Migration Delay Awareness in a Self-Adaptive Balancing Scheme for HLA-based Simulations

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

***Abstract*—Load balancing is a vital mechanism for improving the performance of distributed simulations or even for enabling their execution. A balancing technique has been designed in order to provide a balancing scheme for HLA-based simulations on non-dedicated resources. However, this technique lacks efficiency by producing large amounts of unnecessary federate migrations, so a self-adaptive mechanism has been introduced in the technique in order to correct its balancing responsiveness. As a drawback, the self-adaptation technique assumes that only the frequency of federate migrations represents the balancing efficiency. This leads the scheme to present static parameters regardless of the conditions of the environment, which in turn can restrict the balancing response to imbalances. Thus, awareness to migration delays is inserted into the self-adaptive balancing scheme in order that more precise and more realistic analysis of balancing efficiency can be enabled. Experiments have been conducted to show the performance gain of the proposed scheme when compared to the distributed and self-adaptive load balancing systems.**

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

I. Introduction

Distributed simulations have been receiving increased attention because of the need to design more complex platforms for executing virtual environments. These distributed simulations, which are specifically designed and coordinated using the HLA framework, require a mechanism to prevent load imbalances in order to enable such simulations on distribution resources or provide reasonable execution performance. The imbalances that might occur on the simulation load distributed are caused by the heterogeneity of resources, the existence of external load on non-dedicated resources, or dynamic simulation load oscillations. Since static load balancing cannot offer a load distribution solution for simulations and external processes that present load that changes dynamically and unpredictability, a dynamic load balancing scheme is required.

The HLA framework [1] is composed of a set of rules, interface specifications, and object model templates that intend on providing the proper coordination of large-scale distributed simulation executions. This design specification intends on avoiding simulation inconsistencies and providing re-usability of components as well as the interoperability of simulation entities. In an every HLA-based simulation, which is termed federation, a set of simulation entities, termed federates, run and interact through the access to the HLA Run-Time Infrastructure (RTI) management services. The RTI services are strictly designed to coordinate only the simulation execution. As a result of this, the framework is completely unaware of the load imbalances caused by the misplacement of the simulation load or the dynamic oscillations on the overall load.

Many load redistribution schemes have been devised in order for distributed simulations to cope with the load balancing issues. Most of these schemes are simulation application oriented or otherwise present some limitations that impede them from being applied to HLA-based simulations. Consequently, a distributed balancing scheme has been designed for such HLA simulations, but this solution lacks efficiency by generating a large amount of unnecessary federate migrations to provide simulation performance gain. A self-adaptive balancing system is built in an attempt to improve the efficiency by which the simulation load was dynamically reorganized by this distributed scheme. In this system, efficiency is fundamentally based on the frequency of federate migrations in certain monitoring intervals; however, this assumption of balancing efficiency is misleading if migration delays are negligible.

Therefore, migration delay awareness is introduced into the self-adaptive balancing scheme in order to reduce the number of migrations according to the recently observed delays from the load reallocation. The proposed balancing scheme is based on the self-adaptive scheme by which to employ its method of analyzing balancing efficiency; consequently, the scheme is also based on the distributed balancing scheme in order to redistribute simulation load. The proposed solution introduces the monitoring of federate migrations to obtain accurate migration times and subsequently inserts dynamic modifications of self-adaptation parameters to redefine balancing efficiency.

The remainder of this paper is organized as follows. In section 2, the related work is introduced by describing the previous balancing approaches for distributed simulations. In section 3, the architecture and functioning of the proposed scheme is delineated. In section 4, the proposed technique employed to introduce migration delay awareness is detailed. In section 5, experimental scenarios used for the performance evaluation are described, and the results are explained. Finally, the conclusion and directions for future work are presented.

II. Related Work

Many load balancing schemes have been designed with the aim to improve execution performance improvement of distributed simulations and consequently reducing simulation

time. Most of the devised solutions present dynamic balancing mechanisms that enable the detection and response to load imbalances during run-time. The schemes mainly focus on aspects delimited by a simulation's internal dependencies or by computational load imbalances. The analysis of internal dependencies of distributed simulations indirectly shows the existence of delays that increase simulation time. On the other hand, the observation of computational load characteristics enables the detection of aspects that directly influence performance.

The analysis of simulation delays produced by inter-dependencies between simulation entities uses look-ahead and communication rate metrics to determine the cause of waiting time in simulation entities. The look-ahead indirectly exposes the dependencies among simulation entities [2] [3]. Communication rate and latency, on the other hand, directly indicate the influence of dependencies between simulation elements [4], [5], [6], [7], [8], [9]. In order to provide a more accurate analysis of such dependencies, some balancing schemes consider the proximity of simulation entities and the network topology in their algorithms [10], [11]. This balancing approach can produce performance gains for distributed simulations, but it cannot prevent the performance loss caused by computational load imbalances on distributed, shared resources.

