Self-Adaptive Dynamic Load Balancing for Large-Scale HLA-based Simulations †

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

***Abstract*—Large-scale HLA-based simulations are susceptible to performance issues caused by load imbalances. High Level Architecture (HLA) was designed to organize distributed simulations, but it does not provide any mechanism to prevent imbalances. Also, several load balancing schemes have been proposed to properly re-arrange simulation load, but they do not present all the features needed for large-scale environments. A hierarchical, distributed balancing scheme has been designed to support the execution of such simulations; however, due to the balancing inter-relations, heterogeneity of resources, and cyclic load changes, the scheme reacts unnecessarily. Thus, a self-adaptive balancing scheme is proposed in order to dynamically re-configure the redistribution scheme and avoid the ones that are needless. Experimental results showed that the adaption technique improved the balancing efficiency since less migrations were required to achieve similar or better performance.**

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

I. Introduction

Large-scale HLA-based simulations deeply depends on the available shared resources to execute properly and in reasonable time. These simulations can undergo load imbalances due to the characteristics that pertain to large-scale environments and simulations, such as heterogeneity of resources, external background processes, and load oscillations. Even though static load partitioning may solve load distribution issues for deterministic simulations running on dedicated resources, it cannot predict the load changes caused simulation entities or external load. Dynamic balancing schemes are able to identify these load changes, but they can be misled by differences in resource capacity and dynamic load changes. Therefore, an adaptation technique is required to adjust such balancing system to improve its efficiency accordingly.

The HLA specification [1] has been delimited as the defacto standard for designing and coordinating distributed simulations. These standard consists of a set of rules, interface specifications, and object model templates to enable re-usability and interoperability of simulation elements, called federates. An HLA simulation, which is called federation, is composed of Run Time Infrastructure (RTI) management services and federates. The RTI services are responsible for controlling the simulation interactions according to time constraints and the distribution of simulation data. However, such RTI services are not aware of load imbalances caused by irregular distribution of federates on shared resources.

Several balancing schemes have been proposed aiming to improve distributed simulations' performance. Among these solutions, a hierarchical, centralized approach has been devised to promote load balancing for large-scale HLA-based simulations. Even though the centralization facilitates redistribution through a view of entire simulation, it introduces global synchronization. As a solution for the synchronization issues, a distributed balancing approach has been developed to provide independence among the balancing elements, but with this technique, additional migrations are realized to achieve simulation performance similar to the centralized approach.

In order to reduce the number of migrations caused by high responsiveness to load imbalances, a self-adaptive dynamic load balancing scheme is proposed. This scheme is based on the same balancing functionality and architecture as the distributed balancing approach. However, it introduces an adaptation technique that detects load redistribution irregularities caused by particular characteristics originated from the distributed load or environment.

The remainder of this paper is organized as follows. In section 2, the related work and challenging issues are presented. In section 3, the distributed load balancing scheme and the self-adaptation components are described. In section 4, the adaptation is detailed. In section 5, experiments are delineated, and their results are discussed. Finally, the conclusion briefly summarizes the paper and proposes directions for future work.

II. Related Work

Since performance is highly relevant for distributed simulations, there exist several proposed balancing approaches. Basically, the proposed solutions aim to speed up simulation processing pace by minimizing communication delays or improving utilization of resources. However, these schemes present drawbacks that motivated the design of a self-adaptive balancing scheme.

Some balancing schemes observe look-ahead or the communication rate in simulations to decrease their communication delays. The analysis of look-ahead evidences the simulation dependencies that might be slowing down the simulation [2] [3]. The study of communication rate indicates the delays they introduce in the simulation, which can be performed statically

† 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.

[4] [5] or during simulation run-time [6] [7] [8] [9] [10] [11] [12]. Nevertheless, minimizing delays in communication dependencies is not enough to improve simulation performance.

Other balancing schemes are focused on organizing the computational load of simulations. The simulation-centred solutions analyze the speed of each simulation entity to increase the execution pace [13] [14] [15]. The resource-centred approaches consider the CPU consumption of simulations to maximize resource's utilization [16] [17] [18] [19] [20]. In order to consider resource heterogeneity, causality inconsistencies, and external background load, a scheme has been proposed in [21]. Due to the issues regarding the approach's centralized design, a distributed scheme was proposed in [22]. This distributed technique produces excessive number of migrations to achieve load balance because of the inter-relations between its balancing elements, varied resource capacities, and load oscillations. Thus, an adaptive approach is proposed to modify the load balancing parameters and decrease the number of precipitated migrations to improve performance and reduce the chances of failures and rollbacks.

III. Distributed Load Balancing Scheme

As described in [22], a hierarchical, distributed architecture is introduced to decentralize the redistribution algorithm. In this architecture, the local balancing is performed by collecting data individually, and the redistribution phase is performed in a distributed manner. Based on this distributed scheme, extra components are added to introduce self-adaptation, and slight modifications are applied in the balancing algorithms.

