A Dynamic, Distributed, Hierarchical Load Balancing for HLA-Based Simulations on Large-Scale Environments

Robson Eduardo De Grande and Azzedine Boukerche

School of Information Technology and Engineering, University of Ottawa, Canada
{rdgrande,boukerch}@site.uottawa.ca

Abstract. The dynamic management of load in large-scale distributed systems is essential for the performance of simulations due to the influence that computing capacity and work load have on execution time. The High Level Architecture (HLA) was designed with the purpose of providing management services in order to organize distributed simulations, but the framework does not offer tools for controlling load imbalances of distributed simulations. In order to provide a generic solution for the simulation load imbalances, many approaches have been proposed. These schemes are limited to solve balancing issues regarding specific simulation or environment characteristics. With focus on balancing the computational load specially for HLA-based simulations, an approach have been previously proposed based on a centralized method, but this solution performs load re-distributions based on a central element, introducing global synchronization in the system. Therefore, avoiding the issues caused by centralization, a distributed, hierarchical balancing design is proposed to dynamically organize the load through three phases: monitoring, redistribution, and migration. The proposed scheme addresses improvement of fault tolerance, decrease of balancing overhead, and reduction of delays and bottlenecks, while exhibiting performance similar to the centralized approach in the experiments.

1 Introduction

Due to the need for solving complex problems, large-scale HLA-based simulations have been receiving significant attention, mainly fro solutions to balance their load properly. Distributed simulation applications can undergo performance issues in large-scale systems due to the dynamic characteristics of simulation load, heterogeneity of resources, and external background load. Even though a static partitioning can properly and evenly distribute load according to resources capacities, it cannot predict the dynamic load changes of simulations and external background processes. Therefore, a dynamic, distributed load balancing scheme is devised to react to run-time load changes, re-allocate resources accordingly, and keep the HLA-based simulations running consistently.

The High Level Architecture (HLA) [20] was designed aiming to organize the execution of distributed simulations. The framework consists of a set of rules, interface definitions, object templates, and services, which are used by simulation

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part I, LNCS 6271, pp. 242–253, 2010.

entities, called federates, to provide interactions and data distribution consistently. An HLA simulation is named federation, and it is basically composed of federates and a Run-Time Infrastructure (RTI). The RTI is responsible for providing management services to inter-connect federates. However, this framework only manages simulations and not the resources on where they are running. Consequently, imbalances are caused by inappropriately placing federates on unreliable, non-dedicated shared resources, so additional balancing mechanisms are required in order to allocate resource according to run-time simulation needs.

Several approaches have been proposed to solve the imbalance issues of distributed simulations. Aiming to maximize simulation performance with a generic balancing solution, the proposed balancing schemes observe the distributed computation load and/or the communication dependencies among simulation entities. The balancing schemes that manage computation load consider different simulation and environment aspects, but they do not provide a redistribution solution that solve the balancing issues of large-scale HLA-based simulations. As a result, a balancing scheme has been proposed in [8] in order to consider large-scale aspects and HLA simulation characteristics in its redistribution algorithm. Although this scheme successfully improves simulation performance through a hierarchical structure that decreases data transfers, it employs a centralized redistribution algorithm, which might cause issues depending on the scale of balanced distributed systems.

Therefore, a balancing scheme is conceived to provide a distributed redistribution algorithm that enhances reaction and adaptation to dynamic load changes in large-scale HLA-based simulations. The scheme still employs a similar architecture and multi-layered hierarchical structure to [8], but it discards the use of a centralized element to detect and re-arrange the distributed load. The proposed scheme gradually redistributes by starting load changes from a local scope and reverberating them to entire system. The balancing system presents a scheme divided in three sequential phases: monitoring, redistribution, and migration. Consequently, the proposed balancing system aims to present similar efficiency to the centralized approach but additionally to improve scalability aspects.

The remainder of this paper is organized as follows. In section 2, the related work is presented, evidencing the existing solutions for redistributing load dynamically and their drawbacks. In section 4, the proposed distributed, hierarchical three-phase scheme for balancing computation load is introduced. In section 5, the experimental results are shown and discussed. Section 6 presents the conclusion and directions for future work.

2 Related Work

Due to the complexity in designing balancing schemes and their importance for distributed systems, many balancing approaches have been proposed, attempting to provide considerable performance gain for distributed simulations. These approaches basically consider communication latency and computation load in their schemes. Communication latency has considerable relevance because of the

inter-dependencies between simulation entities and the communication delays produced by the network infrastructure, but computation load is vital since the utilization of shared resources can be maximized and better employed for the benefit of a simulation's performance, speeding up its execution.

Many balancing schemes focus on reorganizing the load according to the internal dependencies of simulations, monitoring and analyzing the simulation look-ahead and/or the communication rate of simulations. The look-ahead provides a simulation-dependent solution for determining simulation entities that are generating delays, as described in [14]. Other approaches analyze the communication behaviour of simulations, such as in [19] [7] [3] [16]. All these schemes produce a performance improvement, but they are not focused on maximizing the shared resource utilization.

Aiming at the improvement of resources' consumption, several balancing schemes extensively employs computation load to evenly distribute load. The computation load is analyzed through a simulation centred approach and a resource centred approach. The simulation centred approach observes and compares the execution speed of simulation entities [15] [9] [11] [21]. This approach uses simulations to indirectly evaluate the resources's load, which can mislead the identification of imbalances. The resource centered approach uses the CPU consumption or simulation throughput to determine imbalances and re-organize the load [22] [12] [5] [18] [10] [23] [2] [4] [1]. This approach provides means to directly improve the distribution of load. These techniques offer some performance gain for distributed simulations, but they are not fully concerned about resource heterogeneity, external background load, or the HLA simulation characteristics.

