Measuring and Analyzing Migration Delay for the Computational Load Balancing of Distributed Virtual Simulations

Robson E. De Grande, Mohamed Almulla and Azzedine Boukerche, *Fellow, IEEE*

***Abstract*—**
Distributed virtual simulations can always undergo load imbalances during run-time due to their dependency on underlying shared resources. Such imbalances are commonly generated by external background load, improper deployment of simulation elements, and dynamic oscillations of simulation load. High Level Architecture (HLA) was devised as a simulation framework that simplifies the design and management of distributed simulations. Even though the framework presents services for the coordination of simulations, it does not provide any tool for identifying and reacting to load imbalances. Due to the importance of balancing simulation load, many schemes have been developed, aiming to reduce simulation time through analysis of specific metrics. These schemes are limited to issues from specific simulation applications, or they disregard large-scale environmental characteristics. In order to overcome the drawbacks of previous balancing approaches, a distributed balancing scheme has been designed. Nevertheless, this scheme, as well as the others, is not concerned with migration latencies when redistributing simulation load. Migration delays are directly involved with balancing responsiveness and efficiency, and they can cause simulation performance loss instead of performance improvement if they are not considered in the redistribution analysis. Thus, a migration-aware balancing scheme is proposed to include migration latency analysis in its load redistribution algorithm. Extensions are also proposed to improve the analysis of migration delays by enabling necessary costly migrations in iterative analysis. Experiments have been conducted to evaluate the proposed migration-aware balancing schemes by comparing performance when costly migrations are present in simulations.

Index Terms—**Distributed Simulations, HLA, Performance, Load Analysis**

I. Introduction

Distributed virtual simulations have become increasingly important and have gained a great deal of attention with the growing interest in simulating complex systems or developing large-scale virtual environments. Such simulations, based on HLA framework for the coordination of simulation interactions in large-scale systems, can experience performance loss, as any distributed system. This performance loss is inherent due to the distribution and the resources that run the simulation parts, and it is mostly caused by improper placement of simulation elements on resources and simulation load changes (oscillations) that might occur during run-time. In both cases, load imbalances in distributed virtual simulations are directly related to performance loss since they produce inappropriate consumption of shared resources. Oscillations in the dynamic load of simulations, heterogeneity of resources, and external background processing that might also have dynamic characteristics can produce load imbalances in such large-scale environments. Due to the need to prevent such imbalances, static and dynamic balancing schemes have been devised. Even though static deployment and partitioning of simulation load can prevent imbalances caused by heterogeneity of resources and simulation entities, this type of balancing is incapable of reacting to dynamic load changes. Therefore, in order to avoid such dynamic imbalances, dynamic balancing of simulation load is necessary.

High Level Architecture (HLA) [1] was devised to facilitate the design and management of distributed simulations through a framework. This framework supports large-scale simulations and introduces a design standard that aims at re-usability of simulation components and interoperability among simulation entities. Basically, the framework consists in a set of rules that delimit the boundaries of designing HLA simulations, interface definitions that delineate the method by which simulation entities interact in simulations, object templates that describe the data exchanged in simulations, and management services that coordinate virtual simulation progress. Simulation entities, called federates, access such services to interact with other federates and retrieve information dynamically, and the services are responsible for controlling the federates' actions to keep the entire simulation consistent. The management services are provided by a Run-time Infrastructure (RTI), which runs together with federates, composing a virtual simulation, called federation. Since HLA framework only coordinates simulations to prevent inconsistencies, it cannot detect load imbalances or any issue regarding the underlying resources that are used to run the simulations. As a result, a balancing scheme is required for such simulations to avoid performance loss.

In order to determine the placement of simulation elements and to redistribute simulation load on shared resources, many balancing schemes have been devised. In summary, the designed balancing systems consider computational and communication load characteristics to detect and redistribute simulation load. Focusing on the balancing of computational load, the majority of the previous

R. E. De Grande is with the School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa. E-mail: rdgrande@site.uottawa.ca

M. Almulla is with the Department of Computer Science, King Saud University, Riyadh, Saudi Arabia.

A. Boukerche is with the School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa. E-mail: boukerch@site.uottawa.ca

schemes do not fully cover the aspects of balancing large-scale distributed simulations. Therefore, a distributed balancing scheme [2] has been devised to provide a load redistribution algorithm to consider such aspects and improve balancing effectiveness. Nevertheless, this decentralized balancing approach, and other previous balancing schemes, disregards the migration latencies when reorganizing load distribution, so delays can be introduced in simulations by a balancing system through considerable migration latencies, jeopardizing simulation performance instead of improving it.

Therefore, a distributed, dynamic balancing scheme is proposed to introduce awareness of migration latencies in its load redistribution algorithms, as designed in [3]. The design of this proposed balancing system is similar to the distributed balancing approach [2]; its scheme is divided into monitoring, redistribution, and migration phases. Working in balancing cycles, the proposed system periodically monitors simulations to enable responsiveness to imbalances. Moreover, measurements of metrics related to federate migrations are used in the proposed balancing system to analyze and evaluate modifications on the load distribution which consider migration costs. This migration-aware balancing scheme lacks efficiency; thus, extensions to the migration-aware balancing scheme are introduced to increase the balancing efficiency by improving analysis techniques applied on gathered migration measurements.

The remainder of this paper is organized as follows. In Section 2, the related work is delineated and challenging issues are presented. In Section 3, the proposed balancing system is described by showing its architecture and the functioning of its redistribution algorithms. In Section 4, the extensions for the migration-aware balancing scheme are detailed. In Section 5, the experimental scenario is defined, and the obtained results are discussed. In Section 6, the conclusion is presented, and future work directions are delineated.

II. Related Work

Distributed virtual simulations, like any other distributed application, rely completely on the underlying resources to execute properly or maintain a reasonable performance, returning the processing result consistently and on time; consequently, load balancing becomes essential to such simulations. Due to its importance, several balancing schemes have been devised, aiming at the increase of simulation performance, which results in a decrease of execution time. These balancing systems attempt to improve the utilization of available resources and to detect more efficient methods of consuming them for the benefit of processing performance. For distributed simulations, the balancing schemes generally observe computational load and simulation inter-dependency aspects to redistribute simulation elements. While simulation inter-dependencies present an indirect result on performance through application-dependent delays, computational load aspects directly influence simulation processing performance through the consumption of resources' computing power. Even though the analysis of such aspects can be performed statically in some of the already designed balancing approaches, most of the approaches present dynamic balancing techniques to enable the detection and response to run-time load imbalances.

For the observation of internal dependencies within simulations, balancing systems commonly use look-ahead and communication rate as metrics to prevent or decrease delay effects on distributed simulation performance. Look-ahead is a metric totally involved with simulation characteristics and is employed in balancing systems to indirectly indicate the dependencies among simulation entities, evidencing simulation interaction latencies that might increase simulation execution time [4] [5]. Communication rates and latencies in distributed simulations enable balancing systems to directly identify which dependency causes delays in a simulation and to quantify the amount of delay that is introduced in the simulation system. Such communication delays between interacting parts are generated basically by the network distance and overhead on the communication resources. In order to detect such delays and to reorganize the simulation distribution, static analysis can be performed through critical communication path analysis [6], [7], or dynamic analysis can be conducted through periodic measurements and the balancing of simulation interactions [8], [5], [9], [10], [11], [12], [13], [14]. Some of these balancing schemes also consider proximity of resources and the network topology in their schemes [15], [16], [17], [18]. Even though the balancing of simulation inter-dependencies enables improvement in performance, it does not solve the performance issues caused by imbalanced load on shared resources, which is the focus of the proposed balancing scheme in the next section.

The computational load aspects employed in the previously devised balancing schemes can be classified through simulation-centred and resource-centred approaches. The balancing schemes using the simulation-centred approach evaluate the load distribution through simulation characteristics. Load imbalances generated on the resources are indirectly reflected on the simulation execution pace, and based on simulation aspects, the balancing of load allows the improvement of simulation execution by increasing the relative execution speed of each simulation entity [19], [20], [21], [22]. On the other hand, balancing systems based on the resource-centred approach measure and analyze the load aspects that directly influence execution performance. Basically and with a more general purpose solution, this type of balancing scheme observes shared resources' load distribution, detects the load discrepancies among resources that might be decreasing simulation execution pace, and attempts to evenly maximize consumption of resources' computational capacity [23], [24], [25], [26], [27], [28], [29], [12], [13]. All these listed computational load balancing approaches present limitations that impede them to cover critical aspects for distributed virtual simulations, such as resource heterogeneity, causality inconsistencies, and external background load. Observing such issues, parameters are used in a balancing scheme specifically designed for optimistic simulations [30] to identify the existence of external load and the differences of computational capacities between resources. Even with such a load management system, a balancing design is needed for the context of HLA-based simulations that also directly observes the influence of load on simulation elements. In recognition of such characteristics' importance, two schemes [31], [2] have been proposed to consider such aspects in the balancing scheme.

However, the two last cited computational load balancing approaches, as well as the others, do not evaluate migration latencies when rearranging simulation load distribution. Considering such latencies in balancing schemes is crucial since migration delays can grow as the execution state of simulation entities (federates) grow with the complexity of simulation designs. When reorganizing load through migration, these migration delays might generate more overhead on systems than improvements on execution time, jeopardizing simulation performance rather than improving it. As a result, a balancing scheme, and its extension, that measures and analyzes migration latencies while redefining simulation load distribution is proposed.

III. Proposed Migration-Aware Balancing System

Load migration latency is completely involved with redistribution performance because it is directly related to balancing responsiveness, which dictates the frequency in which simulation entities of virtual simulations can be moved. For load redistribution, only essential migrations are required to be performed. Thus, minimizing the amount of migrations or allowing only essential modifications on load distribution enables better performance gain. Precipitated migrations generate imbalances by transferring load to resources that become overloaded. Besides generating simulation overhead with new imbalances, precipitated load transfers produce unnecessary migration latencies, which are included in the final simulation time. Even if precipitated migrations are not produced, migrations that are vital for evenly distributing the virtual simulation load may also generate overhead, and this overhead needs to be considered in the balancing algorithms to evidence the advantage of load transfers for the benefit of performance. Since the performance gain provided by some load moves might be less than the overhead caused by their migrations, such migration moves need to be avoided in order to maintain execution performance of virtual simulations.

