A Redistribution Scheme Centred on Communication Delay for Distributed Virtual Simulations

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

***Abstract*—Communication latencies directly influence the performance of distributed virtual simulations due to existent dependencies between simulation elements. The High Level Architecture (HLA) was designed to organize these simulations and reduce communication overhead. Even though the framework successfully manages data distribution, it is not concerned of communication distances and the network topology, which generate mostly of the delays in simulations. In order to provide a solution for organizing distribution simulations according to communication aspects, many approaches have been proposed. The approaches that provide a broader solution consider the proximity of resources in their redistribution algorithms. Even though these schemes considerably improves simulation performance, they are based on static characteristics of networking resources. Thus, a redistribution scheme is proposed to include communication delay as the main balancing metric and to detect the dynamic changes in systems' communication load. Experiments have been performed to compare the proposed scheme with the previous distributed scheme and to determine the effectiveness of using delay for balancing HLA-based simulations.**

I. Introduction

The decrease of communication latencies is highly valuable for distributed virtual simulations' performance since it prevents delays and improves execution time. Consequently, the HLA framework provides mechanisms to organize simulations' communication by allowing the transfer of useful data. However, such mechanisms cannot decrease the communication latencies originated from the network resources or the topological distances between the simulation elements. On the other hand, static and dynamic balancing techniques exist to diminish simulation intra-dependencies, but these techniques are limited because they are unaware of dynamic changes from the network infrastructure.

The HLA framework [1] was proposed to introduce a standard that facilitates the coordination of distributed simulations. The framework is basically composed of a set of rules, interface specifications, and object model templates; these elements allow the re-usability and interoperability of simulation elements, which are called federates. A distributed simulation is composed of federates and Run Time Infrastructure (RTI) services, which are accessed to organize the simulation execution. More specifically, the Data Distribution Management (DDM) service in the RTI limits data transfers by enabling transmission of only relevant information between federates. This approach diminishes the consumption of network resources; nonetheless, the service cannot prevent the communication delays originated from improper deployment of federates.

In order to improve the performance of distributed simulations, many balancing approaches have been proposed. These approaches attempt to organize the computational load and the communication dependencies of simulation entities to maximize the utilization of resources and minimize communication latencies. Even though the balancing of computational load considerably improves simulation performance, distributed simulations' performance is totally involved with communication dependencies. Focusing on balancing HLA-based virtual simulations for decreasing communication delays, schemes were proposed in [2] and [3]. These schemes extended the previous works by observing the proximity of resources to reorganize federates. However, such balancing techniques rely on a static network status scenario, which can mislead the redistribution algorithm.

Therefore, in order for the previously proposed balancing schemes to be able to detect the dynamic changes in the networking resources, a redistribution algorithm based on communication delay is proposed. The proposed scheme also performs a proximity analysis that attempts to reduce or minimize the network latencies in virtual simulations. Likewise the distributed balancing scheme [3], this balancing system is organized in monitoring, re-distribution, and migration phases in order to re-arrange federates dynamically. Furthermore, even though the system's structure is organized hierarchically, the relations between its balancing elements are driven by communication delays to adapt to the dynamic changes that might occur during run-time.

The remainder of this paper is organized as follows. In Section 2, the related work and challenging issues are presented. In Section 3, the proposed balancing system is described by delineating its architecture and functioning. In the Section 4, experiments are defined, and their results are discussed. Finally, in Section 5, the conclusion briefly outlines the paper and delineates directions for future work.

II. Related Work

Due to the importance of balancing communication load of distributed simulations to improve their execution performance, several balancing schemes have been proposed. Some of such balancing techniques just consider communication aspects in their decision-making part of their algorithms while other techniques really reorganize the deployment of simulation entities according to simulation inter-dependencies. These inter-dependencies are influenced by communication dependencies and generate cumulative latencies on simulation time.

Look-ahead and communication rate are the main metrics used in the monitoring and analysis of balancing schemes to decrease delays in distributed simulations. The look-ahead shows the simulation internal relations and provides means to indirectly detect entities that cause slow execution of simulations [4] [5]. On the other hand, the analysis of communication dependencies allows the detection of direct factors that influence simulation performance, such as network distances and/or communication overhead; this analysis can be performed statically [6] [7] for deterministic simulations by observing their critical communication path or dynamically [8] [5] [9] [10] [11] [12] [13] [14] for simulation entities with unpredictable interaction behaviour. All these balancing techniques do not consider the proximity of resources in order to redistribute simulations, so two approaches [2] [3] have been proposed to broaden the redistribution scope.

