a Cluster Load Balancer, a Monitoring Interface, Local Load Balancers, Local Monitoring Interfaces, and Migration Mechanisms. All these components are implemented in Java, aiming at a cross-platform design and an easy integration with an HLA RTI distribution.

The Cluster Load Balancer (CLB) is the main element of the load balancing system since it orchestrates all the required balancing tasks, performs the dynamic load balancing algorithm, and manages the load of one or more clusters of resources. The CLB is composed of Monitoring, Re-distribution, and Migration components. The Monitoring component accesses the Grids Index Service or another CLB through the Monitoring Interface, which is responsible for retrieving and filtering information. The Monitoring component also communicates directly to the Local Load Balancer or to another CLB to gather specific information from each federate. After data is collected, the Monitoring component seeks load discrepancies and triggers the Re-distribution component. The Re-distribution component is responsible for re-defining the load distribution according to the data sample analyzed by the Monitoring component and the load data sample is classified in underloaded, overloaded, and balanced sets. After the re-allocation is performed, migration calls are sent to their respective Local Load Balancer (LLB), which forwards a call to the proper Migration Mechanism.

As shown in the figure 1b, the balancing system is configured in a multi-layered hierarchical architecture, which facilitates the management of large-scale systems. In the design, a CLB can contain a list of LLBs, a list of CLBs to control, and a CLB father. Any CLB can control a list of LLBs that corresponds to a cluster of resources. In the lowest layer in the hierarchy, a CLB manages only LLBs and reports the collected data to its father CLB when performing global balancing. In the middle layers, a CLB also collects information from a list in order to report to its father CLB. In the top of the structure, the root CLB of the hierarchy is responsible for gathering all the collected data from its LLBs and the other CLBs, performing the global balancing procedure, and reporting other CLBs about the load moves.

The Local Load Balancer (LLB) 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 CLB. The information of the local resource is gathered through a direct access to the Local Monitor Interface, which monitors each federate individually. Also, when a LLB receives a migration command, it informs the local Migration Mechanism.

The Migration Mechanism processes the federate migration calls according to pre-defined migration steps, and a simulation agent is employed to support all the migration steps and to keep the simulations' causality consistency. The agent is responsible for managing all the required data transfers, such as a federate's code, its initialization files, and its running status. In order to perform such tasks, the agent is composed of a Migration Manager and a Communication Manager. The migration manager controls all the actions taken during or



(a) Architectural Components (b) Hierarchical Structure

Fig. 1: The Dynamic Load Balancing's General Architecture

for migration, launches the simulation agent at the remote resource, manages the information exchange, triggers the Communication Manager, and requests the simulation state of a federate. Moreover, as depicted in the figure 1a, the Migration Proxy is used as an intermediate element in the migration procedure to help the transfer of data of migrating federates. The proxy is employed when 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.

All these balancing components perform tasks that are grouped in distinct phases. Such classification generates a three-phase hierarchical scheme, which realizes a constant monitoring of resources to be aware of changes, a re-allocation algorithm to re-configure the distributed load dynamically, and an efficient federate migration technique to re-deploy the load in accordance with balancing needs.

A. Monitoring Phase

A reactive, adaptive, or predictive dynamic load balancing system, by default, requires a method to collect and analyze environmental information in order to promptly react to load changes, to precisely define imbalance scenarios, or to increase accuracy of predictions, so the first step in identifying load imbalances is to collect relevant information about the system to be balanced. In this step, the amount of information and frequency in which data is provided delineate the data gathering. Moreover, after data is retrieved from the observed resources, filtering and selection are employed to detect imbalances.

1) Data Collection: In order to balance load dynamically, the gathering of data is guided by a trade-off between precision and overhead. The frequency in which data is collected and transmitted, and the quantity of data which is provided to the balancing system can introduce undesired overhead to the distributed system. Thus, the balancing scheme adopts a two-level hierarchical monitoring design in order to minimize the monitoring overhead: cluster data collection and federate data collection.

a) Cluster Data Collection: In the cluster monitoring level, general information is collected from every resource in a distributed environment. Since this monitoring method is based on the resource load, it provides an application-independent dynamic load balancing solution. Through this information, load generated by applications that are external to the balanced simulation is detected in the distributed resources. Therefore, for a scalable data gathering solution, the managed resources' information is obtained by accessing the Grids Index Service, which provides monitoring information from the Monitoring and Discovery Service (MDS4) [9] that gathers information from Ganglia [15] and makes available through the GlueCE schema [10].