A. Architecture

As depicted in Figure 1, the Cluster Load Balancer (CLB) organizes all the process of the balancing scheme by triggering data gathering, detecting imbalances, re-organizing the load, and performing migration moves. In order to collect information about the resources and federates, the CLB accesses Monitoring Interfaces and its Local Load Balancers (LLBs). A Monitoring Interface facilitates data request to Monitoring Information Services, which is provided through Grid services. Relying the monitoring of resources on Grid computing, which includes a resource sharing system that organizes of resources, individuals, and institutions [23], Globus Toolkit [24] is used for providing Grid MDS services. The collected information comprises the processing queue of resources. Before requested data is delivered, filtering is employed to remove data that misleads the detection of imbalances.

A LLB is placed in every resource in order to intermediate requests of load information and migration calls from a CLB. The requested information regards federates' CPU consumption, and this information is collected from each federate by accessing the Federate Balancing Interfaces (FBI) through the Local Monitor Interface (LMI). A LLB also dispatches migration calls to their respective Migration Manager (MM). Likewise the LMI, an instance MM works for each federate in a resource.

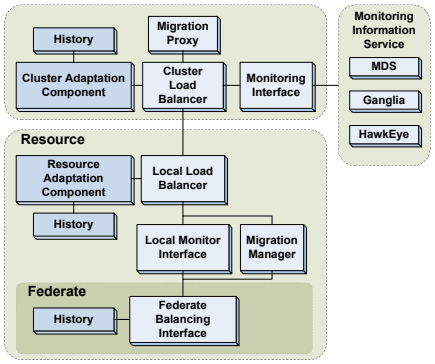

Fig. 1: Dynamic Balancing Scheme's Architecture

As introduced in [25] and [26], the federate migration is performed in two phases in order to minimize the migration latency. With assistance of Grid services, the first part of this two-phase migration transfers configuration files and initiates the federate remotely. The second part of the migration procedure sends the federate's execution state and its incoming messages through peer-to-peer data transfers, and a Migration Proxy intermediates the data transmissions for migrations performed between remote parts.

Adaptation components are added in the load balancing's architecture to observe the balancing actions. Every incoming or outgoing migration is registered by History component, which are placed in federates, resources, and CLBs. These elements store migration frequency data, and they contain a queue that is restricted to a certain size (H). Cluster Adaptation Component (CAC) works together with the CLB by collecting migration histories for determining the proper adjustments. A Resource Adaptation Component (RAC) is placed in each resource, aggregates migration history from the local resource History component, and produces a migration ratio and federate migration frequencies.

B. Redistribution Algorithm

As detailed in Algorithm 1, the load redistribution is performed in inter-domain and local scopes by a CLB. For both scopes, monitoring data is collected from the resources and federates. This data is processed to detect load imbalances and redistribute load properly. The load changes are created through local pair-match evaluations based on a minimum resource load (min'), as shown in Algorithm 2.

After determining local migrations, an inter-domain load redistribution is performed based on its success rate in the previous balancing cycle. For this redistribution, a set of overloaded resources is selected according to $selectionParam'$. Upon receiving the resources candidates for redistribution, the neighbour CLB evaluates them as described in Algorithm 3. The inter-domain pair-match procedure is based on a minimum resource load (min'), which determines if a federate can be transferred to a resource. Because these three thresholds regulate the balancing responsiveness, they receive adjustments in every balancing cycle.

Algorithm 1 Distributed Dynamic Load Balancing Algorithm

```
loop
    loads ⇐ query_MDS()
    spec_loads ⇐ request_LLBs()
    mig_moves ⇐ redistribute_local(loads, spec_loads)
    if mig_moves = ∅ then
        data_neighbours ⇐ request_Neighbour_Load_Data()
    else
        if relFactor ≥ random_number(1, 100) then
            send_migration_moves_to_caller(migration_moves)
        else
            data_neighbours ⇐ ∅
        end if
    end if
    neighbours ⇐ identify_Neighbour_Less_Load()
    if neigbours! = ∅ then
        overloaded_resources ⇐ select(firstNeighbour)
        federates ⇐ select(spec_loads, overloaded)
    else
        selectionParam′ ⇐ selectionParam ∗ selectionAdj
        federates ⇐ select(spec_loads, overloaded_rec, selectionParam′)
    end if
    send_to_neighbour(overloaded_resources, federates)
    migration_moves ⇐ wait_for_migration_moves()
    send_migration_moves(migration_moves)
    adjust(relFactor, overloaded_resources, migration_moves)
    adaptation()
    wait( Δ )
end loop
```