The proximity analysis enables more opportunities to reduce communication latencies and to improve simulation performance. In [2], the proposed balancing system is organized hierarchically with a centralized data analysis that uses the global view of entire system to re-organize load. In order to overcome the negative aspects of this centralized design, such as bottlenecks, synchronization, and overheads, a distributed, hierarchical balancing design is introduced in [3]. Although this approach cannot rely on a global view, it benefits the balancing system by enabling independence and flexibility between the balancing elements. However, even with this approach, the balancing system is unable to detect the dynamic communication load changes caused by external processes or by the uncontrolled saturation of network resources. Therefore, an analysis of these dynamic communication characteristics is needed to coordinate the redistribution according to the network resources' status.

III. Balancing System Based on Communication Delay

The adoption of a distributed structure for the proposed balancing scheme allows to avoid synchronization issues, preventing failures, delays, and overheads. The structure is organized hierarchically according to the network topology, which establishes the classification of simulations in different domains. In order to react to dynamic changes, the balancing scheme analyzes the current system's load status, which is obtained from the system's recent past. This approach simplifies the estimation procedure, but it does not provides the most up-to-date status information about the system since the data collection is performed asynchronously. The architecture of the proposed balancing system is similar to the distributed one in [3]. However, the inter-relations between the balancing components are modified to incorporate awareness of communication delays.

A. Architecture

As described in Figure 1, the Group Manager (GM) organizes all the balancing scheme. This balancing component controls simulation elements inside a domain, determining redistributions according to detected imbalances. As a consequence of generating load re-distributions, a GM coordinates all the balancing steps: monitoring, redistribution, and migration. In order for a GM to access the simulations elements, it interacts with a Local Management Agent (LMA). A LMA collects CPU utilization and communication behaviour of each federate and passes this information to its GM; it also forwards migration calls from GMs to federates. There are two subcomponents inside a LMA: the Communication Load Monitor that realizes data gathering, and the Migration Manager (MM) that forwards and coordinates federate migrations.

Moreover, awareness of resources' load status is essential for avoiding load imbalances, so a GM also accesses resource management systems to obtain such data. Grid services are used by the balancing system to obtain load information and to transfer federates. These services provide tools that facilitate the monitoring and scheduling of resources and applications [15] and are related to Grid computing, which involves the management of resources, applications, and institutions [16].

B. Balancing Scheme

As previously mentioned, the balancing algorithm is divided in monitoring, redistribution, and migration. The monitoring phase consists in collecting information and detecting imbalances. The data collection is responsible for retrieving information related to resources and federates that belong to a specific domain. The collected information comprises computational load of a resource, CPU consumption of a federate, and communication status of a federate. Upon receiving the collected data, a GM filters and analyzes it in order to detect communication imbalances and redistribute the simulation federates. This redistribution is performed between a GM and its neighbour GMs. After a redistribution is determined, migration is performed. Coordinated by a MM, the migration is performed in two main steps [17] [18]. In the first step, a federate is initialized at a remote resource after its static data is transferred reliably through Grid services. During this migration step, the federate's execution is not suspended, so the migration latency's influence on simulation performance is decreased. In the second step, a migration proxy is used to intermediate the peer-to-peer transfer of dynamic data, which is a federate's execution state.

C. Detection and Redistribution Algorithm

Delay is the main metric employed to determine imbalances in simulations. The use of delay allows a better adaption to