The majority of previously devised balancing schemes only considers computational and communication load imbalances and availability of resources in their load balance analysis, without measuring the inherent disadvantages of moving simulation load between resources. Even though some balancing systems observe migration latency in their schemes, they do not perform direct analysis of migration latency influence on simulation performance. As an approach in such balancing schemes, their redistribution thresholds are defined with static values to introduce performance loss caused by migrations in their analysis of simulation load distribution. The static values that delimit these balancing systems are obtained through extensive observation of experiments, which might lead the balancing system to react to specific load patterns and not to general load distributions.

The main objective of the proposed balancing scheme is to introduce the measurement and analysis of migration latency in the load redistribution algorithm of a distributed balancing system [2]. Through such measurements and analysis, costs of migrating load among resources are incorporated into the decision-making of the balancing algorithms to introduce awareness of migration latency by calculating estimations of migration delays, as defined in [3]. Since real migration latency values can only be obtained after migrations are performed, estimations are needed and are based on the past simulation load migrations to enable cost analysis before migration calls are issued to modify load distribution. In order to make possible awareness of migration latencies, migration metrics are measured and monitored so they may be evaluated together with load measurements of resources and simulations. As a result, the balancing algorithms' phases are modified to accommodate migration-aware analysis in the scheme.

A. Architecture

Because migration awareness is introduced on an already defined balancing algorithm, the proposed scheme is defined similarly to the architectures described in [2], [31]. As depicted in Figure 1, the Cluster Load Balancer (CLB) is the main element in the balancing system whereas it coordinates all the other balancing elements in a domain and conducts the main tasks of the balancing procedure. A CLB collects measured load status data in two scopes: simulation and resource. For the gathering of information concerning resources' load status, a Monitoring Interface is accessed. The Monitoring Interface obtains the data by requesting Monitoring Information Services, which can be represented by a third-party monitoring tool that retrieves load status data from a set of resources; in this implementation of the balancing system, Grid Services [32] are used to provide such data, which consists in the processor queue length of each resource. For retrieving information related to federates' load of virtual simulations, Local Monitoring Interfaces are accessed through Local Load Balancers (LLB). Such interfaces gather and aggregate load metrics that represent the CPU consumption of federates for detecting imbalances and migration-related metrics for the awareness of migration latencies.

Since the gathered data contains information from all resources that compose the distributed environment and from federates that are placed on the distributed system, filtering is applied to remove unnecessary information that might lead the balancing scheme load analysis to generate improper modifications on load distribution. The filtering is primarily concerned with the elimination of load information related to resources that cannot be managed or to

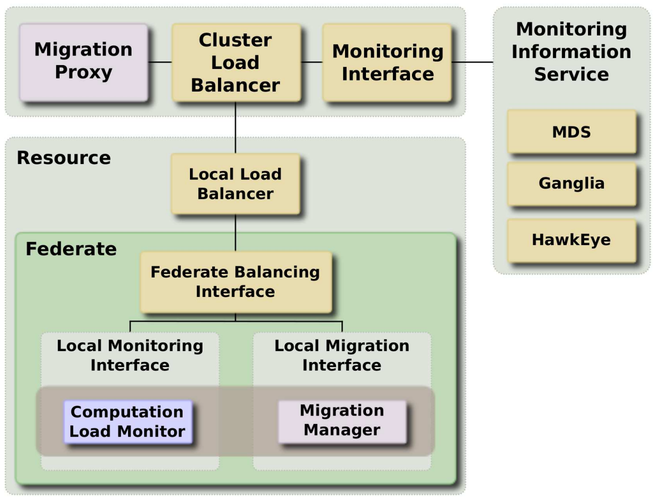


Fig. 1. General Architecture of the Dynamic Load Balancing System

overloaded resources that do not contain any simulation elements running on them.

A LLB acts as an interface between the balancing system and virtual simulations; the interface enables the management of federates in each resource. Placed on each resource, a LLB obtains the information about each federate's load consumption, aggregates this information, and sends it to the respective CLB, which has initially placed a load status request. The data gathering performed by a LLB is achieved by accessing each Computation Load Monitor through a Local Monitoring Interface. The Computation Load Monitor obtains the CPU consumption of a federate by accessing a Java monitoring library interface, *ThreadMXBean*, every time it is triggered. Upon request, this Computation Load Monitor provides the amount of time a federate has used the CPU of a resource. The Local Monitoring Interface gathers this federate's load data during a balancing time interval and sends it upon request. Together with this load data, migration measurements are also collected and sent to a CLB for analysis. The LLB also forwards migration calls from the CLB to the respective federate. Such migration calls are transmitted to a Migration Manager (MM), which is instantiated for each federate.

A MM coordinates all the steps for the federate migration procedure, so the transfer of federates between resources can be conducted consistently and without loosing data. In order to perform this migration procedure efficiently and effectively, a two-phase federate migration protocol [33], [34] is employed in the scheme. This federate migration technique basically divides the reallocation of a federate between resources through transfers of static information and dynamic data. For the transfer of static initialization information, a third-party tool, Grid Services, is employed in the system for transferring data reliably. For the transfer of dynamic execution state information, a peer-to-peer technique is used between MMs.

Accessed by the Monitoring Interface and the MM, Grid Services provide general information about the load resource status through third-party monitoring tools while also providing static federate information through reliable data transfer mechanisms. The Monitoring Information Service and the Reliable File Transfer (RFT) are mechanisms available in resource management systems that facilitate the management of distributed application and resources. As devised in Grid computing [35], a Grid is a resource management system that controls the access of individuals and institutions to distributed resources, coordinating the execution and delegation of distributed applications. In aspects of implementation of such devised services, Globus Toolkit [36] is the *de facto* middleware standard that enables the resource sharing system; it is based on Open Grid Services Architecture (OGSA) [32], which incorporates service oriented architecture on Grid Services. The OGSA consists in monitoring resources and applications, and scheduling and allocating shared resources according to application requirements [37]. For the proposed balancing system, these services are accessed to retrieve resources' load status through the Ganglia Monitoring System and to transfer federate's static data between resources through GridFTP in WS-GRAM service.

The entire proposed balancing system is hierarchically structured between CLBs and LLBs, and it presents a distributed structure for the relations among CLBs when performing inter-domain load balancing. Because LLBs are responsible for federates placed on resources, they are grouped by a CLB; the CLB manages the load of a domain, which comprises a cluster of resources. This group, hierarchically delimited by a CLB, corresponds to a branch in the whole balancing system structure. These branches are not required to be organized in a hierarchy whereas a set of CLBs can be inter-connected in any structure. However, since CLBs might be placed according to the availability of distributed resources, the balancing system might follow the topological structure of resources. A CLB can also be connected to other N CLBs; a larger number of connections enables fast responsiveness to imbalances since distances between CLBs are shortened, and load changes are more effectively transferred to the rest of the distributed system. The load transfers between CLBs are directed by the redistribution algorithms in the balancing scheme according to predefined relations between CLBs [2].

B. *Proposed Migration-Aware Balancing Algorithm*

Essentially, the proposed migration-aware balancing scheme [3] is divided into monitoring, redistribution, and migration phases, as delimited in [2]. Such phases impose a sequential processing and determine the balancing algorithm. The division of the algorithm into the aforementioned phases simplifies the balancing problems by handling the issues from each phase separately. According to the description in Algorithm 1, the balancing scheme is triggered in cycles to enable responsiveness to load imbalances based on the current distribution of load status. In a balancing cycle, monitoring is the first phase to be

Algorithm 1: Main Load Balancing Algorithm

```
1  while $TRUE$ do
2      $loads \Leftarrow query\_MDS()$
3      $current\_loads \Leftarrow filter\_MDS\_data(loads)$
4      $current\_loads \Leftarrow normalize\_loads(current\_loads, benchmark)$
5      $overload\_cand \Leftarrow select\_overload(current\_loads)$
6      $spec\_loads \Leftarrow request\_LLBs(overload\_cand)$
7      $mng\_loads \Leftarrow filter(current\_loads, spec\_loads)$
8      $mean, bds \Leftarrow calculate\_mean\_bds(mng\_loads)$
9      $over, under \Leftarrow select(mng\_loads, mean, bds)$
10     $mig\_moves \Leftarrow redistribute\_local(mng\_loads)$
11     $mig\_moves \Leftarrow analyze\_migration\_latency(mig\_moves)$
12     $send\_migration\_moves(mig\_moves)$
13     if $mig\_moves = \emptyset$ then
14         $data\_neighbours \Leftarrow request\_Neighbour\_Load\_Data()$
15     else
16         if $relFactor \geq random\_number(1, 100)$ then
17             $data\_neighbours \Leftarrow request\_Neighbour\_Load\_Data()$
18         else
19             $data\_neighbours \Leftarrow \emptyset$
20         end
21     end
22     wait( $\Delta t$ )
23 end
```

conducted because detecting imbalances directly regulates the redistribution algorithm. As a scope of the proposed balancing scheme, only metrics of computational load are observed in redistribution analysis. In order to enable fast responses to imbalances, a greedy technique is employed in the redistribution phase; this problem solving heuristic-based technique provides load reallocation for a delimited set of resources, which enables a global sub-optimal balancing solution. Based on the identified imbalances, the rearrangement of load is defined, and migration calls are issued to simulation federates to conduct migration procedures.

Due to its importance in enabling responsiveness to dynamic, unpredictable simulation load changes, monitoring is crucial and necessary for the redistribution algorithm. The monitoring starts with data gathering by accessing a Monitoring Information Service through a call ($query\_MDS()$) from a CLB to its respective Monitoring Interface. Upon receiving information about the resources' and simulation's load status requested from the monitoring services, data filtering is employed to eliminate non-managed resources or overloaded resources that do not contain any simulation element: i.e., a resource that cannot have its load lowered. Also, normalization of load values ($normalize\_loads(current\_loads, benchmark)$) is applied to solve the inherent issues of resources' heterogeneity by using benchmarks that define computational capacities. Based on the selection of overloaded resources ($select\_overload(current\_loads)$) in the gathered data sample, the balancing system requests more detailed load data ($request\_LLBs(overload\_cand)$) through each LLB: the CPU consumption of each federate, which is used to locally rank federates according to their load. According to the responses from the LLBs, overloaded resources without simulation entities, federates, are then identified and disregarded ($filter(current\_loads, spec\_loads)$). All the collected information is relevant to load status measurements; migration-related metrics also need to be provided by each LLB, which provides these metrics together with the simulation load data. The migration-related metrics are gathered to enable the analysis of migration latency when redistributing simulation load. The migration metrics measured and gathered consist of migration delay, migration distance, and federate state size. Migration delay mostly represents the time in milliseconds spent to transmit a federate's execution-state data and incoming messages. Migration distance contains the topological distance, as well as the communication latency, between the two resources involved with the migration of a federate. Federate state size provides the amount of data transmitted through peer-to-peer data transfer. A migration latency estimation is then calculated based on these three metrics.