The average number of ready processes of a CPU queue is employed in the balancing scheme as the coarse-grained CPU load metric. This average is collected from each resource and provided by Ganglia every monitoring interval. The number of ready processes indicates the accumulated load of a processor, offering an application independent metric. However, some parameter is required to overcome the heterogeneity issues, so a capacity factor is adopted in the scheme to level the resources capabilities. In the balancing system, this factor is determined through benchmarks that delimit the processing capacity of each managed resource. Thus, according to the formula 1, the *relative load* of a resource is calculated based on its current load multiplied by the ratio between the common normalizing capacity and the resource's processing capacity. This normalizing capacity is adopted widely in the system and corresponds to the benchmark result of a specific resource.

$$rload_i = \frac{load_i * nCap}{Cap_i} \quad (1)$$

b) Local Data Collection: When imbalances are detected in a distributed system, a more precise and detailed data collection is required from overloaded resources to properly apply federate migration. This additional monitoring information is required to identify if federate migration can benefit the system's load balance. Such collected information regards the number of federates running on a resource and the load each federate produces. The monitoring information comprises the CPU utilization time a federate receives during a monitoring interval, and through such consumption, federates running in a resource can be ranked, providing means to the load balancer to apply transfer policies when migration is needed. The information is collected through a Java library called *ThreadMXBean*, which contains the method *getProcessCpuTime()* that collects a federate's CPU utilization time.

2) Filtering and Selection: After general data is gathered from system resources, filtering and selection mechanisms are employed to seek for determinant aspects in decision factors that will trigger proper balancing actions. Filtering is applied at two levels: a coarse-grained level and a fine-grained level. At a coarse-grained level, the information requested from the Grids Index Service contains general data ranging the many computing domains. As a result, the filtering mechanism selects all the information that matches CPU load. At a fine-grained level, filtering is applied to exclude resources that cannot submit load, i. e., overloaded resources that do not have any simulation federate running.

After all the irrelevant information is removed from the collected data, selection algorithms are applied to identify load imbalances properly. The selection algorithm adopted in the scheme comprehends to compare the load of each resource with overall average load, so load discrepancies can be identified. The discrepancies are the overloaded and underloaded resources that are classified as potential candidates to load transfers. In order to identify such discrepancies, the average load is obtained through the calculation of the arithmetic mean, and thresholds are incorporated in the mean to refine the selection, defining a range of values to differentiate imbalances in a data sample.

Based on a policy, the thresholds are required to identify when a resource is overloaded, balanced or underloaded by delimiting up and down boundaries which establish a tolerance for load variances. Such a tolerance is defined according to the function 2 and is delimited by three factors: mean, β, and α. Because the tolerance grows proportionally to the mean, the tolerance can hide considerable load dissimilarities, so a second factor is incorporated in the function, β. Ranging from 0 to 1, this second factor decelerates the tolerance grow and is calculated in the function 3. Through this function, a relative load oscillation limits the tolerance variation, so overloaded and underloaded resources are selected. Moreover, because the load imbalances are defined according to a criteria specified by certain policy, the deterrent factor α is required to narrow the balance interval. This factor is inserted in the function 2 and contains a value that ranges from 0 to 1, which represents the percentage of the average load that is considered balanced.

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

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

B. Re-distribution Phase

Load re-distribution is the critical part of a dynamic load balancing scheme because re-distribution is directly related to the reactiveness and efficiency of an applied balancing scheme. Re-allocating resources for distributed applications can involve complex processes that introduce considerable overhead to the distributed system or take so long to answer that the calculated re-distribution does not reflect the real workload status of the system. For effectively determining load transfers, computation and communication aspects are considered, but in the proposed load balancing scheme at this step, only computation load is evaluated to re-distribute federates on a distributed system.