The balancing schemes based on computational load aspects adopt approaches that can be classified as either simulation-centred or resource-centred. For the simulation-centred approach, the balancing schemes observe the simulation's current characteristics in order to determine simulation performance [12], [13], [14], [15]. For the resource-centred approach, direct monitoring of a distributed resources' current load status is applied on the the balancing schemes [16], [17], [18], [19], [20], [21], [22], [7], [8], [23]. All above described balancing schemes either contain some limitation that do not allow them to deal with resource heterogeneity and external background load or are designed for a specific type of simulation and cannot be generalized and applied to HLA-based simulations.

In order to introduce a computational load balancing system that can overcome the limitations of the previous solutions, a distributed balancing scheme has been devised [24]. This scheme effectively detects and responds to load imbalances but lacks efficiency in so far that it generates a large amount of unnecessary migrations. With the intention of improving the balancing efficiency of this scheme, a self-adaptive technique is introduced [25] to constantly analyze the balancing decisions through federate migrations. Despite improving efficiency, this self-adaptive balancing scheme does not consider migration latency, which is vital for defining balancing efficiency correctly. Therefore, a balancing scheme that employs a migration-delay aware self-adaptation technique is proposed.

III. Migration-Aware Self-Adaptive Balancing Scheme

The awareness of migration delay is introduced in the self-adaptive scheme [25] by adding algorithms threshold recalculations to the adaptation based on estimated migration times. Consequently, the proposed scheme presents a decentralized load redistribution algorithm and a hierarchical, distributed organization of architectural components.

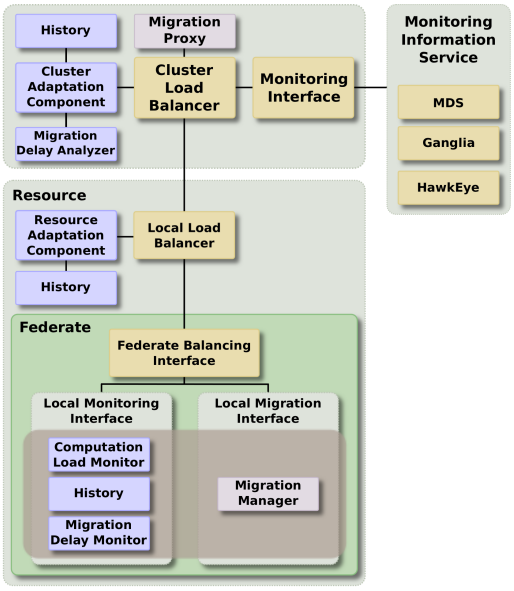


Fig. 1: General Architecture of the Load Balancing System

A. Load Balancing Architecture

As described in Figure 1, the Cluster Load Balancer (CLB) is responsible for organizing the local and inter-domain load redistribution algorithms. A CLB accesses a Monitoring Interface to obtain the resources' load status data from Monitoring Information Services. It also accesses a Local Load Balancer (LLB) to obtain federates' load status data and forward migration calls. The Monitoring Information Service component can represent many monitoring systems, but Grid Web MDS [26] is used in the scheme in order to provide accurate CPU load status regarding resources.

A LLB works as an interface between the simulation federates in each resource and the balancing system (CLBs). Through a Federate Balancing Interface (FBI) that is instantiated for each federate, the information gained from monitoring, such as computational load, migration history, and migration delay, is gathered by accessing a Local Monitoring Interface. Migration calls are also forwarded to a federate by a FBI component through a Local Migration Interface, which triggers migrations through a Migration Manager (MM).

A MM performs a two phase federate migration procedure, similar to the techniques described in [27] and [28]. In this procedure, static files are first transmitted through Grid services [29], and the migrating federate is then submitted to the remote resource. In the second phase, the federate execution state information is transmitted through a peer-to-peer data transfer; once the transfer is complete, the execution of the migrating federate is resumed.