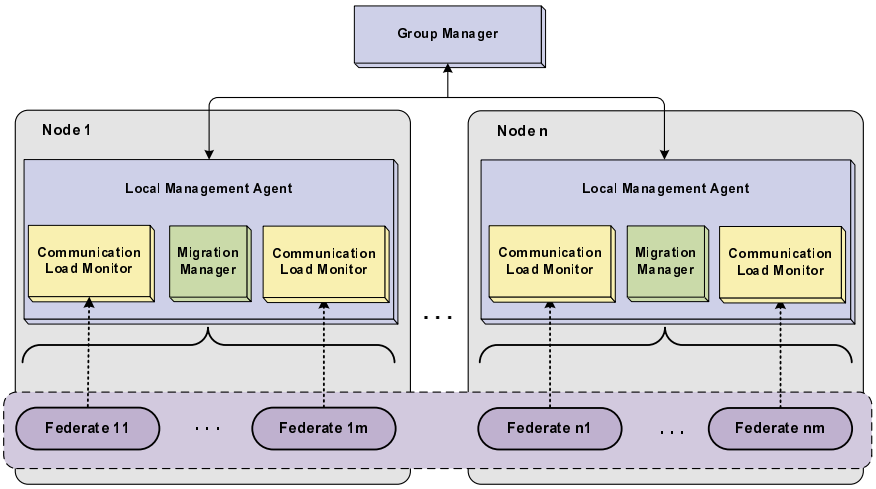

Fig. 1: The Dynamic Communication Balancing's General Architecture

run-time imbalances and awareness of network capabilities. The delay provides means to observe the network conditions from the simulation point of view, detecting when network resources undergo overload. Because detection and redistribution are based on communication delay, the inter-relations between GMs are totally modified. Thus, since delay may change dynamically, the direction to which federates are moved needs to be modified accordingly, which is reflected on the balancing behaviour.

In this proposed balancing scheme, delay is assumed as time (milliseconds) spent per the amount of transmitted data (byte). As a result, the system registers the delay of each message sent by a simulation federate. The message delay is saved in a circular queue of size N. Delay is a volatile metric and oscillates frequently, so an average is calculated to produce a more stable estimation that reflects the network status during a Δ interval, decreasing sudden oscillations. The parameter N is used to limit the amount of stored information and to assign more weight to the recent communication past. Since this information is logged during a Δ time interval, recent communication changes that might occur at the end of the interval may be discarded by a too long list of delay values. On the other hand, delay oscillations can generate imprecise values that vary too much when N presents a small value. In order to simplify the issue, the circular queue is limited to 200 elements in the proposed system.

Furthermore, the balancing system needs to collect the total amount of data transmitted and received by a federate during a time interval. Therefore, this metric is provided upon request of LMA, which aggregates data from all federates in a resource and sends it to its GM. After monitoring data is gathered, an average delay is calculated for each resource based on its federates' average delay. A GM then computes the total amount of data transmitted in a domain, and it calculates the average domain delay based on non-zero resource average delays. After this information is obtained, the GM requests the delays and data transmitted from its neighbour GMs.

Domain delays together with transmitted data portions and goodput are used to identify distribution imbalances, as described in Algorithm 1. The goodput is an application layer metric that represents the amount of data delivered per time, and it is assumed to reflect the network distance that each domain is from the RTI; since goodput is obtained from the application layer, the metric considers all the factors that influence the transmission time until data is delivered to an application, such as processing of transmitted packets and the nominal network capacities in the distance between two communicating parts. A greedy technique is employed in the algorithm to determine a sub-optimal solution for the simulation distribution. As a matter of simplification for the problem complexity, this technique attempts to determine the most appropriate distribution to improve communication latencies. Based on delays, GMs are ranked, and pair analyses are performed between the GM and each neighbour GM, starting with the GM with lowest delay.

In the detection/redistribution algorithm, there are two approaches to even neighbourhood delays: to move communicative federates to resources with larger bandwidth or to free bandwidth for communicative federates. The former approach is performed by detecting neighbour GMs with delay lower than the GM's. This allows the transfer of communicative federates to better locations by iteratively comparing delays and goodputs. If a GM's delay is larger than a neighbour's delays plus an error, goodput analysis is performed. According to this analysis, the resources that are in a closer location (higher goodput) receive the most communicative federates. These federates are obtained by previously ranking the federate list through their communication rate; this list is then classified in three groups of communication intensity: high, medium, and low. The resources that are not closer to the RTI but present better delay receive federates with medium communication intensity. In the second approach, if a GM's delay is similar to its neighbours' delays ($delay_{ext} - error \leq delay \leq delay_{ext} + error$) and its goodput is better than its neighbours', federates with lowest communication intensity are selected. In this later case, the balancing system moves