The detection/redistribution phase of the balancing scheme is triggered just after all the monitoring data is retrieved and gathered in a CLB, as detailed in Algorithms 1 and 2. The balancing algorithm is composed of local and inter-domain scopes, and the inter-domain load reallocation is triggered based on migrations generated in the local scope of the redistribution algorithm. As a premise for the balancing algorithm, the local load rearrangement is applied as a means of diminishing the load discrepancies of resources delimited in a domain (cluster). In the load redistribution for local scope, the gathered information is analyzed by identifying differences on load distribution based on a calculated average load ($calculate\_mean\_bds(mng\_loads)$). The analysis comprises ordering the resources according to their load and matching pairs of them (overloaded and underloaded) in order to achieve a close-to-even distribution of load in a cluster of resources ($select(mng\_loads, mean, bds)$), as described in Algorithm 3. Based on this selection of resources, federate migrations are determined and analyzed according to their estimated delays.

The inter-domain load redistribution is conducted when modifications of local distribution are not produced ($mig\_moves = \emptyset$) or a set of domains (clusters) show substantial load imbalances, evidencing the continuous need of simulation load's transfers ($relFactor \geq random\_number(1, 100)$). The inter-domain imbalances are defined by the distribution factor ($relFactor$), which represents the ratio between the total number of inter-domain migrations and the total number of selected overload resources in the previous balancing cycle for a local CLB; this evidences the degree of imbalance between a local domain and its neighbours. The $relFactor$ contains a value that ranges between 0 and 1, which then is multiplied by 100 in order to be compared with the randomly generated number. Constant triggering of inter-domain redistribution is prevented in the balancing scheme since the inter-domain redistribution requires costly data gathering, and a low distribution factor in the past balancing cycle might incapacitate the detection of current substantial imbalances through a biased threshold. Therefore, a random, uniformly distributed number is used to add uncertainty to the analysis for the need of inter-domain balancing since

Algorithm 2: Inter-Domain Redistribution Algorithm

```
1  Require: $neighbours\_data$
2  $neighbours \Leftarrow identify\_Neighbour\_Less\_Load()$
3  $order\_neighbours\_by\_load(selectionParameter)$
4  if $neigbours = \emptyset$ then
5      $neighbours \Leftarrow select\_Neighbours(extStD, localStD)$
6      $order\_neighbours\_by\_load(selectionParameter)$
7  end
8  foreach $neighbour$ IN $neighbours$ do
9      $overloaded\_RSCs \Leftarrow select(neighbour)$
10     $federates \Leftarrow select(spec\_loads, overloaded)$
11 end
12 if $overloaded\_RCSs \neq \emptyset$ then
13     $sort\_list\_load(overloaded\_RSCs)$
14     $sort\_list\_load(neighbour\_RSCs)$
15     $moves \Leftarrow redistribute(overloaded\_RSCs, neighbour\_RSCs)$
16     $moves \Leftarrow analyze\_migration\_latency(moves)$
17 end
18 $send\_migration\_moves(moves)$
19 $adjust\_factor(relFactor, overloaded\_RSCs, moves)$
```

load can oscillate dynamically.

In summary, the redistribution algorithm in both scopes act similarly; the difference concerns the inter-domain scope, which requires load status data from the resources in the neighbour domains for the detection of imbalances and load redistribution ($request\_Neighbour\_Load\_Data()$). The retrieved inter-domain load information is used to detect large load differences between a local domain and its neighbours, evidencing the largest imbalance and consequently producing the largest number of migrations. As delineated in Algorithm 2, the underloaded domains are first selected ($identify\_Neighbour\_Less\_Load()$) and then ordered ($order\_neighbours\_by\_load(selectionParameter)$) for the next step in the load analysis if any underloaded domain is detected. In iterations, a set of underloaded neighbour CLBs is selected, and local overloaded resources are determined ($overloaded\_RSCs \Leftarrow select(neighbour)$) according to the redistribution algorithm described in 2. The redistribution algorithms in both scopes present lists of resources organized in descending order based on the load, and the most overloaded resources are selected first to perform pair-match evaluations with the most underloaded resources ($redistribute(overloaded\_RSCs, neighbour\_RSCs)$). This matching continues subsequently in the list until all overloaded resources are assigned to a migration move or if the difference of load between overloaded and underloaded resources does not reach the threshold for creating a migration move. After federate migrations are determined, they are analyzed according to the migration latency they can generate in the simulation execution ($analyze\_migration\_latency(moves)$).

As detailed in Algorithm 3, the pair-match algorithm compares the load of an overloaded resource and an underloaded resource. Such a comparison consists in detecting the difference of load between the two resources and analyzing it based on thresholds ($min * \phi$, $min * \delta$, and $number\_fed(src\_rsc)$): if the difference exceeds a certain value, a migration move is generated. In order to enable migration-aware analysis, estimations are defined for the migration moves that are created. For

Algorithm 3: Pair-Match Evaluation Algorithm

```
1  Require: $src\_rsc, dst\_rsc$
2  $selected\_federate \Leftarrow select\_federate\_smallestLatency(src\_rsc)$
3  if $dst\_rsc < min$ then
4      if $number\_fed(src\_rsc) \geq 1$ & $src\_rsc > (min * \phi)$ then
5          $\Delta t' \Leftarrow \Delta t \times \alpha$
6          $create\_migration\_move(src\_rsc, dst\_rsc, selected\_federate)$
7      else if $number\_fed(src\_rsc) > 1$ then
8          $\Delta t' \Leftarrow \Delta t$
9          $create\_migration\_move(src\_rsc, dst\_rsc, selected\_federate)$
10     end
11 else if $(dst\_rsc - src\_rsc) > (min * \delta)$ then
12     if $number\_fed(src\_rsc) \geq 1$ AND
       $(dst\_rsc - src\_rsc) > (min * \phi)$ then
13         $\Delta t' \Leftarrow \Delta t \times \alpha$
14         $create\_migration\_move(src\_rsc, dst\_rsc, selected\_federate)$
15     else if $number\_fed(src\_rsc) > 1$ then
16         $\Delta t' \Leftarrow \Delta t$
17         $create\_migration\_move(src\_rsc, dst\_rsc, selected\_federate)$
18     end
19 end
20 if $migrationMove$ then
21     $estimatedGain \Leftarrow estimateMigGain(dst\_rsc, src\_rsc, \Delta t')$
22     $estMigTime \Leftarrow$
       $estMigTime(dst\_rsc, src\_rsc, selected\_federate)$
23     Return: $migrationMove, estimatedGain,$
       $estimatedMigTime$
24 end
```

each pair of resources and a selected federate, the migration delay ($estMigTime()$) and the performance gain ($estimateMigGain()$) are determined. Based on its load and migration time, a federate from the overloaded resource is selected for the migration move. The measurements of its last migration move are used to produce an estimation, which is evaluated later in the migration filtering. For creating the migration move, the federate with the lowest load consumption is selected for migration if it is eligible. A federate is considered eligible for migration only if it does not have any restriction for running on a specific resource, such as interfacing the rest of a distributed simulation to a user. The estimations use the past migration distance, migration delay, and federate state size to define the current migration delay. As a result, the migration moves are determined, and migration delays are analyzed and compared to obtain the candidates that most benefit simulation performance.

After all possible pair-matches are analyzed, a list of migration moves is obtained. Migration latency and performance gain estimations are defined for each migration move in such a list based on simple rules of proportionality. The migration delay estimation (t_e) of a federate is calculated with the migration latency and the distance obtained from the last migration move conducted for a federate, as described in Formula 1. Based on the distance between the source and destination resources in the migration move, latency is proportionally estimated with the past and current distances. Similarly to migration latency, the performance gain is defined through Formula 2. Essentially, the gain (t_d) is estimated in time (milliseconds) for the purpose of comparison, and it is computed according to a relation between the $load_{src}$ and the $load_{dst}$ resources. The difference of computational load between such resources provides the performance gain in matters of load, which is then factorized into time when multiplied to Δt.

$$t_e = \frac{time_{mig} \times dist_{mig}}{dist_{dst}} \tag{1}$$

- t_e: estimation of migration delay;
- $time_{mig}$: time of past migration;
- $dist_{mig}$: distance of past migration;
- $dist_{dst}$: distance of estimated migration.

$$t_d = \frac{\Delta t \times (load_{src} - min_{load} - load_{dst})}{load_{src}} \tag{2}$$

- t_d: estimation of time gain;
- $load_{src}$: current load of source resource;
- min_{load}: estimated minimum load;
- $load_{dst}$: current load of destination resource.

Δt is employed as a time factor that delimits the base for estimating the improvement in performance for a migration move. This factor is introduced in the calculation since a more consistent time value truly representing the virtual simulation time cannot be used in the balancing scheme. When keeping the balancing system transparent for distributed simulations and consequently minimizing modifications on simulations' design framework, there is no technique or method that can provide a value that represents the time that a simulation runs. There is no prediction mechanism that can be used by the balancing system to obtain a simulation execution time because simulations can be designed for different purposes and objectives, and, hence, with different, unpredictable simulation execution times. Therefore, the balancing cycle interval is employed in the estimation calculations as the base parameter to define the performance gain in a short period of time whereas no value can be used in the balancing system to represent the total real simulation time.