Algorithm 2 Local Pair-Match Evaluation Algorithm

```
Require: src_rsc, dst_rsc
  min′ ⇐ min ∗ minLoadAdj
  if dst_rsc < min′ then
      if number_fed(src_rsc) > 1 then
          create_migration_move(src_rsc, dst_rsc)
      else if number_fed(src_rsc) = 1 & src_rsc > (min′ ∗ φ) then
          create_migration_move(src_rsc, dst_rsc)
      end if
  else if (dst_rsc − src_rsc) > (min′ ∗ δ) then
      if number_fed(src_rsc) > 1 then
          create_migration_move(src_rsc, dst_rsc)
      else if number_fed(src_rsc) = 1 & (dst_rsc − src_rsc) > (min ∗ φ)
      then
          create_migration_move(src_rsc, dst_rsc)
      end if
  end if
```

Algorithm 3 Inter-Domain Pair-Match Evaluation Algorithm

```
Require: int_rsc, ext_rsc
  min′ ⇐ min ∗ acceptanceAdj
  if int_rsc < min′ then
      create_migration_move(int_rsc, ext_rsc)
  else if (dst_rsc − src_rsc) > (min′ ∗ δ) then
      create_migration_move(int_rsc, ext_rsc)
  end if
```

IV. Self-Adaptation Technique

The self-adaptation is provided in three phases: data collection, analysis, and execution of adjustments. The adaptation occurs in each cluster of resources managed by a CLB, and it needs to collect information that reflects the consequences of re-distributions. Consequently, the recent migration past is analyzed to adjust the balancing thresholds. The proposed adaptation technique employs number of migrations and time as the main metrics. The number of migrations is registered according to the time in which they occur, and they are restricted to a history (H). The history is used to limit the amount of stored and exchanged migration information; it also determines the relevance of the recent migration moves when compared with older moves.

Triggered at the end of each balancing cycle, the adaptation components collect federates' migration frequency, resources' migration ratio, and a CLB's migration ratio. The migration frequency shows how often federates are being moved between shared resources. The migration ratio comprises the division of incoming migrations by the outgoing migrations in a resource. The adaptation analysis is realized in collective and individual scopes. The collective scope observes the migration frequencies and the migration ratios as whole to determine balancing inefficiencies. In the individual scope, each federate and resource is evaluated regardless the rest of the system. The adjustment modifications obtained from the adaptation analysis are applied on the parameters that configure the balancing system.

A. Data Gathering

Acting responsively, the adaptation system needs to constantly collect data to mould the balancing system according to actual load redistribution needs. Thus, the adaptation system gathers migration information together with the balancing scheme's monitoring data.

Placed in a resource, each Federate History (FH) registers the last H migration moves of a federate in a list, which is provided to the FBI. Once the information from federates in a resource is gathered by a LLB, it is processed by a Resource Adaptation Component (RAC) and re-attached to the respective federate's monitoring information. Thus, each RAC keeps track of a set of federates' migration history. The migration history is divided in H elements that register (0 or 1) if a federate went through migration in a balancing cycle.

Responsible for a resource, a Resource History (RH) saves the last H incoming and outgoing federate migrations. A RAC requests to RH the register of its H migration moves in the past balancing cycles and then processes the migration information to produce a migration ration, which is sent to a CLB. Thus, a resource migration history stores the migration ratios in the past $H/2$ balancing cycles. Since the balancing system performs only one migration per balancing cycle for a resource, this history list contains only 1s and 0s.

A Cluster History (CH) stores the last H incoming and outgoing federate migrations for each CLB. A CAC requests CH data regarding the H last migration moves. After receiving all the monitoring information from its LLBs, the CLB also extracts the federate frequencies and the resources' ratios. At the end, the adaptation component processes the CLB migration history to generated a ratio, the resource migration ratios to result in a cluster-resource migration ratio, and the individual federate frequencies to produce a general cluster migration frequency. Likewise RH, a CH registers the migration ratios in the past $H/2$ balancing cycles.

B. Calculation of Frequencies and Ratios

The first part of interpreting the collected data is processing it and calculating the migration frequencies and ratios. All

the calculations employ migration history (H) as the main determinant element. In the adaptation analysis, time is fundamental to retrieve ratios and frequencies since they present relevance for the time that they were obtained. Thus, the history comprises this time need, and it is completely based on balancing cycles since they conduct the pace in which migrations are produced in the system.