Algorithm 1 Detection-Redistribution Algorithm

Require: $delay_{GM}, delay\_list, abs\_comm_{GM}, abs\_comm\_list,$
$goodput_{GM}, goodput\_list, federate\_list$
while $delay_{GM} > max(delay\_list)$ **do**
  $order\_ascendingly(delay\_list)$
  $temp\_GM \leftarrow select\_first(delay\_list)$
  if $delay_{GM} > (temp\_GM.delay + error)$ **then**
    if $goodput_{GM} \geq temp\_GM.goodput$ **then**
      $data\_comm \leftarrow calculate\_transfer()$
      repeat
        $fed\_eval \leftarrow select\_most\_comm(federate\_list)$
        if $fed\_eval.ab\_comm \leq data\_comm$ **then**
          $fed \leftarrow fed\_eval$
        end if
      until $fed \neq \emptyset$
    else
      $data\_comm \leftarrow calculate\_transfer()$
      repeat
        $fed\_eval \leftarrow select\_med\_low(federate\_list)$
        if $fed\_eval.ab\_comm \leq data\_comm$ **then**
          $fed \leftarrow fed\_eval$
        end if
      until $fed \neq \emptyset$
    end if
    $federate\_mig.add(fed)$
    $adjust\_delays(delay_{GM}, temp\_GM.delay)$
  else
    endLoop
  end if
end while
$GM\_list \leftarrow select\_similar\_delay(delay\_list)$
$order\_ascendingly(GM\_list)$
for $i = 1$ to $GM\_list.size()$ **do**
  if $goodput_{GM} > GM\_list_i.goodput$ **then**
    $data\_comm \leftarrow calculate\_transfer()$
    $fed\_eval \leftarrow select\_lowest\_feds\_fit(data\_comm)$
    if $fed\_eval.ab\_comm \leq data\_comm$ **then**
      $federate\_mig \leftarrow select\_lowest\_feds\_fit(data\_comm)$
    end if
    $federate\_mig \leftarrow fed\_eval$
    $adjust\_delays(delay_{GM}, temp\_GM.delay)$
  end if
end for
return $federate\_mig$

federates in order to free bandwidth for the federates with higher communication needs.

The parameter $error$ used to define equivalence of delay values corresponds to a percentage of the delay, and this parameter is directly related to detection. The balancing may never detect equivalency of delays if $error$ is too small or may include all delays if $error$ is too large. In this proposed scheme, based on extensive experiments, the $error$ corresponds to 3% of a delay.

For each iteration in any of approaches, a federate is selected to be migrated. However, before a federate is selected for migration, a communication load comparison is performed using Formula 1. A federate is selected if its communication load is lower than the value provided by the formula. This evaluation is required to prevent federate migrations that can aggravate the distribution conditions, producing additional communication latencies in simulations.

$$comm\_transfer = \frac{|d_2 - d_1| \times (a_1 + a_2)}{2 \times (d_1 + d_2)} \tag{1}$$

Formula 1 estimates the difference of communication load between two GMs. It uses the absolute amount of data transmitted in the each domain (a) and delay (d) of each GM. These absolute values provide means to identify the communication importance of federates in each domain during a time interval (Δ). The formula is used to obtain the difference instead of a simple subtraction between the absolute values because the subtraction may produce a wrong result. The simple subtraction does not correspond to best sub-optimal estimation since domains can present different goodputs and external background processes might also be consuming the network resources. The formula allows to incorporate a recent communication and network pseudo-status in the calculation through the introduction of delays. As a result, a more flexible value is achieved, which reflects the sudden network changes and allows a more precise load redistribution.

After a federate is selected to be migrated, communication loads and delays are adjusted for the next analysis iteration. Both GM and its neighbour receive updates for their respective communication load and delay parameters. This adjustment is introduced to control the amount of federates that are transferred and avoid unnecessary migrations. If adjustments are not realized, the balancing system may continuously over-select federates (high, medium, or low communication intensity) to be moved to resources in other domains. The absolute communication load value is adjusted through a subtraction ($a - a_f$) or addition ($a + a_f$), in which a is a GM's value and a_f is the communication load of the selected federate. However, the delay adjustments require additional calculations. The adjusted GM's delay is obtained through Formula 2. In this formula, the adjusted communication load value ($a - a_f$) is used to proportionally determine the value for the delay. Likewise, the adjustment of a neighbour's delay is computed through Formula 3. In this case, the federate's communication load is added instead of subtracted because the neighbour GM is receiving load through a migration process. After these adjustments, the neighbour GMs are ranked again and the procedure (first approach or second approach) is initiated again in another iteration.