With the list of migration moves containing the performance gain and migration latency estimations, the redistribution algorithm identifies the migrations that can really provide performance improvement for distributed simulations. In this case, improvement is basically accomplished by allowing migrations that produce a time gain larger than the migration delay. As defined in Algorithm 4, the filtering procedure is applied on the whole set of migration moves in each scope (local or inter-domain). Comparisons are conducted between the accumulated final estimations of performance gain and migration latency. For these accumulated estimations, it is assumed that the final migration latency is determined by the largest time and the performance gain by the sum of time gains, as described in Formula 3. For latency calculation, the largest value produces most of the delay added to simulations, and other values have a slight impact on the final result without majorly contributing to it, as detailed in Formulas 4 and 5. A simple comparison is performed between such accumulated estimations: performance gain (*timeGain*) and migration delay (*expectedMigrationDelay*). Filtering is repeatedly applied while gain is smaller than delay. The filtering is comprised of removing the migration move with the lowest gain from the list of migration moves

Algorithm 4: Migration Latency Filtering Algorithm

```
1  Require: $mig\_moves$
2  if $mig\_moves != \emptyset$ then
3      $timeGain \Leftarrow calculateGain(mig\_moves)$
4      $expectedMigrationDelay \leftarrow calculate(mig\_moves)$
5      while $timeGain \leq expectedMigrationDelay$ do
6          $mig\_moves \Leftarrow eliminate\_smallest\_gain(mig\_moves)$
7          $timeGain \Leftarrow calculateGain(mig\_moves)$
8          $expectedMigrationDelay \Leftarrow calculate(mig\_moves)$
9      end
10 end
11 Return: $mig\_moves$
```

($eliminate\_smallest\_gain(mig\_moves)$). In each iteration, the accumulated gain and latency are calculated and compared until the accumulated gain shows performance improvement on the simulation time.

$$t_{ds} = \sum_{i}^{n} t_{d_i} \tag{3}$$

- t_{ds}: overall sum of migration time gains;
- t_{d_i}: time gain of each migration move.

$$t_{es} = t_{e_{largest}} + \frac{\alpha \times \sum_{i \neq e_{largest}}^{n} t_{e_i}}{(n-1)} \tag{4}$$

- t_{es}: overall estimation of migration delays;
- $t_{e_{largest}}$: largest migration delays in the set of migration candidates;
- α: influence of other migration delays on overall estimation;
- t_{e_i}: delay of a migration move candidate;
- $e_{largest}$: migration move with the largest delay.

$$\alpha = \frac{2 \times \bar{t_e} - 1 + t_{e_{largest}}}{2 \times t_{e_{largest}}} * (k+1) \tag{5}$$

- α: influence of other migration delays on overall estimation;
- $\bar{t_e}$: the mean of migration delay;
- $t_{e_{largest}}$: largest migration delay;
- k: number of migrations with delay larger than mean.

At the end of the migration-aware analysis (filtering), the resulting list of migration moves are forwarded to their respective resource to advance with the migration calls, as shown in Algorithm 1 and 2. Such calls are then issued to their respective MM in the LLB. As soon as the migration procedure finishes in the remote resource by restoring the migrating federate's execution state, new migration metrics are measured and registered in the balancing system to provide up-to-date measurements and improve the redistribution decision making in the next balancing cycles. In the inter-domain scope, after the migrations are issued to their respective source resources, adjustments are calculated to redefine *relFactor* for the next balancing cycle ($adjust\_factor(relFactor, overloaded\_RSCs, moves)$).

IV. Extension

The proposed migration-aware balancing scheme is able to avoid costly migrations and to not jeopardize simulations' performance through modifications in the load distribution. However, the proposed scheme presents some drawbacks and requires modifications to overcome some issues and improve the load redistribution. The needed modifications are incorporated as an extension to the current scheme. Under observation of the balancing behaviour, the extension attempts to mitigate the absence of migration history for simulation federates, to overcome the reduction in the number of redistribution possibilities after migration latency filtering is applied, to use the parallelism of load redistribution as an aspect to improve the filtering, and to reformulate the method used to analyze migration delays in filtering of migration moves.

A. *Absence of Migration History*

As delimited in the migration-aware balancing scheme, the migration information of each federate is kept locally in the Local Monitoring Interface of each LLB. Since an interface is instantiated for each federate and runs simultaneously with it in order to provide information related to a federate, the interface needs to store all the information locally and to provide it upon request of the LLB. Thus, a federate's migration information exists only if the federate has been migrated in any moment of its past execution time. For initial execution of simulations, there is no migration data that can be used to filter and to determine current migration moves. Even for simulations that have advanced execution time, this migration information might be scarce if simulations are composed of a large number of federates when compared to the number of resources, or if load imbalances are not detected so often in the distributed system.

Since information of the previous migration process is vital for the migration-aware analysis, the scarcity of this information leads the balancing system to ignore migration delays when redistributing load. In this case, for every simulation and without exception, there exists a period of time in which the load balancing is not aware of the federate's migration delays. This situation occurs due to the absence of migration history, which feeds the proposed balancing system and allows it to react properly, based on each previous migration latency. Because of its importance, the lack of migration information needs to be prevented or minimized. Therefore, measurements of other federate's migration processes are used to calculate estimations and improve the filtering of load redistribution.

The most recent migration delay or the average of N migration delays of other federates can be used to determine and define a federate's unknown migration time. The most recent delay provides metrics that evidence the most accurate data about the conditions and load status of the communication resources for migration purposes. This technique also enables the matching of federates with similar migration distance, decreasing the discrepancies when assigning other migration delays to a federate. On the other hand, even though the technique increases the chances of existing migration data for the analysis, a small number of migrations still might lead to an absence of migration history. The average of delays is based on a set of federate's migration time values, so it does not represent the most recent measurement of resource conditions for federate migrations since the average might contain old samples. However, the absence of measurements is minimized with this approach since the calculation of the average requires the existence of only one migration time in the set of federates that is considered in the average calculation. The average also provides a value that represents the measurements of a set of federate migrations in the data sample, resulting in a more generic characteristic that can be applied to the estimation of a federate without any migration history.

With the utilization of migration averages, the set of N federates needs to be defined to feed the calculation of the average. In the distributed balancing system, the LLBs aggregate some federates in a resource and can store the average migration information for the federates that run locally in the resource. A CLB can also aggregate all the migration information of federates that are running in a cluster of resources. The latter approach, aggregation of an average in a CLB, avoids the burden of transmitting averages from LLBs to a CLB for redistribution filtering and the introduction of more complexity into the redistribution algorithm. Therefore, a migration latency average is obtained based on the migration moves generated in the last balancing cycle and stored in the CLB; as soon as a federate migration procedure finishes, the migration information is transmitted to the CLBs involved with the particular migration process.

A set of federate migrations might contain load transfers with different characteristics based on transfer distance, execution state size, and time aspects. Since transfer distance and execution state size directly influence the migration time, they need to be considered in the estimation, as well as in the calculation of averages. Same or similar migration end-points are needed to calculate the estimation that best predicts the current migration of a federate. Since this condition leads to a scarce migration history due to the difficulty in matching migrations with the same source and destination resources, it cannot be used. Even similar migration distances between end-points can restrict the matching of a previous federate migration procedure to the characteristics of the current prediction of a migration move. As a result, as a matter of simplification for comparison and for estimation purposes, averages are then calculated for distance, state size, and migration delay; thus, the average distance and average state size are used to directly and proportionally obtain a migration delay.

B. *Reduced Number of Redistribution Possibilities*

In the migration delay analysis of the proposed scheme, the filtering of migration candidates is performed based on the set of migrations identified in the load redistribution algorithm. This filtering prevents costly migrations that might introduce a delay larger than the time gain produced by the simulation load rearrangement. However, the filtering might also prevent the balancing system from iden-

Algorithm 5: Extended Main Load Balancing

```
1  while TRUE do
2      $loads \Leftarrow query\_MDS()$
3      $current\_loads \Leftarrow filter\_MDS\_data(loads)$
4      $current\_loads \Leftarrow normalize\_loads(current\_loads, benchmark)$
5      $overload\_cand \Leftarrow select\_overload(current\_loads)$
6      $spec\_loads \Leftarrow request\_LLBs(overload\_cand)$
7      $mng\_loads \Leftarrow filter(current\_loads, spec\_loads)$
8      $mean, bds \Leftarrow calculate\_mean\_bds(mng\_loads)$
9      $over, under \Leftarrow select(mng\_loads, mean, bds)$
10     $mig\_moves \Leftarrow redistribute\_local(mng\_loads)$
11     while $mig\_moves \neq \emptyset$ do
12         $moves \Leftarrow analyze\_migration\_latency(mig\_moves)$
13         $moves\_excluded \Leftarrow determine\_moves\_excluded()$
14         $add\_RSCs\_to\_list(overloaded\_RSCs, moves\_excluded)$
15         $add\_RSCs\_to\_list(neighbour\_RSCs, moves\_excluded)$
16         $sort\_list\_load(over)$
17         $sort\_list\_load(under)$
18         $moves \Leftarrow redistribute(overloaded\_RSCs,$
           $neighbour\_RSCs, moves\_excluded)$
19     end
20     $mig\_moves \Leftarrow analyze\_migration\_latency(mig\_moves)$
21     $send\_migration\_moves(mig\_moves)$
22     if $mig\_moves = \emptyset$ then
23         $data\_neighbours \Leftarrow request\_Neighbour\_Load\_Data()$
24     else
25         if $relFactor \geq random\_number(1, 100)$ then
26             $data\_neighbours \Leftarrow request\_Neighbour\_Load\_Data()$
27         else
28             $data\_neighbours \Leftarrow \emptyset$
29         end
30     end
31     wait( $\Delta t$ )
32 end
```

Algorithm 6: Extended Inter-Domain Redistribution

```
1  Require: $neighbours\_data$
2  $neighbours \Leftarrow identify\_Neighbour\_Less\_Load()$
3  $order\_neighbours\_by\_load(selectionParameter)$
4  if $neigbours = \emptyset$ then
5      $neighbours \Leftarrow select\_Neighbours(extStD, localStD)$
6      $order\_neighbours\_by\_load(selectionParameter)$
7  end
8  foreach $neighbour$ IN $neighbours$ do
9      $overloaded\_RSCs \Leftarrow select(neighbour)$
10     $federates \Leftarrow select(spec\_loads, overloaded)$
11 end
12 if $overloaded\_RCSs \neq \emptyset$ then
13     $sort\_list\_load(overloaded\_RSCs)$
14     $sort\_list\_load(neighbour\_RSCs)$
15     $moves \Leftarrow redistribute(overloaded\_RSCs, neighbour\_RSCs)$
16     while $moves \neq \emptyset$ do
17         $final\_moves \Leftarrow analyze\_migration\_latency(moves)$
18         $moves\_excluded \Leftarrow determine\_moves\_excluded()$
19         $add\_RSCs\_to\_list(overloaded\_RSCs, moves\_excluded)$
20         $add\_RSCs\_to\_list(neighbour\_RSCs, moves\_excluded)$
21         $sort\_list\_load(overloaded\_RSCs)$
22         $sort\_list\_load(neighbour\_RSCs)$
23         $moves \Leftarrow redistribute(overloaded\_RSCs,$
           $neighbour\_RSCs, moves\_excluded)$
24     end
25 end
26 $send\_migration\_moves(final\_moves)$
27 $adjust\_factor(relFactor, overloaded\_RSCs, moves)$
```

tifying beneficial migration moves. The initial migration moves are selected based on the most suitable resource, and federate candidates are selected based on a load analysis; after some migrations are excluded from the list, there might be other pair-matches or other federates that can still provide performance gain with load transfers.