As described in the algorithms 1 and 2, the analysis consists of comparing the most recent load of a pair of resources, in which one of them is an underloaded or an overloaded resource. Based on each pair analysis, a federate is chosen to be migrated from a source resource with higher load to a destination resource with lower load, and an overloaded or an underloaded resource is always present in the migration move. Moreover, federate migration consumes resources and adds delays to the simulation, so the load balancing system does not react as fast as the application load changes in order to avoid precipitated, unnecessary migration moves produced to react to abrupt, short-term computation load changes.

The re-distribution is composed of two steps. The first step consists of classifying the resources according to their load in three categories – balanced, underloaded, and overloaded. A set is created for each category, and the resources of each set are organized according to their load samples in ascendant order. The second step follows three premises: to establish migration moves from the most loaded resource to the least loaded resource, to accomplish only one migration

Algorithm 1 Pair-Match Re-distribution Algorithm

Require: $resources\_data$
  $classify(resources\_data)$
  $order\_ascendant(underloaded, balanced, overloaded)$
  for all src_rsc IN $overloaded$ **do**
    $dst\_rsc \Leftarrow get\_next(underloaded, balanced)$
    $migration\_moves \Leftarrow evaluate(src\_rsc, dst\_rsc)$
  end for
  for all dst_rsc IN $underloaded$ **do**
    $src\_rsc \Leftarrow get\_next(overloaded, balanced)$
    $migration\_moves \Leftarrow evaluate(src\_rsc, dst\_rsc)$
  end for

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 $number\_fed(src\_rsc) = 1$ & $src\_rsc > (min * \phi)$ **then**
      $create\_migration\_move(src\_rsc, dst\_rsc)$
    end if
  else if $(dst\_rsc - src\_rsc) > (min * \delta)$ **then**
    if $number\_fed(src\_rsc) > 1$ **then**
      $create\_migration\_move(src\_rsc, dst\_rsc)$
    else if $number\_fed(src\_rsc) = 1$ & $(dst\_rsc - src\_rsc) > (min * \phi)$ **then**
      $create\_migration\_move(src\_rsc, dst\_rsc)$
    end if
  end if

move for each pair of resources, and to migrate the federate producing the least load in determined resource. This step comprises selecting the migration pairs regarding the first premise and analyzing them in order to generate effective federate migration moves. According to this method, if all the resources are stored in a queue and organized in ascendant order, the migration pairs are assembled with both extremities in the queue. This assembling process keeps running until both the underloaded and overloaded elements are treated or when there are not two resources that have enough load difference to justify a migration.

According to the algorithm 2, in each pair match, the load of each resource is analyzed and compared with its respective resource. A migration move is generated and triggered through analysis conditions regarding the load of each resource and number of running federates. The first condition observes the destination resource's load. If this load is less than a minimum, the resource is a potential candidate to receive a federate. Then, a migration move is created if the source resource has more than 1 federate, or if the resource has just one federate and load more than the threshold $min * \phi$, which corresponds to the usage of more than one CPU. The second condition observes the relation between the destination resource's load and the source resource's load. If the relation is larger than the threshold $min * \delta$, which comprises the load of less than one CPU, and the source resource has more than one federate running, a migration move is created, and if the source has only one federate and the relation is under the threshold $min * \phi$, a migration move is created.

C. Migration Phase

This phase is responsible for efficiently implementing the calculated load re-distribution, and its efficiency dictates the number of times migration is called and the cumulative delay inserted in the overall simulation run-time. The migration move is accomplished through federate migration, which comprises transferring the federate code and its initialization files to the destination node, suspending the federate and saving its running state, transferring the state, and restoring the federate execution. Even though federate migration process comprises a few simple steps, it introduces a considerable overhead to the simulation due to data transfer latencies. Thus, the federate migration technique adopted in the balancing system is similar to the approaches devised by Boukerche and De Grande [11] and Li et. al. [32], which is freeze-free, requires minimal external tools, and avoids unnecessary communication and computing to be accomplished.