The calculations are first realized in each resource by an RAC. Basically, this component obtains federates' migration frequencies and resource's migration ratios. The migration frequency of each federate is obtained through the calculation of a weighted mean of its history list elements, as shown in Formula 1. The weights are disposed on the history elements to provide priority to the most recent migration moves. As a result, each i element receives w_i weight of $H - i$ in the calculation, considering H as the size of the history.

$$\overline{freq} = (\sum_{i}^{H} f_i \times w_i) / \sum_{i}^{H} w_i \tag{1}$$

Each resource's migration ratio is calculated by dividing its weighted mean of $H/2$ incoming migrations by its weighted mean of $H/2$ outgoing migrations. The resource's migration weighted mean is obtained according to Formula 2. The mean is considered only if the sum (n_{sum}) of the first n elements in the history list is larger than zero.

$$\overline{avg}_{rsc} = \begin{cases} (\sum_{i}^{H} x_i \times (H-i)) / \sum_{i}^{H} i & , n_{sum} > 0 \\ 0 & , n_{sum} = 0 \end{cases} \tag{2}$$

The parameter n that restricts the first elements of a history list is determined by Formula 3. In the formula, an n is selected from a pre-defined condition, which defines the minimum frequency for considering frequency calculations. In this work a minimum resource migration frequency is assumed as the frequency represented by the migration move pattern that conforms with the regular expression $(100)^+$. Observing this history list as an arithmetic progression, Formula 4 represents this minimum frequency. Therefore, with x defined, the sum of elements in such sparse progression is calculated in Formula 3. The calculated sum is compared with the sum of the first n elements in the history containing non-zero elements.

$$\sum_{i}^{n} h_i \leq (H + x) \times (\lceil H/3 \rceil + 1) \times 1/2 \tag{3}$$

$$x = H + \lceil H/3 \rceil \times (-3) \tag{4}$$

$$\overline{freq}_{avg} = \sum_{j}^{R} \sum_{i}^{Fj} freq_i / Fj / R \tag{5}$$

After the frequencies and ratios computed in each resource are retrieved by a CLB, a CAC merges all the information in a second step of calculations. The component initially computes the inter-domain migration ratio by also employing Formulas 2, 3, and 4, and then it computes the the ratios' and frequencies' average. An arithmetic mean is used to obtain the migration ratios' average in a cluster while the frequencies' average is determined by Formula 5. Assuming R as the number of resources in a cluster and F_j the number of federates in a specific resource, the CAC calculates the frequency mean of a cluster based on the arithmetic mean of frequencies in each resource.

C. Detection

In this part of the adaptation process, the collected data is analyzed to identify irregularities in the load balancing system that might lead it to malfunction. All the gathered data, as well as the computed averages, are used to determine adjustments in load balancing thresholds or to flag some specific federates and/or resources. Modifications in the thresholds are defined from the analysis of collective ratios and frequencies, i. e., metrics that represent the load balancing system's behaviour for the entire cluster. Analysis of individual ratios and frequencies indicates that only certain resources and federates present some particular characteristic, such as cyclic oscillations of federate and external background load, which drive the load balancing system to react improperly. Thus, in this detection step, a pre-processing is applied on the collected data sample and comparison analyses are performed.

Initially, the CAC requires a pre-processing for the averages in order to represent the load balancing system's efficiency more accurately. These average adjustments regard the relations between the number of federates and resources in a cluster. Because the frequency average can be unrepresentative of the real load redistribution effects when the number of resources is largely inferior than the number of simulation federates, the frequency average is adjusted by multiplying the current average by the ratio $resources/federates$ given by Formula 6. In the formula, the average is increased in an inversely proportional rate to the ratio between the number of federates and the number of resources. This adjustment improves the perception of migration frequency in a cluster because a misleading highly reactive balancing behaviour can be hidden by a low average as a reflect of numerous federates running on the shared resources, which always leads to a interpretation of just sparse migrations in the system.

$$f_{adj}(r, f) = \begin{cases} (2 - r/f) & , r/f \leq 1 \\ 1 & , r/f > 1 \end{cases} \tag{6}$$

The migration ratio average also needs to be adjusted, introducing awareness of the inter-domain migration ratio and the relation between number of federates and number of resources, as described in Formula 7. In this case, the opposite case of the relation between the number of federates and the number of resources influences the interpretation of the migration ratio. In order to introduce awareness of such a situation, the migration ratio average is modified by ratio $federates/resources$, which is computed in Formula 8. Moreover, because resources can experience intra-domain and inter-domain migrations, their migration ratio may contain some migrations that belong to inter-domain redistribution; thus, an adjustment regarding the inter-domain migration moves is provided, which is represented by β in Formula 9. The formula considers the inter-domain incoming and outgoing migration frequencies that need to be excluded from the migration ratio average. For each balancing cycle (i) in the history list ($H/2$), the number of in migrations is summed to the out migrations. This sum is divided by the current number of resources in order to obtain the ratio of inter-domain migrations per resource. An weighted mean is computed with