$$adj_d = \frac{a \times d}{(a - a_f)} \tag{2}$$

$$adj_i = \frac{a \times d}{(a + a_f)} \tag{3}$$

Finally, all federate candidates are grouped in a list of selected federates together with their respective destination neighbour GMs. These federates in the list are sent to their GMs in order to perform the last step in the redistribution phase. Basically for a migration, it is required information to identify a federate, the resource on where it is running, and the destination remote resource. Up to this step, information about the destination resource is needed to complete the migration call and trigger the migration processes. Therefore, the GM groups the federate candidates according to their destination GMs and issues a message to the neighbour GM, requesting the proper resources to receive the candidates. Upon receiving his request, the neighbour GM orders its resources by their computational load and selects the resources with the least load. In this case, communication latencies are decreased

without producing computational load imbalances in the system. After resources are identified, the neighbour GM sends their information to the GM, which defines migration calls accordingly and forwards them to their respective federates. The resources that were involve with the triggered federate migrations are inserted in a list in order to be excluded from the next balancing cycle. With this approach, enough time is provided to resources to establish their normal execution, for irregularities (oscillations) that are caused by the migrations and can lead the balancing system to misinterpret the collected data the are avoided.

IV. Experimental Results

In order to evaluate the proposed redistribution scheme's effectiveness, an experimental analysis has been realized. The experimental analysis involved the execution of HLA-based distributed virtual simulations on two computing clusters and one computing server. One of the clusters (IBM) was composed of 24 computing servers, which were inter-connected through a gigabit Ethernet network link. Each computing server contained a Quadcore 2.40GHz Intel CPU and 8 gigabytes of RAM. The other cluster (Dell) consisted of 32 computing servers that were connected through a 2-gigabit Myrinet optical network link. Each computing node contained a Core 2 Duo 3.4 GHz Intel CPU and 2 gigabytes of RAM. The additional computing server contained an Intel i7 CPU with 8 cores and 6 gigabytes of RAM. This external computing server was located in a different network, so it was connected to the Dell cluster through a fast Ethernet network link and to the IBM cluster through a 10 base T network link. Consequently, due to the simulation dependency characteristics, all the communication latency in the simulations is driven by the these two connections to the external server, more specifically to the connection with smaller bandwidth. In this environment, Linux operating system and the Globus Toolkit 4.2.1 [19] were installed to support all the experiments, and the HLA platform with RTI version 1.3 was used to coordinate the simulations.

In the experiments, federates were deployed evenly on the 56 computing resources in both clusters. The HLA RTI executive that coordinated the simulations was placed on the external computing server. In the same way, the proposed balancing system was deployed on each cluster computing server. Each server received a Local Management Agent, and both management nodes of the clusters ran Group Managers.

A simulation scenario was used in the experiments; the scenario embraced training operations coordinated in a two-dimension routing space. Two teams of interactive tanks were coordinated by federates in time-stepped simulations. Each federate calculated the movement of a set of tanks, and for communication purposes, the tanks performed simple movements that did not require extensive computing. Also, each federate realized the publication of new positions and subscription to other federates' tank updates. Generally, 200 federates composed the simulations and organized 1 to 10 tanks each in 100 time steps. In order to control the communication imbalances, the tanks' updates were reduced to around 700 bytes while special objects were introduced in some federates to produce a 54000-byte load, generating considerable communication overhead.

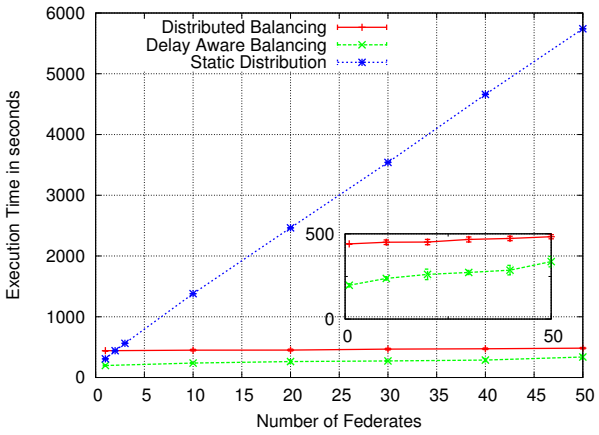

Fig. 2: Performance Analysis with Increasing Number of Communicative Federates

A static, restricted communication imbalance is introduced in the experimental simulations in order to observe the responsiveness of the proposed balancing scheme compared with the previous distributed balancing system [3]. As shown in Figure 2, every simulation federate controlled one object (tank), and some federates were selected to additionally coordinate the update of special objects, which generated considerable communication overload in the system. The curves in the graph show the influence of the communication delay in simulation performance as the number of communicative federates increased in the simulations. The static distribution presented the worst simulation performance with the induced communication imbalance since it did not modify the distribution according to the simulation communication behaviour. This linear, fast execution time increase was a consequence of the severe bottleneck introduced between the IBM cluster and the external computing server. Both proposed and distributed balancing systems presented curves that grew similarly. However, observing the focused graph in the figure, the proposed scheme showed a better improvement, which achieved almost the double of performance improvement when compared with the distributed approach.