In order to allow for balancing adjustments, adaptation components are added into the balancing architecture. A History component is responsible for recording the migration decisions inside each cluster of resources, each resource, and

Algorithm 1: Main Load Balancing Algorithm

```
1  while TRUE do
2      loads ⇐ query_MDS()
3      current_loads ⇐ filter_MDS_data(loads)
4      current_loads ⇐ normalize(current_loads, benchmark)
5      overload_cand ⇐ select_overload(current_loads)
6      spec_loads ⇐ request_LLBs(overload_cand)
7      mng_loads ⇐ filter(current_loads, spec_loads)
8      mean, bds ⇐ calculate_mean_bds(mng_loads)
9      over, under ⇐ select(mng_loads, mean, bds)
10     mig_moves ⇐ redistribute_local(mng_loads)
11     send_migration_moves(mig_moves)
12     if mig_moves = ∅ then
13         inter_domain_balancing()
14     else
15         if relFactor ≥ random_number(1, 100) then
16             inter_domain_balancing()
17         else
18             neighbours_data ⇐ ∅
19         end
20     end
21     adaptation()
22     wait( Δt )
23 end
```

Algorithm 2: Inter-Domain Redistribution Algorithm

```
1  data_neighbours ⇐ request_Neighbour_Load_Data()
2  selectionParam′ ⇐ selectionParam ∗ selectionAdj
3  order_neighbours_by_load()
4  neighbours ⇐ identify_Neighbour_Less_Load()
5  if neigbours ≠ ∅ then
6      neighbours ⇐ select_Neighbours(selectionParam′)
7  else
8      overloaded_rcs ⇐ filter(extStD, localStD)
9      neighbours ⇐ select_Neighbours(extStD, localStD)
10 end
11 foreach neighbour IN neighbours do
12     overloaded ⇐ select(neighbour, selectionParam′)
13     federates ⇐ select(spec_loads, overloaded, selectionParam′)
14 end
15 if overloaded ≠ ∅ then
16     sort_list_load(overloaded)
17     sort_list_load(overloaded)
18     moves ⇐ redistribute(overloaded, overloaded)
19 end
20 send_migration_moves(moves)
21 adjust_factor(relFactor, overloaded, moves)
```

Algorithm 3: Local Pair-Match Evaluation Algorithm

```
1  Require: src_rsc, dst_rsc, federate
2  min′ ⇐ min ∗ minLoadAdj
3  if dst_rsc < min then
4      if number_fed(src_rsc) ≥ 1 & src_rsc > (min ∗ ϕ) then
5          create_migration_move(src_rsc, dst_rsc, federate)
6      else if number_fed(src_rsc) > 1 then
7          create_migration_move(src_rsc, dst_rsc, federate)
8      end
9  else if (dst_rsc − src_rsc) > (min ∗ δ) then
10     if number_fed(src_rsc) ≥ 1 AND
       (dst_rsc − src_rsc) > (min ∗ ϕ) then
11         create_migration_move(src_rsc, dst_rsc, federate)
12     else if number_fed(src_rsc) > 1 then
13         create_migration_move(src_rsc, dst_rsc, federate)
14     end
15 end
16 if migrationMove then
17     estMigTime ⇐ estMigTime(dst_rsc, src_rsc, federate)
18     Return: migrationMove, estMigTime
19 end
```

each federate. The History registers the presence of migration of a federate, the incoming or outgoing migrations for a resource, and the number of incoming and outgoing migrations for a cluster in a given balancing interval. The History is restricted by a size (H), which is used in calculating the Cluster Adaptation Component (CAC). The CAC performs all its balancing efficiency analysis based on the federate migration frequency and migration ratio. A Resource Adaptation Component (RAC) is used in order to aggregate the migration history from each resource and set of federates.

B. Load Balancing and Self-Adaptation Algorithms

The central balancing algorithm primarily collects load status information, analyzes this data, and provides modifications to the simulation load distribution, as described in Algorithm 1. The load redistribution consists of local and inter-domain (cluster) scopes. A CLB performs the local load analysis of the federates and resources that belong to a cluster. As delineated in Algorithm 2, the inter-domain load analysis is performed between a CLB and its neighbouring CLBs. Concurrent with both these algorithms, the self-adaption is executed, in turn generating adjustments for the balancing system.

As detailed in Algorithm 3 and Algorithm 4, the load of two selected resources is observed in a pair match analysis. A migration move is generated based on a comparison of the load of these selected resources, which is directed by thresholds (min for both local and inter-domain pair analysis) that are constantly adjusted by the self-adaptation mechanism ($minLoadAdj$ and $acceptanceAdj$). The inter-domain redistribution algorithm also contains an adjusted threshold ($selectionParam$) that is redefined each balancing cycle by an adaption variable ($selectionAdj$).