In order to enable more migration moves, an iterative load redistribution and migration filtering is introduce in the scheme. Consequently, the redistribution algorithm is executed while there still exists possibilities of modifying the simulation load distribution, and the iterations stop when there are no more overloaded resources or there are no more possible load changes (underloaded resources that can receive a federate). For enabling this iterative load redistribution, the existent proposed scheme needs to be modified. The redistribution algorithms, as shown in Algorithm 1 for local scope and in Algorithm 2 for inter-domain scope, are extended to accommodate the iterative load reorganization and filtering, as described in Algorithm 5 and Algorithm 6.

In the non-extended redistribution algorithm, as soon as a migration move is generated, the resources involved with the move are removed from, or unchecked in, the list of resources; thus, they are not used for the next pair match selection. Both extended redistribution algorithms continue performing this same technique; however, they re-insert the resources that were in the migration moves and were rejected in the filtering ($add\_RSCs\_to\_list(overloaded\_RSCs, moves\_excluded)$ and $add\_RSCs\_to\_list(neighbour\_RSCs, moves\_excluded)$) into the list of resources (overload and underloaded). As these resources are inserted in the lists, the lists are reorganized ($sort\_list\_load(overloaded\_RSCs)$ and $sort\_list\_load(neighbour\_RSCs)$), and the load redistribution is executed again to identify more needed modifications on the load distribution, reconsidering the resources that were rejected in the previous iteration. This migration filtering and load redistribution is conducted in iterations while no more federate migration moves are generated ($moves \neq \emptyset$).

However, adding resources for redistribution analysis after the filtering might lead the matching process to an endless loop. It is most likely to match the same pair of resources for the purpose of performing the load comparison, and the selected migration move is then rejected in filtering. Thus, when observing the extension on algorithms, another modification is realized, adding the excluded migration moves for the redistribution analysis. During the evaluation of resources before pair-matches are conducted, the resources' conditions are compared with the respective migration moves generated and rejected in the past iterations. The comparison consists in first determining if there is any other federate with a smaller migration delay than the federates selected in the past migration moves for the same distance. Federates are likely to present different execution state sizes and consequently different estimated migration times for the same migration distance. Thus, if there is any other federate that matches this condition, another attempt to create a migration move with the same pair resources is realized. On the other hand, if there is no other federate with a smaller migration delay, other underloaded resources are searched for and selected to match an overloaded resource following the redistribution procedure.

The load rearrangement and filtering iterations are finalized when there is no migration move produced by the redistribution algorithm. This attempt to perform additional searches for resources' pair-matches might not produce considerable improvement on simulation execution performance, but it can improve the balancing responsiveness to load imbalances.

C. Migration Parallelism of Load Redistribution

As described in Formulas 3 and 4, the migration moves defined in a balancing cycle tend to be performed in parallel since they are launched mostly at the same time and last for a certain time interval (migration delay). Because these migrations occur in a semi-parallel trend, the effect on the simulation execution time is calculated based on Formula 4, and the gain is estimated in Formula 3 as the sum of every migrating federate's execution time improvement. Nevertheless, the analysis based on these two estimations, migration delay and performance gain, leads to the production of migration moves that generate performance loss in a simulation. Since the overall gain estimation is obtained through a sum, the analysis wrongly allows the execution of more migrations than are necessary for the achievement of a gain. Therefore, the estimation of overall performance gain needs to be redefined by considering that just a sum as estimation cannot represent the real gain of a load redistribution.

Since the gain affects the simulation performance by speeding up the processing of a federate's tasks, the smallest gain determines most of the gain generated by the migration moves defined in the load rearrangement algorithm. The smallest gain corresponds to the federate that is predicted to present the slowest processing, based on the difference of loads. Because a federate's load cannot be precisely predicted due to its dynamic execution characteristics, it is assumed that a migrating federate's execution acts according to information obtained through comparing the load of its hosting resource with that of the resource designated for migration. As a result, as described in Formula 6, the smallest performance gain estimation ($t_{d_{smallest}}$) is used to calculate the overall gain (t_{ds}). The gain of other federate migrations are considered in the calculation, but they have less influence on the final overall estimation value.

$$t_{ds} = t_{d_{smallest}} + \frac{\beta \times \sum_{i \neq d_{smallest}}^{n} t_{d_i}}{(n-1)} \tag{6}$$

- t_{ds}: overall time gain;
- $t_{d_{smallest}}$: smallest time gain in migration moves;
- β: influence of other time gains on overall estimation;
- $d_{smallest}$: migration move with the smallest time gain;
- t_{d_i}: time gain of a migration move.

$$\beta = \frac{2 \times \bar{t_d} - 1 + t_{d_{largest}}}{2 \times t_{d_{largest}}} * (k+1) \tag{7}$$

- β: influence of other time gains on mean;
- $\bar{t_d}$: the mean of time gain;
- $t_{d_{largest}}$: smallest time gain in migration moves;
- $t_{d_{largest}}$: largest time gain in migration moves;
- k: number of time gains larger than the mean.

As defined in Formula 7, β restricts the influence of the average gain in a set of migrations. Similarly to α in Formula 5, β consists in representing the influence that the performance gains exercise on the simulation execution time; the value of β is determined proportionally to the amount of gain close to $t_{d_{largest}}$. k in the formula represents the number of time gains with value over $t_{d_{largest}}/2$. Even though, theoretically, the gain is limited by the smallest performance gain, the overall gain might be influenced by other factors, or it may present a different real final gain after the migrating federates resume their execution. Since all analysis is based on estimations, which might present final effects on simulations different from the ones that are expected, the final load redistribution gain also includes the estimations of other migrations. Consequently, other gains might also present a slight influence on the simulation performance; this influence is defined and represented by β. Based on this assumption, less migration moves are enabled and a more conservative responsiveness, regarding the awareness of migration latencies, is introduced.

D. Reformulation of Migration Analysis

As defined in Formula 2, the proposed analysis employs a static time interval (Δt) as a base for the gain estimation. This time interval is used to evaluate the migration moves because of the difficulty in predicting the period of time required for the conclusion of a simulation execution: the simulation can run for a long period of time or can end its execution in the next simulation time step. Consequently, the fixed parameter is defined as a base of comparison between migration delays and performance gain, but this value might not really represent the time that can be used to compare with delays. Moreover, the formula used in the analysis can be redefined to represent performance in time with improved accuracy.

D.1 Modification on the Estimation Formula

In Formula 2, the design is defined by a rule of proportion to provide a gain when compared to the execution status of a federate placed on an overloaded resource. Consequently, the difference of load between a pair of resources dictates the amount of gain based on the time spent to produce the work for Δt with load on the source resource. The calculation based on proportionality provides a reasonable solution for obtaining a time value out of Δt, but load is used as the major aspect to perform the calculation. On the other hand, gain can be represented by the work that can be produced with the available resources and time ($w = t \times r$), in which available resources is inversely proportional to the load of a CPU. Therefore, as delineated in Formula 8, t_d is also obtained through a rule of proportion, but it is based on the difference of available resources between an overloaded resource and an underloaded resource.

$$t_d = \frac{\Delta t \times (load_{src} - min_{load} - load_{dst})}{2 \times load_{dst}} \tag{8}$$

- t_d: estimated delay of a migration move;
- Δt: balancing time interval;

- $load_{src}$: load of source resource;
- min_{load}: estimated minimum load;
- $load_{dst}$: load of destination resource.

D.2 Pseudo-Dynamic Time Interval

The use of a static, predefined time interval in the formula might mislead the migration delay analysis. The analysis might allow costly migration moves or block useful, fast migration procedures when compared to the total remaining simulation time. Thus, as an attempt to introduce some dynamic aspects on the time interval used in the decision-making of the migration delay analysis, the value of Δt is modified in certain circumstances to enable the production of migration moves that are considered costly in the analysis but possibly useful for the simulation performance.

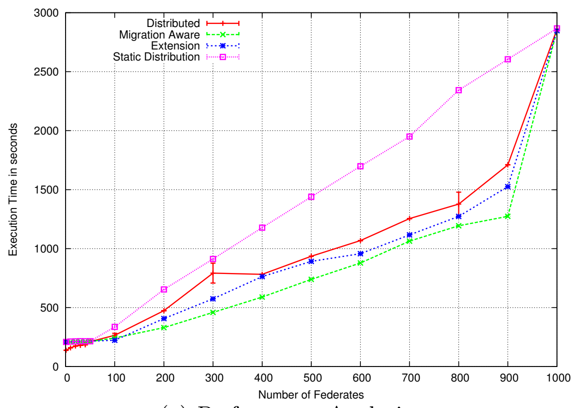
In each balancing cycle, after migration moves are filtered, the value of Δt is incremented based on the ratio between the number of generated migration moves and the total number of source resources ($r = moves/RSCs$). With this approach, the balancing system might enable useful costly migration moves by introducing some tolerance to migration delays. The ratio represents the need to redistribute load for the distributed system and provides an assessment to identify and measure the amount of tolerance the balancing system can allow through the increase of Δt value. This adjustment on the time interval is performed only if there is no filtered migration move, and the approach is executed in iterations. The incremented time interval ($\Delta t'$) is obtained by adding up to 50% of its value based on the ratio ($\Delta t' = \Delta t + r \times \Delta t/2$). Such increments are applied until any migration move is in the filtered list. In summary, with this technique migration moves are blocked until the cost/benefit ratio reaches a threshold that enables it to pass through the migration delay filtering. Thus, even migrations considered as costly by the proposed scheme are allowed to be conducted; this is undergone with the expectation that the move is beneficial to the simulation performance.