For the same simulations, Figure 3 describes the number of migrations required by each balancing scheme to obtain a simulation performance improvement. The distributed balancing scheme performed considerably more migrations than the balancing scheme based on communication delay. This was a result of the inter-relations between balancing components in the distributed scheme; in order to minimize communication load, the balancing system constantly moved federates between domains and a large effort was spent to achieve such a distribution. On the other hand, the proposed scheme employed communication delays to identify and control the amount of migrations between domains, allowing the system to present a better balancing efficiency. Nevertheless, the proposed scheme

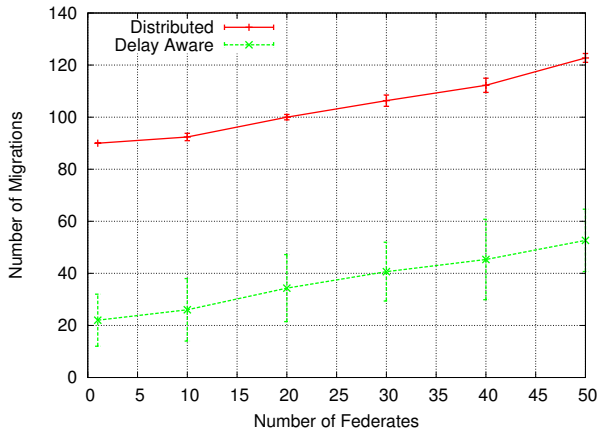

Fig. 3: Efficiency Analysis based on Number of Federate Migrations

showed more instable results, which is identified by the large error in its curve in the graph. This large variation was a result of the delay oscillations during the simulations. These sudden delay changes led the balancing system to vary the redistribution, generating different simulation execution times. Even with these oscillations, the proposed balancing scheme presented a more stable redistribution technique than the previous scheme, which reflected on the smaller number of migrations.

V. Conclusion

A redistribution scheme based on communication delay is proposed to improve adaptability of the previously-proposed distributed balancing system for distributed virtual simulations. The balancing system's components are structured hierarchically according to the network topology where it is deployed. The use of delay imposes major modifications on the inter-relations among the distributed balancing components, which differs completely from the previous proposed system but still allows independent, asynchronous interactions between components. In this case, the structure is organized dynamically as the network resources are available. Moreover, the balancing scheme is organized in monitoring, detection, redistribution, and migration, in which the detection and redistribution basically rely on dynamic communication status. Grid services are accessed in the balancing system to provide monitoring computational load data, reliable data transfers and process submissions for federate migrations.

The experimental results showed performance and efficiency improvement when the proposed balancing scheme was compared with the previous distributed balancing approach. This improvement results from the reduction in number of costly federate migrations. The use of delay as a monitoring metric allowed the balancing system to avoid unnecessary, precipitated load transfers, reducing the number of migrations. However, the volatility of delay during the experiments evidenced the need of further studies to achieve more accurate redistribution of simulation elements.

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] R. E. D. 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)*. IEEE Computer Society, 2009, pp. 176–181.

[3] ——, "Distributed dynamic balancing of communication load for large-scale hla-based simulations," in *Proceedings of the IEEE Workshop on Performance Evaluation of Communications in Distributed Systems and Web based Service Architectures*. IEEE Computer Society, 2010, to Appear.

[4] 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 *Proceedings of the 14th workshop on Parallel and distributed simulation*. IEEE Computer Society, 2000, pp. 139–146.

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

[6] R. Schlagenhaft, M. Ruhwandl, and C. S. 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*. IEEE Computer Society, 1995, pp. 175–180.

[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*. IEEE Computer Society, 1999, pp. 101–108.

[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*. IEEE Computer Society, 2004, pp. 3232–3238.

[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)*. IEEE Computer Society, 2007, pp. 219–228.

[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*. IEEE Computer Society, 1999, pp. 20–28.

[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*. IEEE Computer Society, 1994, pp. 164–172.

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

[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*. IEEE Computer Society, 2008, pp. 61–68.

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

[16] 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.

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

[18] 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*. IEEE Computer Society, 2007, pp. 113–121.

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