The adaption is performed for both local and inter-domain load redistribution algorithms. As described in Algorithm 5, the inter-domain adjustment is defined according to the ratio ($ratio_{clb}$) between incoming and outgoing migrations in a cluster in a given balancing history (H). The adjustment is applied by calling $ajdust\_Acceptance\_Threshold()$ for $acceptanceAdj$ or $adjust\_Overloaded\_List\_Threshold()$ for $selectionAdj$. As shown in Algorithm 6, the local adjustment employs the average resource ratio (avg_ratio_{rsc}) of incoming and outgoing migrations and the average federate migration frequency (avg_freq). The local adjustment is defined by $adjust\_Minimum\_Load\_Threshold(finalAdj)$ for $minLoadAdj$. In the local adaptation algorithm, $finalAdj$ is obtained through the analysis of both frequency and ratio.

IV. Migration-Delay Analysis in the Self-Adaptive Scheme

Even though the self-adaptation mechanism enables the constant evaluation of migration decisions, it is entirely limited to the analysis of migration frequency and migration ratio. The adaptive balancing scheme may implicitly consider migration delays in its algorithm and parameters, though the scheme does not observe any metric that represents or reflects these delays. However, this design cannot directly cope with or flexibly identify balancing efficiency loss solely through the analysis of migration frequency and ratio. As a result, a detection and reconfiguration method needs to be introduced into the self-adaptive scheme to enable the proper balancing adjustment according to the different migration delays.

Algorithm 4: Inter-Domain Pair-Match Algorithm

```
Require: int_rsc, ext_rsc, federate
min' ⇐ min * acceptanceAdj
if ext_rsc < min then
    create_migration_move(src_rsc, dst_rsc, federate)
else if (ext_rsc − int_rsc) > (min * δ) then
    create_migration_move(src_rsc, dst_rsc, federate)
end
if migrationMove then
    estMigTime ⇐ estMigTime(dst_rsc, src_rsc, federate)
    Return: migrationMove, estMigTime
end
```

Algorithm 5: Adaptation Algorithm for the Distributed Load Redistribution

```
Require: ratio_clb, clusterINMig, clusterOUTMig
if ratio_clb < t_ratio_clb & ratio_clb > 1/t_ratio_clb then
    if ratio_clb ≥ 1 then
        ajdust_Acceptance_Threshold()
    else if ratio_clb < 1 then
        adjust_Overloaded_List_Threshold()
    end
else
    if clusterINMig == 0 then
        if past_Increase_IN then
            decay_IN_factor()
        end
    else if clusterOUTMig == 0 then
        if past_Increase_OUT then
            decay_OUT_factor()
        end
    end
end
```

Algorithm 6: Adaptation Algorithm for the Local Load Redistribution

```
Require: avg_ratio_rsc, avg_freq, clusterINMig, clusterOUTMig
if avg_freq ≥ t_freq then
    freqAdj ⇐ adjustFreq(t_freq)
end
if ratio_clb < t_ratio_rsc & ratio_clb > 1/t_ratio_rsc then
    if ratio_clb ≥ 1 then
        ratioAdj ⇐ adjustRscRatio(t_ratio_rsc)
    else if ratio_clb < 1 then
        ratioAdj ⇐ adjustRscRatio(1/t_ratio_rsc)
    end
end
if freqAdj > 0 || ratioAdj > 0 then
    if freqAdj = 0 then
        finalAdj ⇐ ratioAdj
    else if ratioAdj = 0 then
        finalAdj ⇐ freqAdj
    else
        finalAdj ⇐ calculateAdj(freqAdj, ratioAdj)
    end
else
    decayLocalParameers()
    finalAdj ⇐ 0
end
adjust_Minimum_Load_Threshold(finalAdj)
```

A. Migration Latency

The analysis of migration delay (d_{mig}) provides a means by which to determine the cost of load migrations in a distributed environment. The awareness of such costs allows the balancing system to more properly react to load imbalances. This enables the system to define the amount of load it can transfer without compromising the final distributed simulation performance. Since simulation performance is measured according to the amount of time spent finishing an execution, a reduction of this time corresponds to a performance gain. Consequently, an increase in simulation time directly represents a performance loss, and this increase might be produced when redistributing a load with costly migration delays.