Algorithm 4 Distributed Redistribution Adaptation Algorithm

Require: $ratio_{clb}, clusterINMig, clusterOUTMig$
  if $ratio_{clb} < t$ & $ratio_{clb} > 1/t$ **then**
    if $ratio_{clb} \geq 1$ **then**
      $ajdust\_Acceptance\_Threshold()$
    else if $ratio_{clb} < 1$ **then**
      $adjust\_Overloaded\_List\_Threshold()$
    end if
  else
    if $clusterINMig == 0$ **then**
      if $past\_Increase\_IN$ **then**
        $decay\_IN\_factor()$
      end if
    else if $clusterOUTMig == 0$ **then**
      if $past\_Increase\_OUT$ **then**
        $decay\_OUT\_factor()$
      end if
    end if
  end if

all this ratio values in the history list. At the end, this inter-domain ratio shows the amount of inter-domain migrations that have been performed by the load balancing system in each resource. Consequently, the complement of this ratio represents the amount of intra-domain migrations in each resource, and it is used to specify the migration ratio value to the local scope.

$$avg_{rsc}^{adj} = avg_{rsc} \times \alpha \times \beta \tag{7}$$

$$\alpha = \begin{cases} (2 - f/r) & , f/r \leq 1 \\ 1 & , r/f > 1 \end{cases} \tag{8}$$

$$\beta = 1 - \frac{\sum_i^{H/2}((in_i + out_i)/rsc) \times w_i}{\sum_i^{H/2} w_i} \tag{9}$$

At this point in the analysis, a CAC contains the cluster migration ratio, the resource migration ratio average, the federate migration frequency average, a list of resource migration ratios, and a list of federate migration frequencies. All these metrics are used to determine the current adaptation, which comprises adjustments to both inter-domain and local (intra-domain) load balancing. The cluster migration ratio is the main parameter to determine the inter-domain balancing adjustment, as described in Algorithm 4. According to the algorithm, the ratio ($ratio_{clb}$) is evaluated; if it presents value in the interval delimited by t and $1/t$ thresholds, adjustments are calculated to modify the balancing parameters. The threshold t represents the difference between incoming and outgoing migrations, and it determines how selective the adaptation is with the load balancing responsiveness. Furthermore, current cluster incoming ($clusterINMig$) and outgoing ($clusterOUTMig$) migration frequencies are used to identify a decay for the last balancing parameter adjustment when no new adaptations are needed for the current balancing efficiency.

For analyzing the intra-domain balancing efficiency, resource ratio average and federate frequency average are employed in the adaptation analysis, as described in Algorithm 5. Initially CAC assess the migration ratio by checking if the ratio lies in the interval delimited by t_{rsc} and $1/t_{rsc}$. Likewise in the case of the inter-domain adjustment, t_{rsc} is pre-defined and corresponds to the difference allowed between the number of incoming and outgoing intra-cluster migrations. If a need of adaption is identified, the respective adjustment is calculated.

Algorithm 5 Local Redistribution Adaptation Algorithm

Require: $ratio_{avg}^{rsc}, freq_{avg}$
  if $ratio_{avg}^{rsc} < t_{rsc}$ & $ratio_{avg}^{rsc} > 1/t_{rsc}$ **then**
    if $ratio_{avg}^{rsc} \geq 1$ **then**
      $adj_{rsc} = calc\_rsc\_ajdust\_factor1()$
    else if $ratio_{avg}^{rsc} < 1$ **then**
      $adj_{rsc} = calc\_rsc\_ajdust\_factor2()$
    end if
  end if
  if $freq_{avg} > t_{freq}$ **then**
    $adj_{freq} = calc\_freq\_adjust\_factor()$
  end if
  if $adj_{rsc} > 0$ || $adj_{freq} > 0$ **then**
    if $adj_{rsc} == 0$ **then**
      $adj_{final} = adj_{freq}$
    else if $adj_{freq} == 0$ **then**
      $adj_{final} = adj_{rsc}$
    else if $adj_{rsc} > adr_{freq}$ **then**
      $adj_{final} = calc\_local\_adjustment1()$
    else
      $adj_{final} = calc\_local\_adjustment2()$
    end if
    $calc\_decay\_fraction()$
  else
    $decay\_adjust\_factor()$
    $adj_{final} = 0$
    $resource\_list = identify\_unstable\_rsc()$
    $federate\_list = identify\_unstable\_fed()$
  end if

CAC also analyzes the frequency average by evaluating if the general migration frequency in a cluster exceeds a threshold, t_{freq}. The threshold represents minimum migration frequency considered normal by the adaptation algorithm and is obtained through Formulas 10 and 11. This minimum frequency is pre-defined in the system and presents a sparse migration pattern that can be represented by the following regular expression: $(1001)^+$. To obtain this minimum allowed migration frequency, it is assumed that a element i in the migration history list receives weight $H - i$ for the weighted mean calculation for a federate's migration frequencies, which is described in Formula 1. According to this assigned weight, the second formula determines the number of elements (n') for the pre-defined sparse migration frequency. Then, the first formula defines the sum (S') of elements in the progression according to the number of elements (n'). CAC uses S' as the threshold t_{freq} to identify the need of adaptation.

$$S' = \frac{3n'^2 - n'}{2} \tag{10}$$

$$n' = \lfloor \frac{a_n + 2}{3} \rfloor \tag{11}$$

After both adjustments related to frequency average and migration ratio average are obtained, CAC checks the need for adaptation ($adj_{rsc} > 0$ or $adj_{freq} > 0$). If both values are equal to zero, the final adjustment is not computed, but a decay is applied to the last adjustment, and a search for unstable resources and federates is realized. For this search, each federate and resource is evaluated individually. The evaluations consist in comparisons with t_{freq} for determining unstable federates and comparisons with t_{rsc} and $1/t_{rsc}$ for identifying unstable resources. These elements are marked in order not to be considered in the next $H/2$ load balancing cycles.