V. Experimental Results

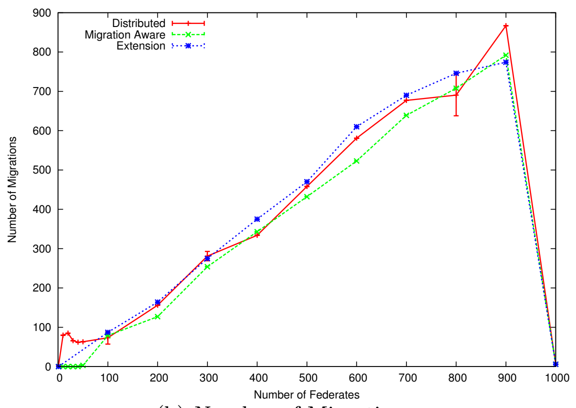
Experiments have been conducted to evaluate the effectiveness of the proposed and extended balancing schemes, observing the balancing efficiency gain when improving simulation performance. In the experiments, the proposed migration-aware balancing scheme, the extended version of the proposed approach, and the distributed balancing scheme [2] are compared using a testbed. The testbed used in the experiments was an environment composed of two computing clusters inter-connected through a fast-Ethernet network link. One cluster was comprised of a set of 24 computing servers that were connected through a Myrinet optical network, which enabled data transfers of up to 2 gigabits per second. Each computing server consisted of a Quadicore 2.40GHz Intel(R) Xeon(R) CPU and 8 gigabytes of RAM memory. The second cluster was composed of a set of 32 computing servers that were connected through a gigabit Ethernet network. Every server in this cluster presented a Core 2 Duo 3.4 GHz Intel(R) Xeon(R) CPU and 2 gigabytes of RAM. All the computing servers had Linux operating system installed on them. HLA RTI version 1.3 was used to consistently coordinate the execution of the distributed virtual simulations. Globus toolkit was installed to provide Grid Services to the balancing systems deployed in both clusters.

The experimental scenario consisted in evenly deploying the simulation federates on all available shared computing servers except the dedicated server that received the HLA RTI executive. With this deployment, an equal or similar amount of load (number of federates) was assigned to the 55 servers in both clusters. This deployment of simulation components and entities was also used as the baseline in performance gain comparisons; with this placement of load, even the most simplistic scenario (static simulation load) presented imbalances due to the heterogeneity of resources: the servers with Quadicore CPUs approximately outperformed by 4 times the other CPUs in computing power aspects. Moreover, for all the evaluated balancing systems, each LLB was deployed on a computing server in order to monitor each set of local federates, and a CLB was placed on the management node of each cluster to manage the federates running in a cluster.

The distributed simulation scenario comprised of the movement control of tanks' teams for training operations in a two dimensional routing space. The teams were composed of interactive tanks that required data exchange to determine positions, which moved according to a time-stepped simulation model. Running for 100 simulation time steps, each simulation ranged from 1 to 1000 federates. Each federate coordinated the movement of only one tank (simulation object), published the resulting calculated tank's position to related federates, and subscribed for tanks' updates based on the interest regions related to the position of its own tanks in the routing space. Because the calculation of tanks' movement did not impose considerable load on the CPUs involved with the simulation, synthetic load [38], [39] was added into each federate. This produced intensive computing processing, which generated enough load to totally consume a processor's computational resource for a whole simulation time step. The synthetic load was intended to create a static, intensive load for all federates that could identically replicate load scenarios for comparison purposes. In dynamic scenarios, the simulation load was randomly selected to change its processing intensity to complex or simple calculations each 40 seconds and thus dynamically increase or decrease the CPU consumption of a resource. Furthermore, additional objects were incorporated into federates in order to increase their execution state size, but these objects did not participate in the simulation model for the purpose of minimizing the influence of communication load and emphasizing aspects of computational load. The number of these additional objects assigned to each federate were 1, 200, 400, and 600. Since the network resources provided a high bandwidth connection for data transfers, even a high number of objects exerted little influence on the migration

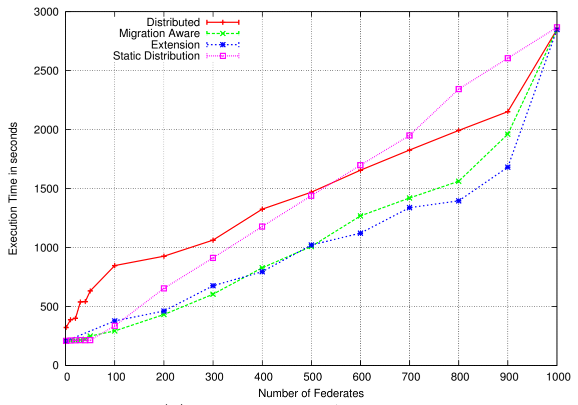


(a) Performance Analysis

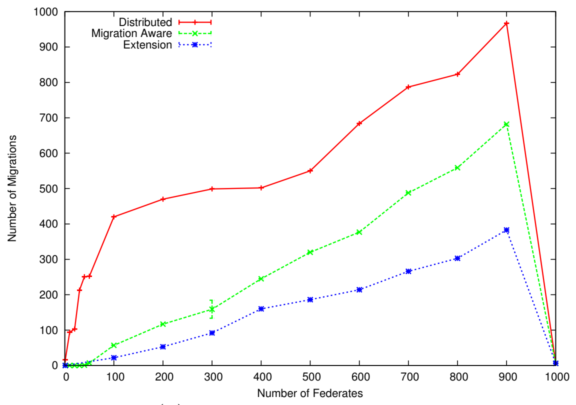


(b) Number of Migrations

Fig. 2. Performance Gain Analysis for an Increasing Number of Federates with Static Load and Migration Latency of 1 object



(a) Performance Analysis



(b) Number of Migrations

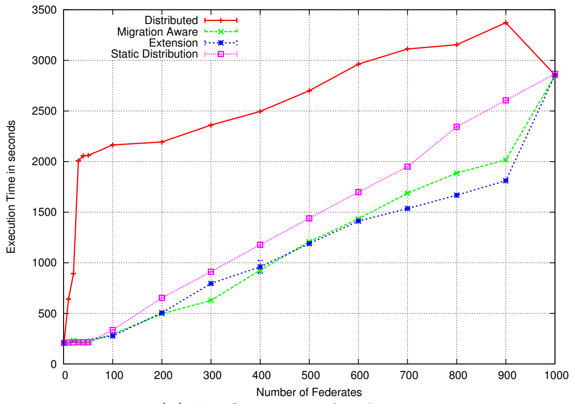
Fig. 3. Performance Gain Analysis for an Increasing Number of Federates with Static Load and Migration Latency of 200 objects

latency. Thus, a latency proportional to the number of objects was also added to the migration process to simulate the transmission in high latency networks.

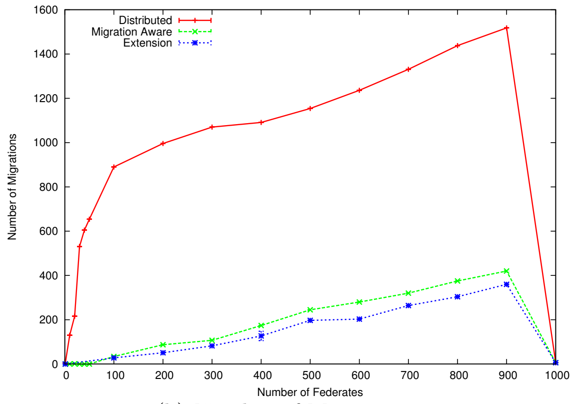
A. Static Simulation Load

As described previously, the simulation in this study case scenario presents static load that constantly consumed the CPU resources with the same intensity. In this set of experiments, the efficiency of reacting to static imbalances is evaluated by comparing the distributed, migration-aware, and extended balancing schemes with a base line, initial even static distribution of load. Also, in this case study, four different migration delays are applied on the system in order to analyze the efficiency of detecting delays and reorganizing simulation load distribution by the migration-aware approaches.

As shown in Figure 2(a), the three balancing schemes were able to present similar performance gains for the simulation. They also showed approximately the same amount of migrations required to reach simulation performance improvement, as described in Figure 2(b). Since the core algorithms of proposed balancing schemes are based on the distributed balancing approach, it was expected that they would show simulation performance improvements with close values in simulations that have little migration overhead. However, in Figure 2(a), migration latency was increased by adding 199 objects to the migration transfer, and the distributed balancing scheme presented a significant performance loss due to the migration delays when comparing its curve to the imbalanced static distribution's curve for simulations under 500 federates. Consequently, the discrepancy in simulation performance gain increased between the migration-aware approaches and the distributed approach, presenting a maximum difference of 49% in simulation time. This improvement in performance was confirmed in Figure 3(b), which depicts the amount of migrations in which simulation performance improved. In this case, where the simulation performance gain reached the maximum difference between the distributed and extended migration-aware techniques in the graph, the extended migration-aware technique presented a 69% decrease in the number of migrations. This significant improvement in the balancing efficiency of the extended scheme outperformed the migration-aware scheme due to the modifications in the migration delay analysis, such as allowing more parallel migrations, whose delays did not sum up to result in an influence on simulation time. A particular balancing behaviour was also evidenced in both

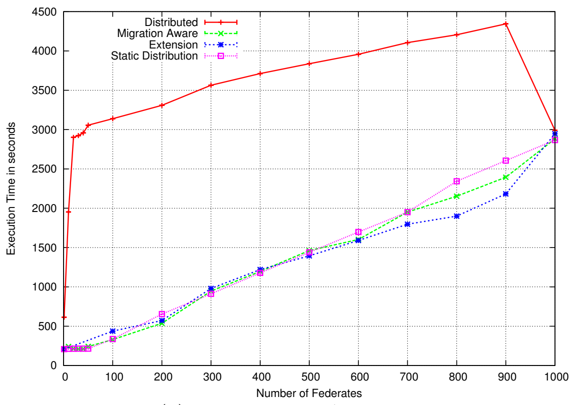


(a) Performance Analysis

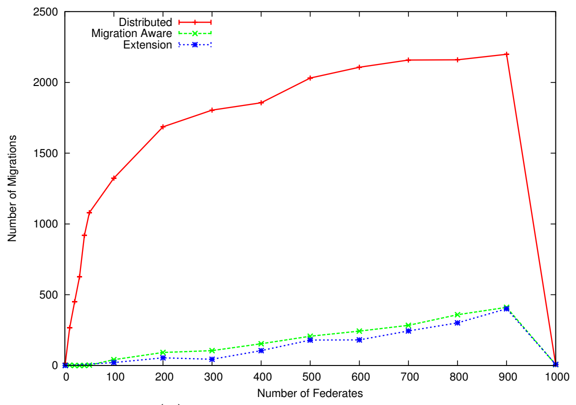


(b) Number of Migrations

Fig. 4. Performance Gain Analysis for an Increasing Number of Federates with Static Load and Migration Latency of 400 objects