Due to the importance of migration latency for balancing distributed simulation load, this becomes a vital metric in defining the self-adaptation mechanism. Not only is the frequency of federate migrations used to adjust the balancing behaviour, but a weight for migrations in such frequencies is also applied to more precisely define an adaptation. The improvement of balancing efficiency is obtained through a reduction in the number of migrations, which achieves the same simulation performance improvement. However, according to this efficiency assumption, the number of additional federate migrations does not affect the simulation performance gain. For instance, a reduction in the number of federate migrations with negligible migration latency does not influence the final distributed simulation time. On the other hand, when costly delays are present in the migrations, not considering this time cost might lead the self-adaptation to not restrict the imbalance tolerance enough to impede the increase in the simulation execution time. Therefore, a control mechanism can be applied to the adaptation technique by increasing the tolerance to migration frequency and ratio when higher d_{mig} are present in the last balancing cycles or by increasing the tolerance to migration frequency and ratio when lower d_{mig} exist.

d_{mig} is estimated for the distributed balancing system's design as the time that a federate's execution is suspended for the migration procedure, which reflects a delay on the simulation execution. For the migration process, the total time of migration is the amount of time spent transferring the static files and remotely starting up the federate, plus the time to transfer and recover the federate's execution ($T_{mig} = t_f + d_{mig}$). Since a two phase migration technique is used in the balancing system, only the second part of the migration (d_{mig}) generates delays on simulations.

Moreover, each balancing cycle lasts for Δt independent of the time needed to perform all load analyses, issue migrations, and conduct the entire migration procedure. Consequently, at the load balancing system scope, the total time needed for a migration to be completed is the sum of required tasks from initial analysis and resumption of a federate's execution ($T'_{mig} = d_{mig} + t_f + t_a + t_i$, in which t_a is the analysis time and t_i is the issuing time). In order for this total migration time to interfere with the frequencies, it is necessary for it to overlap 2 balancing cycle time intervals ($T'_{mig} \geq 2 \times \Delta t$) since a pair of resources involved in a federate migration is exempted from balancing analysis for one balancing cycle. Based on this inequality, the value of the migration delay that can interfere with the calculation of migration frequencies must be at least the subtraction of the balancing interval time by the time used to generate the migration ($d_{mig} \geq 2 * \Delta t + t_f - t_a - t_i$). As a matter of simplification, it is assumed that all balancing time consumed before migration peer-to-peer data transfer initiates is up to half of the balancing interval ($t_f + t_a + t_i \leq \Delta t/2$). According to this assumed interference value for d_{mig}, for the purpose of this migration-aware technique, it is established that the maximum allowed migration time in the analysis is two times the balancing interval.

B. Migration Delay Influence on Balancing Adaptation

In order to introduce the migration delay awareness into the self-adaptation mechanism, the most recent average migration latency is determined. Based on the migration delays gathered from each federate migration in the last balancing interval, the value of migration delay that is obtained, as described in [30] and by Formula 1. The calculated migration delay value represents the overall time spent for migration in the last balancing cycle. This value is used to modify the self-adaptation parameters rather than each migration delay individually because the modified parameters are responsible for coordinating the balancing adaptation of an entire cluster, which is managed by a single CLB. In the formula, the overall migration delay is primarily estimated by the largest migration delay, which is responsible for the majority of the time spent for migrating during a balancing interval. In a set of migrations generated by the balancing system, the migration procedures occur concurrently, which provides evidence of the existence of overlapping migration times. As a consequence, the estimated migration delay consists of the largest value and the time computed in Formula 2, which defines the influence of other migration delays on the final value. The influence of these values is determined by the average migration delay (t_e), the number of migration delays above the average (k), and largest migration delay in the sample.

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

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

The self-adaptation is modified by replacing the main parameters that define the adjustments in the balancing scheme by thresholds that can be altered as the migration delay conditions on the balancing system change. As defined in [25], the main parameters of self-adaptation are static values that represent inefficient load balancing. These values define the adaptation thresholds and are obtained through an analysis based solely on estimations of migration frequency and ratio conditions because the adaptation technique does not consider migration delay in its design. Because balancing efficiency is not only measured through the number of migrations but also determined by the time spent on migrations, the adaptation thresholds need to be adjusted according to the current migration times. This adjustment is applied proportionally to the amount of overall migration delay from the last balancing cycle. Consequently, based on the self-adaptation technique's design, maximum and minimum values are delimited for each adaptation parameter as follows: migration frequency, resource migration ratio, and cluster migration ratio. The minimum migration frequency is defined as the $2/(H+1)$, which corresponds to the occurrence of one migration in the entire stored history according to the self-adaptive analysis. On the other hand, the maximum migration frequency is delimited as 1, which represents the existence of migrations in every balancing cycle. The resource migration ratio has its minimum defined as 1, which reflects the exact same amount of incoming and outgoing migrations in the resources. The maximum is delimited as $(H^2+2\times H)/8$, which is contained in a resource only either receiving or sending federates during the whole history. For the cluster migration ratio. The minimum value is determined as 1, which represents the same minimum as the minimum resource migration ratio, and the maximum as $\sum_{i}^{H/2}(H/2-i)*n_i$, which corresponds to the cluster presenting only either incoming or outgoing migrations multiplied by the number of current resources in the cluster.