The marking of unstable federates and resources prevents the load balancing system to perform load redistribution based on them. Some simulation federates or shared resources

produce unstable load due to cyclic load changes, excessive simulation load, or heterogeneity of resources. A federate or an external process might oscillate its load dynamically in certain frequency that makes the balancing system reorganize the load often. Also, the excessive load or the heterogeneity of resources might cause the balancing system to move load constantly due to the creation of new overloaded resources through the redistribution procedure. Thus, such unstable elements are exceptional cases in the system, which require additional handling, and being marked enables a decrease of improper migrations.

D. Adjustment

After needed adaptations are detected, adjustments are calculated according to the amount of divergence that the load balancing presents when compared with its expected behaviour. At the end of the process, the adaptation involves the modifications in three load balancing parameters: overloaded list selection, inter-domain acceptance, and minimum resource load. These parameters are directly related to the distributed load balancing scheme, so modifying them results in an increase or decrease of migrations. Also, all the balancing parameters initially present small values, enabling high responsiveness and a large number of migrations. These migrations are then gradually reduced if adjustments are detected by the adaptation system.

The overloaded list selection parameter determines the number of resources that are selected for the inter-domain redistribution analysis, as described in Algorithm 1. This balancing parameter is directly modified by $selectionAdj$, which contains the respective adaptation value. This value is calculated based on the range $1..1/t_{clb}$, which corresponds to the range for a larger amount of outgoing migrations. A migration ratio ($ratio_{clb}$) in this range means that outgoing migrations are required to balance the system's load, but such migrations are produced excessively. Consequently, the parameter's adjustment proportionally increases to the proximity of $ratio_{clb}$ to 1, as computed with the following formula: $ratio_{clb} \times (ratio_{clb} - 1/t_{clb})/(1 - 1/t_{clb})$.

The inter-domain acceptance is responsible for matching external load with internal resources of a cluster, as delineated in Algorithm 3. The parameter is updated according to $acceptanceAdj$, which receives the adaptation value determined in the last balancing cycle. The value is obtained through the following formula: $ratio_{clb} \times (t_{clb} - ratio_{clb})/(t_{clb} - 1)$. Likewise the calculation of the adjustment for the overloaded list selection parameter, the formula considers the distance from $ratio_{clb}$ to 1: a closer $ratio_{clb}$ to 1 reflects in a larger adjustment for the acceptance. However, in this case, the range $t_{clb}..1$ is observed since this adjustment is focused on the number of incoming migrations. Thus, when $ratio_{clb}$ is larger than 1, the balancing system is receiving load, but if this ratio is in the range, the CLB is accepting load excessively.

As presented in Algorithm 2, the minimum resource load is used in the intra-domain re-partitioning procedure to determine migration pairs through load comparisons. Modifying this balancing parameter considerably affects the pair-match algorithm. In Formula 12, the local adjustment related to the resource ratio is calculated by determining the proximity of r_{rsc} to 1, i. e., a high incidence of unnecessary migration moves. In this case, the migration ratio adjustment is proportional to this proximity. In Formula 13, the migration frequency adjustment is calculated according to the migration frequency, which ranges between 1 and t_{freq}. In the range, 1 means the maximum migration frequency, and t_{freq} the minimum acceptable frequency. A closer value of $freq$ to 1 requires a larger, proportional adjustment, what is provided in the formula. The final local adjustment (adj_{final}) is calculated with both ratio and frequency adjustments. According to Algorithm 5, method $calc\_local\_adjustment1()$ uses Formula 14 to calculate the final adjustment if adj_{rsc} is larger than adj_{freq}. On the other hand, if adj_{rsc} is smaller of equal to adj_{freq}, method $calc\_local\_adjustment2()$ employs Formula 15 to compute the final adjustment value. For both formulas, more weight is given in the sum to the parameter that denotes more drastic modifications in the load balancing system.