The federate migration approach accomplishes data transfers in two steps. In the first step, the system transmits only the configuration files and any other piece of data that is required to initialize the migrating federate. In this step, the federate is also launched remotely in order to be ready to receive the rest of data regarding its running state and ongoing messages. Grid services are employed to send all these static files and the code of the migrating federate. The Web Service Grid Resource Allocation and Management (WS GRAM) [9] is used to submit remotely the migrating federate. Before the federate is launched remotely, all its required data is transmitted through a staging process using GridFTP [9]. All the third-party mechanisms are used to transfer and launch the federate reliably, but even though they introduce a large delay in the simulation, the delay is negligible because the migrating federate does not stop its execution while these actions are performed. In the second step, when the federate launched at the remote resource is ready to continue the migration process, the communication channel with the RTI is set to the migrating federate at its new location. After that, through a call to an interfacing method named *federateSave*, which retrieves the federate's running status and the Local Runtime Component's (LRC) state, the migration mechanism stops the local federate's execution, saves its state and messages, and send them directly to the remote federate. The messages comprehends the events that are received during the migration procedure and need to be processed properly in order to avoid simulation inconsistencies, so the incoming messages are stored in a queue, which is transferred to the destination resource. Finally, after receiving the messages, the remote federate restores completely its execution state, and continues its execution.

D. Load Balancing Algorithm

As described in the algorithm 3, the complete dynamic load balancing procedure merges the three balancing phases previously presented: resource monitoring, load re-distribution, and federate migration. These phases are interrelated in a sequential dependency, being triggered by the previous phase in the balancing cycle. In this case, firstly, the monitoring starts from a time trigger, which initiates a new load balancing cycle. The balancing cycle occurs in pre-defined periodical Δ intervals. This value directly dictates the overhead ratio the balancing scheme generates and the balancing reactiveness, and it is limited to the frequency in which the MDS updates its monitoring data base, 20 seconds.

Algorithm 3 Global Dynamic Load Balancing Algorithm

```
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 CLB_son IN list_CLB do
        ext_loads ⇐ request_loads()
      end for
      merge(ext_loads, current_loads)
      for all CLB_son IN list_CLB do
        ext_spec_loads ⇐ request_spec_loads()
      end for
      merge(ext_spec_loads, spec_loads)
      mng_loads ⇐ filter(current_loads, spec_loads)
      mean, bds ⇐ calculate_mean_bds(mng_loads)
      over, under ⇐ select(mng_loads, mean, bds)
      mig_moves ⇐ redistribute(mng_loads)
    else
      for all CLB_son IN list_CLB do
        ext_loads ⇐ request_loads()
      end for
      merge(ext_loads, current_loads)
      send_father(current_loads)
      for all CLB_son IN list_CLB do
        ext_spec_loads ⇐ request_spec_loads()
      end for
      merge(ext_spec_loads, spec_loads)
      send_father(spec_loads)
      mig_moves ⇐ retrieve_mig_moves()
    end if
  else
    overload_cand ⇐ select_overload(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)
    mig_moves ⇐ redistribute(mng_loads)
  end if
  send_migration_moves(mig_moves)
  balancing_globally ⇐ choose_scope()
  wait( Δ )
end loop
```

According to the algorithm 3, the balancing occurs locally or globally. In the monitoring phase, the system collects information in cluster and local scopes. The cluster data gathering is triggered as soon as the load balancing cycle startss. After collected, the cluster data is filtered, and a local gathering is performed based on the median calculated with the cluster data. With the local gathering, the monitoring data can be filtered to focus on the overloaded resources that can be managed through federate migration. Additionally, if a global balancing is determined, a CLB also gathers all the monitoring information from its sub-CLBs and sends them to its father CLB.

If any load imbalance is detected in the monitoring phase, the re-distribution algorithm is triggered to identify the possible migration moves and consequently to re-allocate resources for the running federates. When the balancing system is managing the load globally, the root CLB detects imbalances and re-distributes the loads accordingly, and send the migration moves to their respective CLB in order to be finally forwarded to their LLBs. When federate migration moves are determined, only one move regarding a node is accomplished per balancing cycle in order to avoid abrupt changes on the distributed load configuration. Thus, the balancing process works gradually, attempting to correctly distributed the load by dealing with a resource and its respective migrating federate and by evaluating the load change results. Moreover, if any migration move is generated, the migration mechanism responsible for the resource respective to the migration move is triggered to realize the needed load transfer. Triggered locally in the chosen source resource, the migration acts independently from a load balancing cycle, and when performed, the balancing system ignores the involved resources in the next balancing cycle to provide enough time to a resource to stabilize its processing according to the load changes.