The ranges for migration frequencies and ratios delimit the boundaries constraining the migration delay and determine its influence on the adjustment of the parameters of the self-adaptive mechanism. From each parameter side, the boundaries dictates the weight a migration delay presents on each adaptation parameter. Based on the defined ranges and the latest migration delay estimation, three main self-adaptation thresholds are then modified. These thresholds are strictly related to each of the three adaptation parameters and are used to directly control the load balancing responsiveness. For the migration frequency, t_{freq} in Algorithm 6 is altered. In order to modify the adjustment weight of the resource migration ratio on balancing, a new value is recalculated for $t_{ratio_{rsc}}$ in Algorithm 6. Comparable to the resource migration ratio, the cluster migration ratio is altered by computing a new value for $t_{ratio_{clb}}$ in Algorithm 5.

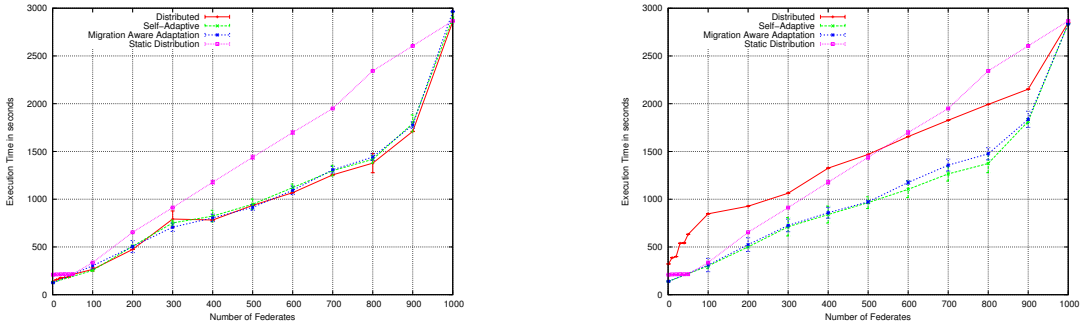
A range is also delineated for the migration delay to determine through comparison the amount of modification that needs to be applied on the self-adaptation mechanism in order to properly reflect the efficiency aspects according to the latest observed migration delays. This range is delimited by determining a minimum migration delay (min) and a maximum migration delay (max). min is set to a hypothetical value of 0, in which no delay is spent in applying the simulation load rearrangement defined by the balancing system, whereas max contains a value that restricts or enforces the influence of the migration delay. Based on both range boundaries, a normalized migration delay is calculated by Formula 3.

$$n_{d_{mig}} = \frac{(d_{mig} - min)}{max} \tag{3}$$

Adjustments for the adaptation parameters (thresholds) are computed using this normalized value of migration delay. All calculations adopt the same technique to modify the adaptation values, applying a rule of proportionality. As described in Formula 4, t_{freq}, limited by its computed range, presents its alteration inversely proportional to the migration delay since the higher the delay the more intolerant it becomes to frequent migrations. As shown in Formula 5 and in Formula 6, both $t_{ratio_{clb}}$ and $t_{ratio_{rsc}}$ are calculated proportionally to the normalized migration delay based on their own defined ranges. For the ratios, the higher the delay the larger the balancing intolerance delimited by the ratio range.

$$t_{freq} = min_{t_{freq}} + (max_{t_{freq}} - min_{t_{freq}}) \times (1 - n_{d_{mig}}) \tag{4}$$

$$t_{ratio_{clb}} = min_t + (max_t - min_t) \times n_{d_{mig}} \tag{5}$$



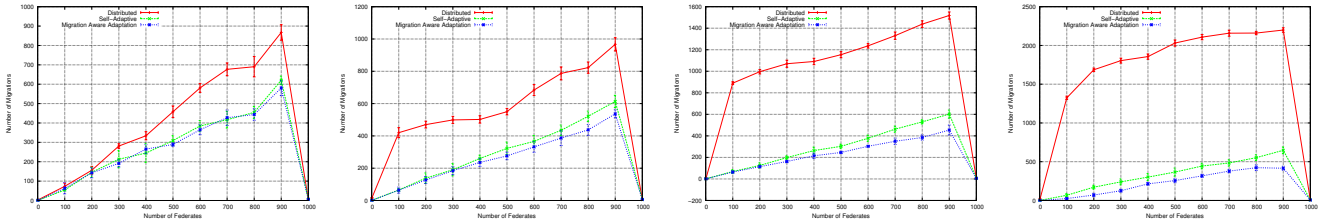
(a) Migration Latency of 1 object

(b) Migration Latency of 200 objects

(c) Migration Latency of 400 objects

(d) Migration Latency of 600 objects

Fig. 2: Performance Gain Analysis for an Increasing Number of Federates with Static Load