$$adj_{rsc} = \begin{cases} (r_{rsc} - t_{rsc})/(1 - t_{rsc}) & , r_{rsc} \geq 1 \\ (t_{rsc} - r_{rsc})/(t_{rsc} - 1) & , r_{rsc} < 1 \end{cases} \quad (12)$$

$$adj_{freq} = \frac{freq - t_{freq}}{1 - t_{freq}} \quad (13)$$

$$adj_{final} = adj_{rsc} \times (1 - \frac{adj_{freq}}{adj_{rsc}}) + adj_{freq} \times (\frac{adj_{freq}}{adj_{rsc}}) \quad (14)$$

$$adj_{final} = adj_{freq} \times (1 - \frac{adj_{rsc}}{adj_{freq}}) + adj_{rsc} \times (\frac{adj_{rsc}}{adj_{freq}}) \quad (15)$$

V. Experimental Results

The proposed self-adaptive load balancing system is evaluated through a series of simulation experiments that compares the centralized and distributed balancing schemes with the adaptation technique applied on the distributed scheme. These experiments are deployed on a environment composed of two clusters of computing servers and a fast-ethernet link connecting the cluster's management nodes. One cluster consists of 32 nodes interconnected through a Gb Ethernet network; each node presents a Core 2 Duo 3.4 GHz Intel Xeon CPU and 2 GB of RAM. The other cluster comprises 24 nodes interconnected through a Myrinet optical network that allows data transmission up to 2 Gb/s; each node contains a Quadicore 2.40GHz Intel Xeon CPU and 8 GB of RAM. In all environment, Linux operating system and the Globus Toolkit 4.2.1 were installed, and the HLA platform with RTI version 1.3 was used to coordinate the experimental simulations.

As the initial common deployment of simulation for each experiment, the elements were evenly distributed on the 55 shared computing servers. The HLA RTI executive was placed on a dedicated computing node of the environment. The balancing system had its elements distributed on all the shared resources, considering that each CLB was positioned on a cluster management node.

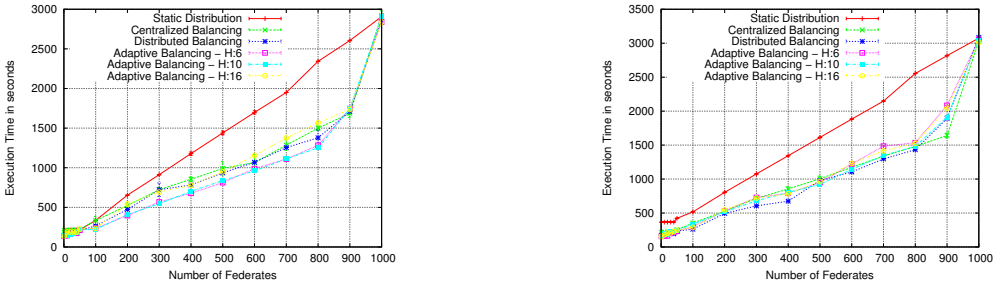

Fig. 2: Performance Analysis for an Increasing Number of Federates and External Background Load

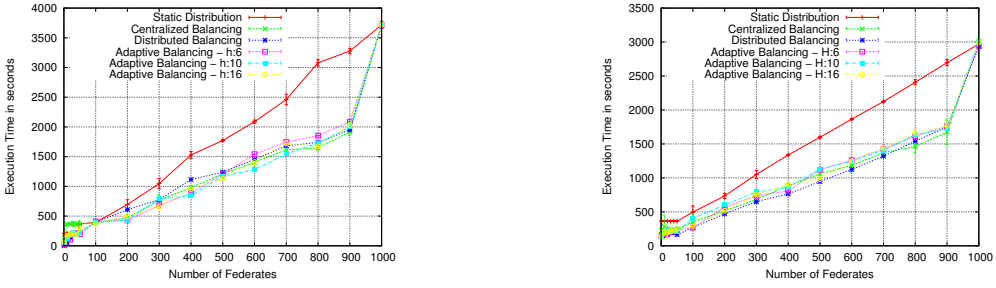

Fig. 3: Performance Analysis for an Increasing Number of Federates and External Process with Dynamic Load Changes

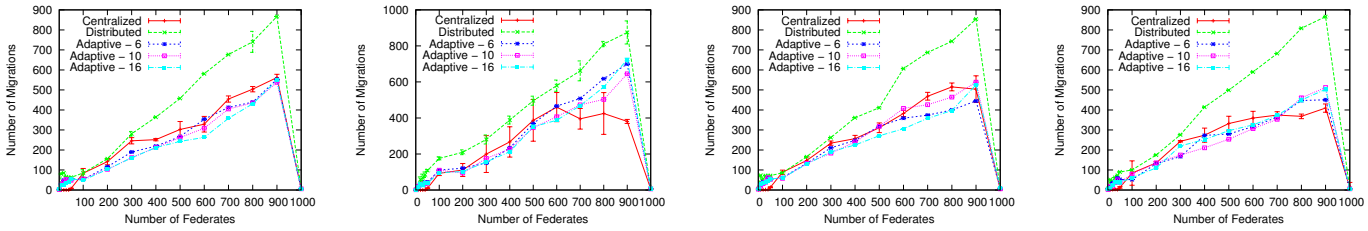

Fig. 4: Comparative Studies - Number of Migrations versus Comparison of Number of Migrations

The simulation scenario used in the experiments consisted in time-stepped homogeneous simulations that performed training operations coordinated in a two-dimension routing space. In each simulation, 1 to 1000 federates periodically calculated the movement of tanks for 100 time steps. The calculation of movements included computing intensive tasks, publishing a tank's position, and subscribing to a interest space. Different history sizes (6, 10, and 16) for the adaptation system were used in the experiments, and t_{clb} and t_{rsc} received value 3.