(a) Performance Analysis

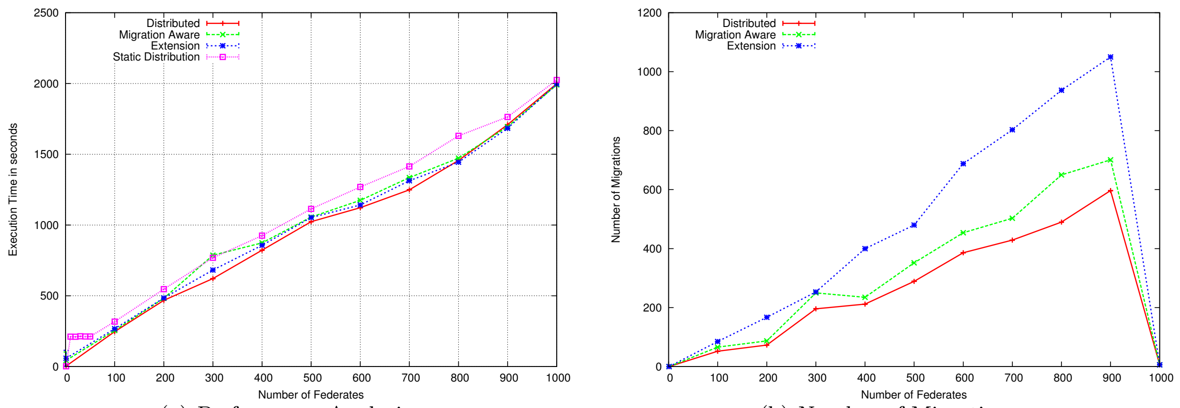


(b) Number of Migrations

Fig. 5. Performance Gain Analysis for an Increasing Number of Federates with Static Load and Migration Latency of 600 objects

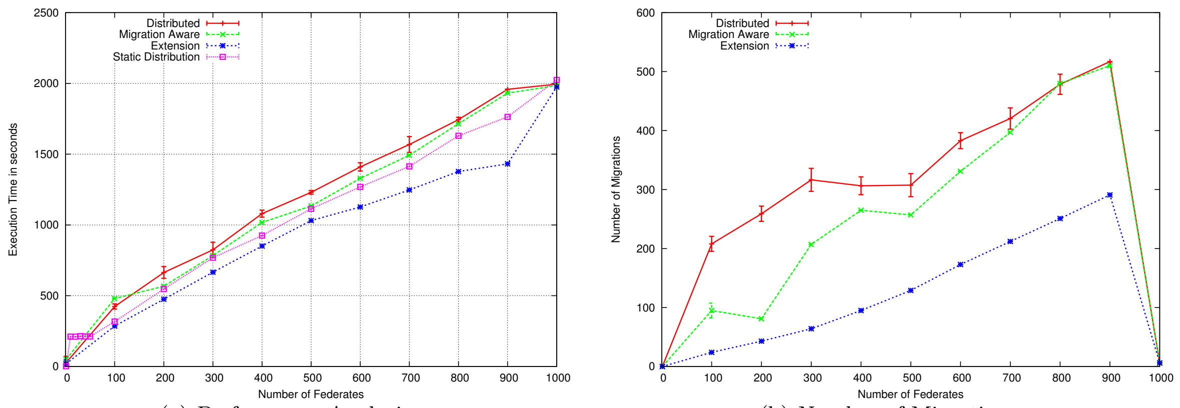
Figures 2 and 3, as well as any other experimental result; when the experiments reached 1000 federates, all curves in any simulation performance analysis graph converged to a single point, and all curves in any number-of-migration analysis graph showed zero or nearly zero migrations. This specific behaviour developed from the saturation of the distributed environment; the federates which were deployed on shared resources caused very high overload on the entire environment, and no migration move was able to improve the simulation performance.

As the migration latency introduced in the federate transfers increased, the difference in performance gain between the migration-aware and distributed balancing schemes grew. As shown in Figures 4(a) and 5(a), the distributed balancing technique only caused overhead to simulations with any number of federates. This was a result of its lack of awareness of migration latency because its redistribution algorithm continued to perform migration as the migration delays were nonexistent. Figures 4(b) and 5(b) highlight the performance loss introduced by the distributed technique; it produced a larger number of migrations, generating 5.04 times more migrations for 400 objects and 8.84 times more migrations for 600 objects with simulations of 700 federates. When comparing the extended approach to the original migration-aware technique, the extended approach was able to improve balancing efficiency: to decrease simulation time and the number of migrations. For the 400-object migration scenario, the extended version could improve performance gain 22.6% with a reduction of 29% in the number of migrations, and for the 600-object migration scenario, the extended version produced 22.8% performance improvement and a 27.2% reduction in migrations. These results show that the extension for the migration-aware scheme could improve the balancing efficiency, even allowing more migrations to be performed when conducting filtering, as described in Sections B and C. However, the migration-aware scheme showed a trend among the migration analysis graphs (200, 400, and 600); as the amount of migration latency increased between the experiments (graphs), this balancing technique presented a number of migrations that were closer to the extended version of it. The behaviour originated from the migration latency, which was prevented due to its high values, even without the improvements on the load redistribution scheme.



(a) Performance Analysis (b) Number of Migrations

Fig. 6. Performance Gain Analysis for an Increasing Number of Federates with Dynamic Load and Migration Latency of 1 object



(a) Performance Analysis (b) Number of Migrations

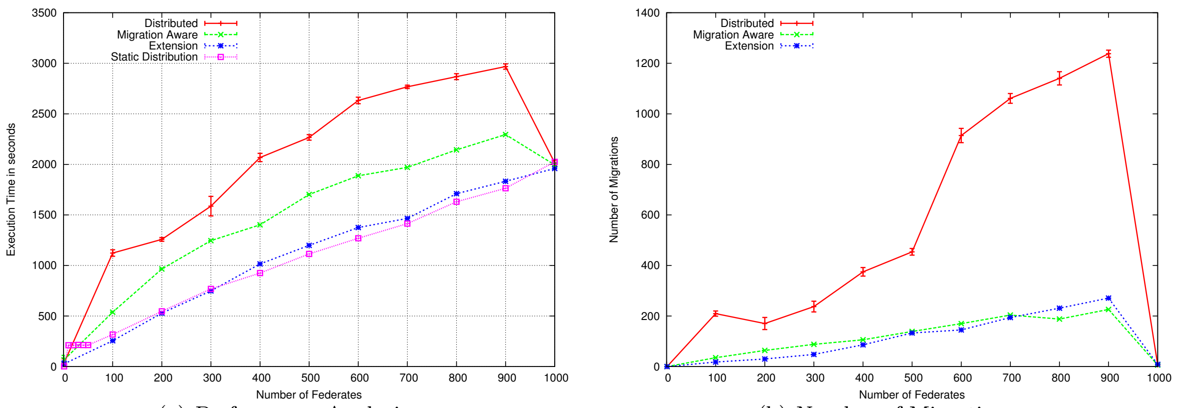
Fig. 7. Performance Gain Analysis for an Increasing Number of Federates with Dynamic Load and Migration Latency of 200 objects

B. Dynamic Simulation Load

In this case study scenario, simulations with dynamic load changes were used to evaluate the balancing schemes. Such simulations presented load that oscillated during runtime: totally consuming CPU resources or producing little load. The introduction of dynamic load changes enabled the observation of the balancing response to unpredictable load behaviour. As described in the previous section, the three balancing schemes were evaluated by comparing them with a baseline: a simulation statically deployed and without any dynamic balancing. Four experiments were conducted in this study case; each of them was conducted with a different migration latency in order to analyze the balancing efficiency loss as migration latencies were introduced in the system.

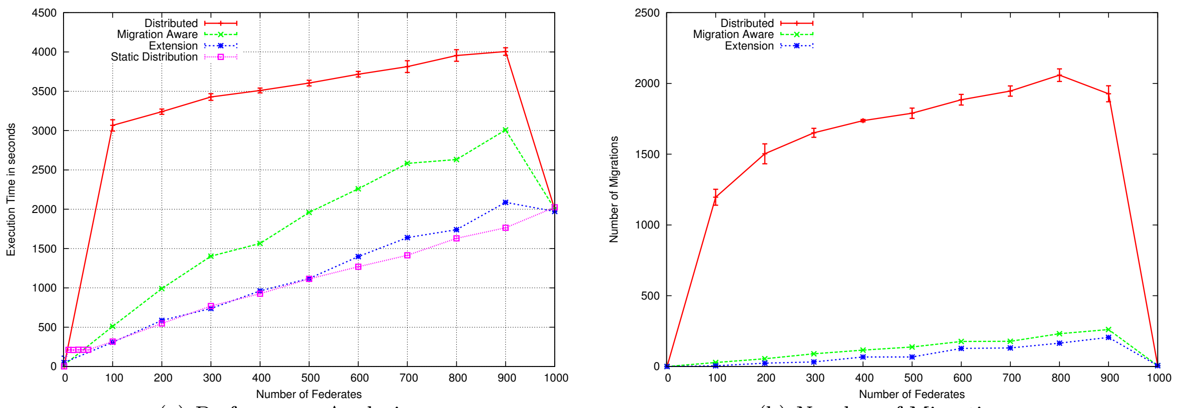
As depicted in Figure 6(a), the low latency scenario allowed the balancing schemes to perform their redistribution of simulation load as no migration delay existed. The distributed balancing scheme showed a slightly better simulation performance gain: a maximum of 6.1% time decrease when compared with other balancing approaches. It also produced less migrations when reaching similar performance gain: a reduction of 48.8% in number of migrations, as described in Figure 6(b). This occurred due to the enforcement of producing essential migrations in the migration-aware balancing system when migration delays did not influence the performance gain. However, when a 200-object latency was introduced in the federate migrations, this favourable scenario changed, as shown in Figures 7(a) and 7(b). The distributed approach presented a decrease in efficiency; it performed a larger number of migrations, 77.7% more migrations, and resulted in worse simulation times for all simulations. The migration-aware approach also introduced performance loss to simulations through a larger number of migrations, resulting in a similar amount of migrations as the distributed approach. The extension still provided performance gain to the simulations with a decrease in number of migrations when compared with the other two approaches. The improvement in efficiency of the extended scheme was achieved through formulas that calculate performance, which produced a lower value that increased as the imbalances grew.