(a) Migration Latency of 1 object (b) Migration Latency of 200 objects (c) Migration Latency of 400 objects (d) Migration Latency of 600 objects

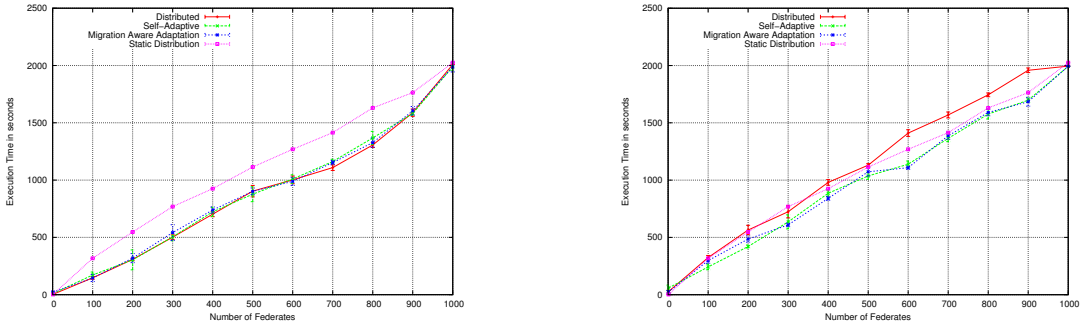
Fig. 3: Balancing Efficiency Analysis for an Increasing Number of Federates with a Static Load

$$t_{ratio_{rsc}} = min_{t_{rsc}} + (max_{t_{rsc}} - min_{t_{rsc}}) \times n_{d_{mig}} \quad (6)$$

V. Experiments

In order to assess the effectiveness of the proposed scheme in providing balancing efficiency gain, experiments have been undertaken within a distributed environment. The proposed balancing, self-adaptive [25], and distributed [24] balancing systems are compared through extensive execution of HLA-based simulations on a testbed. The distributed environment used in the experiments consisted of two computing clusters connected by a fast-Ethernet network link. One cluster was composed of 24 computing servers. Each server consisted of a Quadicore 2.40GHz Intel(R) Xeon(R) CPU and 8 gigabytes of RAM, and they were connected through a Myrinet optical network with a throughput up to two gigabits per second. The other cluster was comprised of a set of 32 computing servers. Each server consisted of a Core 2 Duo 3.4 GHz Intel(R) Xeon(R) CPU and 2 gigabytes of RAM, and they were connected through a gigabit Ethernet network. The main operating system used in the servers was Linux. The HLA RTI 1.3 was employed to manage the simulations' execution in each experiment, and the Globus toolkit was used to provide the Grid services for the balancing systems. In each experiment, simulation federates were evenly deployed on every computing server, consisting of a static distribution of load. This load deployment also defined the experimental baseline because of the heterogeneity of resources between the two clusters.

The simulation scenario consisted in the move coordination of tank teams in a two dimensional routing space. Each federate was responsible for exchanging data and computing movement positioning in time-stepped simulations. These simulations ran for 100 time-steps and were composed of 1 to 1000 federates. In order to generate intensive computing processing on the resources, a synthetic load [31] was added into each federate. The synthetic load was used to create replicable static, intensive loads for all federates for the purpose of analysis. In the dynamic load scenario, the simulation load randomly changed its processing intensity each $2 \times \Delta$t seconds as an attempt to produce unpredictable load oscillations. The migration delay was generated by adding special objects into the federates. These objects, together with the latency, gener-



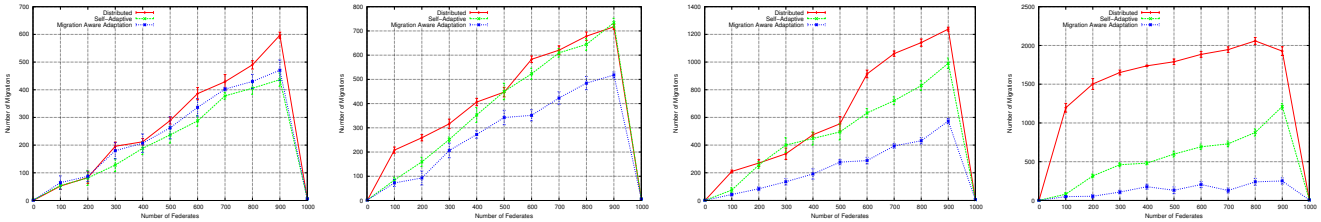
(a) Migration Latency of 1 object

(b) Migration Latency of 200 objects

(c) Migration Latency of 400 objects

(d) Migration Latency of 600 objects

Fig. 4: Performance Gain Analysis for an Increasing Number of Federates with a Dynamic Load