IV. Experiments and Results

Experiments have been accomplished to evaluate the performance benefits that the proposed dynamic load balancing approach provides to HLA-based simulations running in large-scale, heterogeneous, non-dedicated distributed environments. Such experiments have exposed how the balancing mechanism has detected and reacted to the load imbalances in the simulations. The system's reaction has been evaluated by noticing the effectiveness of resource re-allocation according to load imbalances using federate migration.

A. Experimental Scenario

All experiments have been conducted on a system comprising an Dell cluster composed of 24 nodes, an IBM cluster composed of 32 nodes, and a fast-ethernet connection between them. Each node in the Dell cluster had a QuadiCore 2.40GHz Intel(R) Xeon(R) CPU and 8 gigabytes of DIMM DDR RAM memory, and all nodes were inter-connected through a Myrinet optical network that allowed data transmition up to 2 gigabits per second. Every node in the IBM cluster consisted of a Core 2 Duo 3.4 GHz Intel(R) Xeon(R) CPU and 2 gigabytes of DIMM DDR RAM, and a gigabit ethernet network connected the cluster's nodes. Moreover, Linux operating systems were installed in both clusters, and the experiments were supported by the HLA platform with an RTI version 1.3. The Globus version 4.2.1 was installed in the system in order to provide Grids services to the distributed applications.

In order to perform the experiments, simulation federates were deployed on the 56 nodes, considering that one node was totally dedicated to run the HLA RTI executive. Configure

TABLE I: Dynamic Balancing System Configuration

α	min	δ	ϕ
0.15	150	8	6

with the values shown in the table I, the balancing system's LLBs were deployed evenly on the nodes, and a CLB was placed in one node of each cluster where one was the root CLB. Moreover, as a simulation scenario, federates simulated training operations of two teams of interactive tanks in a two-dimension routing space in time-stepped simulations. In order to accomplish tank movements, each federate had to perform computing intensive calculations and to publish a tank's position and subscribe to a interest space. In the scenario, the HLA simulation was composed of a range of 1 to 1000 federates that performed communication and computation tasks in 100 time steps, and each federate managed just one tank to focus on computation load and on parallelism.

B. Analyses

In order to evaluate the proposed system's efficiency, the experiments were accomplished in two test case groups over heterogeneous, non-dedicated sets of resources, applying a increasing large load to the distributed system. In the first test case group, the effectiveness of the dynamic load balancing system was observed as distributed load imbalances occur. In the second test case group, to analyze the detection of external background load, a external load is added in the system and the balancing reaction is observed.

1) Reactiveness to load imbalances: In this test case, all the distributed simulations were deployed based on an initial static partitioning that evenly placed the federates on the shared resources. However, due to the heterogeneity characteristics, the differences in the resources' computing capacities resulted in an slight uneven distribution of load, decreasing the simulations' performance. In order to evaluate the balancing system's reaction to load imbalances, the balanced simulation's performance was compared with a baseline. In this case of experiments, the baseline comprehended the run of the experimental scenario without any load balancing and any external background load, having a configuration of federates that ranged from 1 to 1000.

According to the graph in the figure 2, the proposed dynamic balancing scheme improved the performance of HLA-based simulations on large-scale distributed systems in most of the experiments. When the distributed load was under 100 federates, the balancing scheme's improvement is unnoticeable or nonexistent because the simulations did not require any load balancing. In this case, the balancing just caused a tiny overhead for the distributed system, consuming computing from the resource where the balancing system was deployed. A noticeable improvement was detected with experiments over 100 federates because considerable load imbalances occurred during the simulation. Even though with a even partitioning of federates, the heterogeneity of resources caused an imbalanced

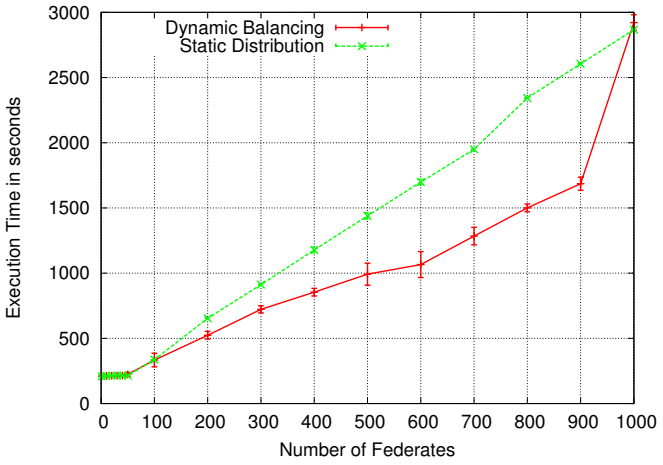


Fig. 2: Dynamic Balancing Scheme versus a Static Distribution for an Increasing Number of Federates