In this set of experiments, the four experimental scenarios presented in the previous work [22] were realized to show the efficiency of the proposed scheme: static scenarios with increasing number of federates 2a and external load 2b, and dynamic scenarios with increasing number federates presenting run-time load changes 3a and moving external load 3b.

As presented in all experiments, the self-adaptive solution was able to provide performance improvement as the distributed balancing scheme did. However, when observing the number of migrations, as depicted in graphs of Figure 4, the adaptive solution, with any history size, was able to reduce the number of migrations for the distributed approach. For any case, the number of migrations drastically reduced for experiments with 1000 federates due to the saturation of the distributed system. A deeper analysis showed that history size 16 decreased the balancing responsiveness due to high influence of migration past on the recent moves, as describe in the small number migrations in Figure 4a and in the higher simulation time in Figure 2a. On the other hand, history size 6 led the balancing to an unstable behaviour due to the high weight to recent past, as delineated in higher simulation times in Figures 2b and 3a and in the increase in instability in Figures 4b and 4c. The adaptive solution with history size 10 showed the most stable adjustments in all the experiments since it best adapted the balancing system for the migration latencies and cyclic load oscillations.

The previous experiments showed that performance gain was achieved even with large number of migrations. This particular characteristic resulted from short migration delays, but experiments with large number of federates showed an 9.7-second migration latency average. This latency was hidden by simultaneous migrations and by highly intensive computing load that overlapped migration latencies. Consequently, in the

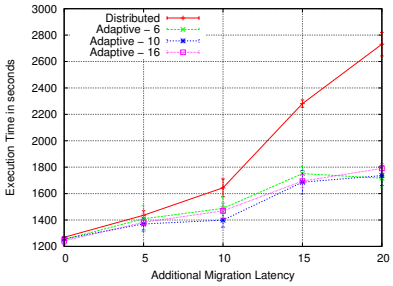

Fig. 5: Analysis with Increasing Migration Latency

experiments in Figure 5, migration time was observed when comparing the distributed load balancing scheme with the proposed self-adaptive solution. Slower migrations can be caused in real simulation by communication distances and federate sate size, but in these experiments, a delay was intentionally introduce in the second phase of federate migrations. For such experiments, a scenario with static load composed of 500 federates was used. The curves showed that the numerous migrations caused by the non-adaptive scheme influenced the performance negatively when the additional delay exceeded 5 seconds since part of these larger delays was not covered by the long simulation time. Thus, simulations more sensitive to migration latency benefited from the adaptive approach.

VI. Conclusion

In this paper, a self-adaptive distributed balancing scheme for large-scale HLA-based simulations is presented. The adaptation technique aims to decrease the number of unnecessary balancing migrations while keeping simulation performance gain similar to or better than the distributed balancing system. The adaptation generates adjustments for the balancing system's parameters to control the sytem's responsiveness. Such adjustments are based on the incoming and outgoing migration moves performed. The experiments showed that the adaptation technique considerably reduced the number of federate migrations without hampering the balancing effectiveness. Moreover, as observed in the experiments, migration time and cyclic load changes substantially affected the balancing. Thus, as future work, the scheme's scope will be extend to support heterogeneous simulations, and the influence of migration latency and load on balancing will be further studied.

References

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

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

[3] M. Y. H. Low, "Dynamic load-balancing for bsp time warp," in *Proc. of the Annual Simulation Symposium*. IEEE Computer Society, 2002, pp. 267–274.

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

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

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

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

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

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

[10] E. E. Ajaltouni, M. Zhang, A. Boukerche, and R. E. D. Grande, "An adaptive dynamic load balancing technique for grid-based large scale distributed simulations," *Journal of Interconnection Networks*, vol. 10, no. 4, pp. 391–419, 2009.

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

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

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

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

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

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

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

[18] W. Cai, S. J. Turner, and H. Zhao, "A load management system for running hla-based distributed simulations over the grid," in *Proc. of the International Workshop on Distributed Simulation and Real-Time Applications*. IEEE Computer Society, 2002, pp. 7–14.

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

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

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

[22] R. E. D. Grande and A. Boukerche, "A dynamic, distributed, hierarchical load repartitioning for hla-based simulations on large-scale environments," in *Proc. of the International European Conference on Parallel Processing*, 2010, to Appear.

[23] 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.

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

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

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