(a) Migration Latency of 1 object (b) Migration Latency of 200 objects (c) Migration Latency of 400 objects (d) Migration Latency of 600 objects

Fig. 5: Balancing Efficiency Analysis for an Increasing Number of Federates with Dynamic Load

ated considerable delay time when transferring the federate's execution state. In order to observe different migration latency scenarios, 1, 200, 400, and 600 objects were assigned to each federate. Moreover, the proposed balancing in the experiments used a history size of 10 and a maximum migration delay (max) of two times the balancing interval ($2 \times \Delta t$).

In the first set of experiments, a static load was used to evaluate the effectiveness of the proposed scheme in detecting migration delays and adjusting the adaptation mechanism. For these experiments, federates presented the same constant static synthetic load. As depicted in Figure 2, the static distribution corresponded to the baseline, which was the basic initial deployment of federates on all resources. The experiment with negligible migration delay (1 object) displayed similar balancing performance gains for all compared balancing schemes. However, as the added migration delay increased in the simulations, the distributed and self-adaptive schemes presented an increasing overhead. This overhead was caused by the large amount of migrations, which can be observed in Figure 3. When comparing both distributed and self-adaptive approaches, the self-adaptive approach presented less overhead in the simulation execution time since by its design it systematically reduced the frequency of migrations. The proposed scheme was able to prevent the overhead caused by the costly migrations by weighing the self-adaptation mechanism, which consequently increased the intolerance to load imbalances in the distributed balancing system. When migration delays were caused by simulations with more than 400 objects, all the balancing systems except the proposed scheme presented performance loss.

The balancing efficiency in detecting and responding to load imbalances was evaluated in the set of simulations with dynamic load oscillations. As presented in Figure 4, the simulation performance gain was achieved by all balancing systems with minimum migration delay. All curves show the same simulation execution times. Similarly to the static scenario, both balancing schemes that are unaware of migration delays exhibit increasing balancing performance loss as the migration delay increases. The effect of the delays can be observed throughout the graphs in Figure 5. In the dynamic load oscillations the self-adaptive scheme presented a larger amount of migrations. This was the result of the oscillations occurring in intervals that overextended the history used to manage the frequencies and ratios, even when the

balancing responsiveness was restricted by the self-adaptive actions. Since the proposed scheme totally relies on the self-adaptation mechanism, it presented simulation performance loss for migration delays generated by simulations with over 600 objects. This occurred because the balancing system had its migration ratio and frequencies renewed every time interval with the size of whole adaptation history.

VI. Conclusion

In this paper, a self-adaptive balancing scheme aware of migration delays and suitable for use in HLA-based simulations is presented. The migration delay analysis aims to modify the needs of balancing efficiency according to the current latency in redistributing simulation load. Adjustments on the adaptation technique were realized, influencing the detection and modification of balancing efficiency. The experiments conducted in order to evaluate the proposed scheme demonstrated that the migration awareness in self-adaptation was indeed able to improve the balancing efficiency. However, the devised scheme presented performance loss into simulations with very costly migration scenarios (600-object simulations). This loss was generated by the migration-aware technique's dependency on the self-adaptation mechanism. Therefore, as future work will be conducted in which to consider migration time gain into the adaption adjustment. It will also endeavour to perform simulation composed of federates with variable state sizes as well as employ other adaptation methods.

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 14th 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 35th Annual Simulation Symposium (SS02)*. IEEE Computer Society, 2002, pp. 267–274.

[4] 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.

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

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

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

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

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

[10] R. E. D. 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.

[11] ——, "Measuring communication delay for dynamic balancing strategies of distributed virtual simulations," *Instrumentation and Measurement, IEEE Transactions on*, vol. 60, no. 11, pp. 3559–3569, 2011.

[12] 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.

[13] 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.

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

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

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

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

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

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

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

[21] 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.

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

[23] 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, 2000.

[24] 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 (Euro-Par)*, ser. EuroPar'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 242–253.

[25] ——, "Self-adaptive dynamic load balancing for large-scale hla-based simulations," in *Proc. of the International Symposium on Distributed Simulation and Real Time Applications*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 14–21.

[26] 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.

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

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

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

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

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