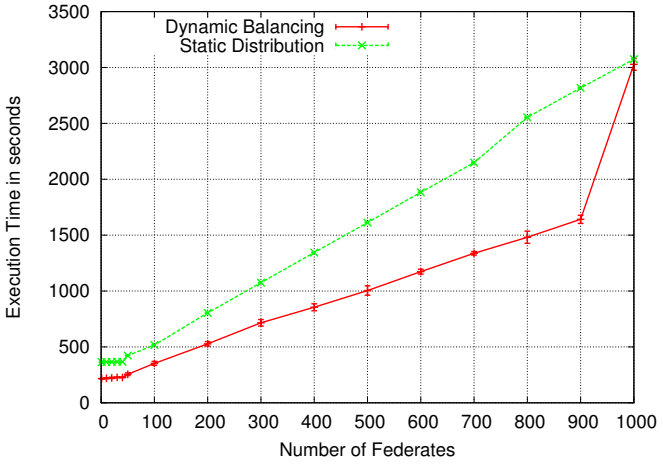


Fig. 3: Capacity of the Balancing Scheme in Detecting External Load for an Increasing Number of Federates

division of load. Finally, a high increase in execution time in the balanced system's curve is observed between experiments from 900 to 1000 federates. This increase evidences that the distributed system is reaching a saturation point in which the balancing system cannot improve the simulation performance since all shared resources are becoming totally overloaded.

2) Detection of External Background Load: In order to measure the efficiency of the load balancing system in detecting and reacting to an external background load, an external application was placed in one of the shared resources. The application was build to introduce a controlled amount of load, so it produced a considerable overhead, slowing down the federates in certain resource. In the test case, the federates were deployed evenly on the clusters' nodes, and external application was placed on a node of the IBM cluster.

As shown in the figure 3, the curves are similar to those in the figure 2 except that introdution of an external load caused an addition of execution time for low load experiments, from

1 to 100 federates. Over 100 federates, the load produced by the federates in the system surpass the external application's overhead. Thus, the load balancing system presented a performance improvement in all the experiments since it detected the external load, triggering re-distribution of load.

V. Conclusions and Future Work

In this paper, a hierarchical dynamic load balancing scheme is proposed. The scheme is designed and implemented to support the re-distribution of load for HLA-based simulations running on large-scale distributed systems. Accessing Grid services and with a tree structure, the balancing system splits the balancing algorithm in three sequential phases that detect, re-distribute, and migrate load, and successfully detects external background loads and load differences in heterogeneous resources. The system makes use of Grids services to retrieve monitoring information from shared resources and to aid federate migration. The system's hierarchical architecture provides a scalable solution for the management of resources and load of large-scale distributed systems.

In the devised balancing scheme, the communication overhead is not managed. As a future work, the simulation intercommunication will be considered in the scheme, so minimizing the communication can lead to performance improvement. Therefore, the interaction of federates will be incorporated in the load balancing algorithm since grouping federates according to their interaction group decreases the consumption of network resources, decreasing delays and freeing computing resources.

References

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

[2] Peschlow, P., Honecker, H., Martini, P.: A Flexible Dynamic Partitioning Algorithm for Optimistic Distributed Simulation. In Proceedings of the 21st Workshop on Parallel and Distributed Simulation (PADS07), pages 219–228. IEEE Computer Society, 2007.

[3] Wilson, L. F., Shen, W.: Experiments in load migration and dynamic load balancing in speedes. In Proceedings of the 1998 Winter Simulation Conference, pages 483490. IEEE Computer Society, 1998.

[4] Low, M. Y. H.: Dynamic load-balancing for BSP time warp. In Proceedings of the 35th Annual Simulation Symposium (SS02), pages 267–274. IEEE Computer Society, 2002.