In the simulations described in Figures 8(a) and 5(a), all the balancing systems were unable to produce any simulation performance gain. The migrations in this case generated substantial time overhead on the system and any modification on the load distribution was useless due to the time spent transferring migrating objects. The dis-



(a) Performance Analysis (b) Number of Migrations

Fig. 8. Performance Gain Analysis for an Increasing Number of Federates with Dynamic Load and Migration Latency of 400 objects



(a) Performance Analysis (b) Number of Migrations

Fig. 9. Performance Gain Analysis for an Increasing Number of Federates with Dynamic Load and Migration Latency of 600 objects

tributed balancing scheme, unaware of the migration delays, caused a large performance loss in the simulations; this can be observed in the large amount of migrations produced, as detailed in Figures 8(b) and 9(b). The migration-aware scheme also generated overhead in the simulations instead of improvements in execution time, but the loss it introduced was much smaller than the distribution approach's loss: 23% smaller for the 400-object scenario and 43% smaller for the 600-object scenario. The extension also did not provide any improvement, but the overhead produced for the simulation time was much smaller than the original migration-aware technique: 21.1% for the 400-object migrations and 21.7% for the 600-object migrations. These cases resulted from the prevention of costly migrations: 29% less in the 600-object migrations. However, for the 400-object scenario, the migration-aware approach produced less migrations than the extension did. In this case, the extension enabled the creation of a group of migrations that produced less overall migration delays to simulations. Using the concept that migrations have a high probability of occurring in a parallel manner, the extended technique enabled migrations that could achieve performance improvement even with considerable migration delays.

The migration-aware balancing scheme and its extension were able to improve simulation performance in such dynamic load scenarios. The results showed substantial improvement due to the chosen case study, which had an emphasis on migration delay. In experimental scenarios with simulations composed of federates with different migration delays, the schemes are still able to detect the delays and produce load redistribution properly. However, the overall gain provided by the balancing systems is in this case similar or slightly less than the described results because of the smaller influence of migration delay on simulation time. In another simulation scenario, federates with dynamically-changing migration delays can also be detected; however, because of the varying migration time, the estimations might be misled proportionally to the frequency and intensity of delay variations: increasing or decreasing the balancing responsiveness.

VI. Conclusion

In this paper, a migration-aware load redistribution scheme and an extension are proposed to manage the load of distributed virtual simulations in the case of federate migration latencies. Both balancing schemes rely on the measurement of load and migration metrics, detection of imbalances, redistribution of load, filtering of migration moves, and execution of migration procedures. All these tasks are divided into monitoring, redistribution, and migration phases, which reallocate simulation load in local and inter-domain scopes. The architectural components of the schemes are organized hierarchically in domains, following the placement of resources in an environment. In order to introduce awareness of migration costs, estimations are calculated based on measurements of migration metrics, such as migration time, migration distance, and federate state size. Such estimations enable valuable migration moves to be determined for improving simulation performance. The extension introduced some modifications on the balancing algorithms and calculations to improve the analysis of such migration costs.

Experiments have been conducted in order to analyze the improvement in simulation performance and the balancing efficiency of the proposed balancing schemes. For such experiments, a testbed containing different predefined migration latencies has been used to identify the balancing response when costly migrations exist. The evaluation of these schemes consisted in comparing the reduction of simulation time and the number of migrations of the proposed migration-aware balancing scheme and its extended version with a distributed balancing scheme. As delineated in the experimental results, both proposed schemes were able to react to load imbalances according to migration latencies that were imposed by the experimental distributed simulations. The extension also improved the original migration-aware balancing approach by allowing necessary costly migrations or by grouping such migrations so their influence on simulation performance would be minimized. As future work, additional experiments will be performed to observe the responsiveness of the proposed balancing schemes in simulations containing federates with variable execution state sizes. These variations in size require further improvement of estimations to determine the correct migration costs. The difficulty in defining Δt still remains due to the complexity of predicting the total simulation execution time; thus, additional studies are required to better define flexible (variable) time values to determine more precise estimations; the balancing system may measure the reaction of system distribution and simulation performance to enforce modifications on such time intervals. These variations in execution size require further improvement on estimations to determine the correct migration costs. Such an improvement is achieved through further analysis of the current or additional estimation methods, regarding the accuracy in defining the cost and benefit of migrations. As another method to improve estimations, dynamic adjustments on Δt can also be introduced according to adaptive techniques that observe and react to the balancing behaviour.

Acknowledgement

This work is partially supported by NSERC, the Canada Research Chair program, MRI/ORF research funds, the Ontario Distinguished Research Award, and the EAR Research Award.

References

[1] Simulation Interoperability Standards Committee (SISC), "Ieee standard for modeling and simulation (m&s) high level architecture (hla) framework and rules," IEEE Computer Society, September 2000.

[2] R. E. De Grande and A. Boukerche, "A dynamic, distributed, hierarchical load repartitioning for hla-based simulations on large-scale environments," in *Proceedings of the International European Conference on Parallel Processing (Euro-Par)*, 2010, To Appear.

[3] R. E. De Grande and A. Boukerche, "Dynamic load redistribution based on migration latency analysis for distributed virtual simulations," in *Proceedings of the IEEE International Workshop on Haptic Audio visual Environments and Games (HAVE)*. 2011, IEEE Computer Society.

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

[5] M. Y. H. Low, "Dynamic load-balancing for bsp time warp," in *Proceedings of the 35th Annual Simulation Symposium (SS02)*. 2002, pp. 267–274, IEEE Computer Society.

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

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

[8] J. Jiang, R. Anane, and G. Theodoropoulos, "Load balancing in distributed simulations on the grid," in *Proceedings of the International Conference on Systems, Man and Cybernetics*. 2004, pp. 3232–3238, IEEE Computer Society.

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

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

[11] A. Boukerche and C. Tropper, "A static partitioning and mapping algorithm for conservative parallel simulations," in *Proceedings of the 8th workshop on Parallel and distributed simulation*. 1994, pp. 164–172, IEEE Computer Society.

[12] A. Boukerche, "An adaptive partitioning algorithm for conservative parallel simulation," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium*. 2001, pp. 133–138, IEEE Computer Society.

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

[14] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello, "An adaptive load balancing middleware for distributed simulation," in *Workshop on Middleware and Performance (WOMP)*, 2006, pp. 864–872.

[15] R. E. De Grande, A. Boukerche, and H. M. S. Ramadan, "Decreasing communication latency through dynamic measurement, analysis, and partitioning for distributed virtual simulations," *Instrumentation and Measurement, IEEE Transactions on*, vol. 60, no. 1, pp. 81–92, jan. 2011.

[16] R. E. De Grande and A. Boukerche, "Dynamic partitioning of distributed virtual simulations for reducing communication load," in *Proceedings of the IEEE International Workshop on Haptic Audio visual Environments and Games (HAVE)*. 2009, pp. 176–181, IEEE Computer Society.

[17] Robson Eduardo De Grande and Azzedine Boukerche, "Distributed dynamic balancing of communication load for large-scale hla-based simulations," in *IEEE Symposium on Computers and Communications*, IEEE Computer Society, Ed., 2010, pp. 1109–1114.

[18] R. E. De Grande, A. Boukerche, and H. M. S. Ramadan, "Measuring communication delay for dynamic balancing strategies of distributed virtual simulations," *Instrumentation and Measurement, IEEE Transactions on*, vol. 60, no. 11, pp. 3559–3569, nov. 2011.

[19] D. W. Glazer and C. Tropper, "On process migration and load balancing in time warp," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 3, pp. 318–327, 1993.

[20] C. Burdorf and J. Marti, "Load balancing strategies for time warp on multi-user workstations," *The Computer Journal*, vol. 36, no. 2, pp. 168–176, 1993.

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

[22] G. Tan and K. C. Lim, "Load distribution services in hla," in *Proceedings of 8th IEEE Distributed Simulation and Real-time Applications*. 2004, pp. 133–141, IEEE Computer Society.

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

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

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

[26] J. Luthi and S. Grossmann, "The resource sharing system: dynamic federate mapping for hla-baseddistributed simulation," in *Proceedings of the 15th Workshop on Parallel and Distributed Simulation*. 2001, pp. 91–98, IEEE Computer Society.

[27] W. Cai, S. J. Turner, and H. Zhao, "The resource sharing system: dynamic federate mapping for hla-based distributed simulation," in *Proceedings of the 6th International Workshop on Distributed Simulation and Real-Time Applications*. 2002, pp. 7–14, IEEE Computer Society.

[28] K. Zajac, M. Bubak, M. Malawski, and P. Sloot, "Towards a grid management system for hla-based interactive simulations," in *Proceedings of the 7th International Symposium on Distributed Simulation and Real-Time Applications*. 2003, pp. 4–11, IEEE Comp. Society.

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

[30] C. D. Carothers and R. M. Fujimoto, "Efficient execution of time warp programs on heterogeneous, now platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 3, pp. 299–317, mar 2000.

[31] A. Boukerche and R. E. De Grande, "Dynamic load balancing using grid services for hla-based simulations on large-scale distributed systems," in *Proceedings of the 13th International Symposium on Distributed Simulation and Real-Time Applications*. 2009, pp. 175–183, IEEE Computer Society.

[32] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, no. 6, pp. 37–46, 2002.

[33] A. Boukerche and R. E. De Grande, "Optimized federate migration for large-scale hla-based simulations," in *Proceedings of the 12th International Symposium on Distributed Simulation and Real-Time Applications*. 2008, pp. 227–235, IEEE Computer Society.

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

[35] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[36] "Globus," University of Chicago, 7 Feb. 2008.

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

[38] Pankaj Mehra and Benjamin W. Wah, "Synthetic workload generation for load-balancing experiments," in *Proc. First Symposium on High Performance Distributed Computing*, 1995, pp. 208–217.

[39] Pankaj Mehra and Benjamin W. Wah, "Physical-level synthetic workload generation for load-balancing experiments," in *International Symposium on High Performance Distributed Computing*, 1992, pp. 208–217.