[5] Tan, G., Lim, K. C.: Load Distribution Services in HLA. In Proceedings of 8th IEEE Distributed Simulation and Real-time Applications, Budapest, Hungary, October 2004, pp 133–141.

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

[7] Jiang, M. R., Shieh, S. P., Liu, C. L.: Dynamic load balancing in parallel simulation using time warp mechanism. In Proceedings of the 1994 International Conference on Parallel and Distributed Systems, pages 222–229. IEEE Computer Society, 1994.

[8] Ajaltouni, E. E., Boukerche, A., Zhang, M.: An Efficient Dynamic Load Balancing Scheme for Distributed Simulations on a Grid Infrastructure. I Proceedings of the 12th 2008 IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, pages 61–68. IEEE Computer Society, 2008.

[9] Globus, University of Chicago, 7 Feb. 2008. [Online]. Available: http://www.globus.org/

[10] Ws mds: cluster monitoring information and the glue resource property, The Globus Toolkit, 7 Feb. 2008. [Online]. Available: http://www.globus.org/toolkit/docs/4.0/info/key/gluerp.html

[11] Boukerche, A., De Grande, R. E.: Optimized Federate Migration for Large-Scale HLA-Based Simulations. In Proceedings of the 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, pages 227–235. IEEE Computer Society, 2008.

[12] Cai, W., Turner, S. J., Zhao, H.: A load management system for running hla-based distributed simulations over the grid. In Proceedings of the Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications, pages 7–14, Washington, DC, USA, 2002.

[13] 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.

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

[15] Massie, M. L., Chun, B. N., Culler, D. E.: The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing, 30(7): 817–840, 2004.

[16] Foster, I. Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 15(3): 200–222, 2001.

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

[18] Boukerche, A., Zhang, M.: Towards Peer-to-Peer Based Distributed Simulations on a Grid Infrastructure. In Proceedings of the 41st Annual Simulation Symposium, pages 212–219. IEEE Computer Society, 2008.

[19] Zajac, K., Bubak, M., Malawski, M., Sloot, P.: 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, pages 4–11. IEEE Computer Society, 2003.

[20] Luthi, J., Grossmann, S.: The resource sharing system: dynamic federate mapping for HLA-baseddistributed simulation. In Proceedings of the 15th Workshop on Parallel and Distributed Simulation, pages 91–98. IEEE Computer Society, 2001.

[21] Carothers, C.D., Fujimoto, R.M.: Background execution of time warp programs. In Proceedings of the tenth Workshop on Parallel and Distributed Simulation, pages 12–19. IEEE Computer Society, 1996.

[22] Jiang, J., Anane, R., Theodoropoulos, G.: Load balancing in distributed simulations on the Grid. In proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pages 3232–3238. IEEE Computer Society, 2004.

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

[24] Boukerche, A.: An adaptive partitioning algorithm for conservative parallel simulation. In Proc. of the 15th International Parallel and Distributed Processing Symposium, pages 133–138. IEEE Computer Society, 2001.

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

[26] Schlagenhaft, R., Ruhwandl, M., Sporrer, C., Bauer, H.: Dynamic load balancing of a multi-cluster simulator on a network of workstations. In proceedings of the ninth workshop on Parallel and distributed simulation, pages 175–180. IEEE Computer Society, 1995.

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

[28] Deelman, E., Szymanski, B. K.: Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. In proceedings of the twelfth workshop on Parallel and distributed simulation, pages 46–53. IEEE Computer Society, 1998.

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

[30] Xiao, Z., Unger, B., Simmonds, R., Cleary, J.: Scheduling critical channels in conservative parallel discrete event simulation. In proceedings of the thirteenth workshop on Parallel and distributed simulation, pages 20–28. IEEE Computer Society, 1999.

[31] Boukerche, A., Tropper, C.: A static partitioning and mapping algorithm for conservative parallel simulations. Proc. of the eighth workshop on Parallel and dist. simulation, p.164–172. IEEE Computer Society, 1994.

[32] Z. Li, W. Cai, S. J. Turner, and K. Pan, Federate migration in a service oriented hla rti, in Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications. IEEE Computer Society, 2007, pp. 113–121.