In order to fulfil this lack, a dynamic balancing scheme has been proposed in [8]. This scheme focuses on computation load due to its direct importance to large-scale systems. In a hierarchical structure, this scheme rearranges HLA-based simulation load and is aware of heterogeneity of resources and external background load. However, this approach presents a centralized redistribution algorithm that can become a bottleneck in the balancing system, cause delays, and present single a point of failure. To overcome such a drawback, a distributed balancing scheme that performs inter-domain re-distributions is introduced to coordinate the execution of large-scale HLA-based simulations.

3 Distributed Load Balancing Scheme

The main objective of the proposed balancing scheme is to decentralize the redistribution process while allowing a balancing performance similar to the centralized approach, which employs a global distributed system view to produced to most proper load changes. Instead of having an unique element that performs balancing at global scope, each balancing element is responsible for managing the load of a particular region. This regional balancing starts at the local management of a set of resources and is expanded to the entire distributed system through inter-domain, peer-to-peer load moves. Consequently, the balancing scheme is concerned of the relations between balancing elements to maintain an even distribution of load. To react properly to run-time load changes,

the scheme is divided in three phases that perform federate and resource monitoring, distributed load rearrangement, and federate migration.

3.1 Architecture

The architecture of the balancing system consists of elements that are divided according to the mentioned three balancing phases, similarly to the architecture proposed in [8]. As depicted in Fig 1, the Cluster Load Balancer (CLB) organizes all process of the balancing scheme by triggering data gathering, detecting imbalances, re-organizing the load, and performing the proper migration moves. In order to collect information about the resources and federates, the CLB accesses Monitoring Interfaces. A Monitoring Interface is responsible for retrieving information from federates through direct access to them or from all resources managed by a CLB through Grid services [13]. The collected information comprises computation load: CPU consumption for federates and processing queue for resources. After data is gathered, filtering is employed to remove unnecessary information that might wrongly influence the detection of imbalances.

In the architecture, the Local Load Balancer (LLB) acts an intermediate between the CLB and the federates. The LLB retrieves federates' load data for a CLB through its Local Monitor Interface and forwards migration commands from the CLB to the respective managed federates. The migration calls are all forwarded to their respective Migration Mechanism (MM), which controls all the required steps to perform a two-phase federate migration [6] [17] successfully and efficiently. The migration process consists in transferring a federate to a remote resource indicated by the migration call, but the procedure has to avoid to jeopardize the federate's execution flow and to minimize the time that a federate is suspended to migrate. To keep the simulation consistency, MM is required to properly manage the events or messages exchanged by a federate during its migration. To minimize the migration latency, the migration process is divided

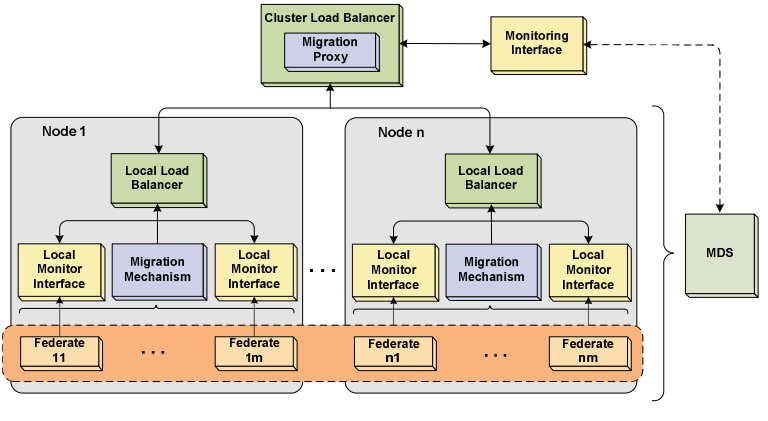


Fig. 1. The Dynamic Load Balancing's General Architecture

in two phases. In the first phase, the MM realizes the preparation for migration, which comprises the use of Grid services [13] to transfer the static files required to initialize the federate remotely. In the second phase, the MM retrieves the migrating federate's execution state, suspends the federate, restores the federate at the remote resource with the execution state and the simulation messages exchanged during its migration. For inter-domain migrations, a migration proxy is added in order to act as an intermediate component to allow the data transfer in the second phase of migration.

Even though inter-domain load reorganization procedure is distributed, the balancing system is structured hierarchically inside a cluster (CLB). A LLB works in each managed resource and is represented as a leaf in the hierarchy. A CLB is responsible for a set of resources and interacts with other neighbour CLBs to redistribute load. Instead of the balancing scheme in [8], the hierarchical structure does not necessarily need to have a root CLB to centralize data gathering and redistribution of load. According to the proposed scheme, the CLBs transfer to each other only relevant information for determining imbalances and generating load moves. Due to heterogeneity of resources, a benchmark is required to properly identify imbalances among resources with different capacities through a normalization of the gathered data sample for the local or the inter-domain load reorganization.

Therefore, this distributed design allows the load balancing to be performed even if some elements fail. If the root CLB in the centralized approach stops working, the entire balancing system is compromised. On the other hand, if a CLB fails, other CLBs continue working properly. Moreover, the centralized redistribution algorithm requires periodical synchronization with the central CLB, introducing delays and processing overheads. In the proposed distributed solution, CLBs interact with other neighbour CLBs asynchronously.

3.2 Distributed Load Redistribution Algorithm

The proposed balancing algorithm is divided in two scopes: local and inter-domain. The redistribution in the local scope is responsible for reorganizing the load evenly among the shared resources managed by a CLB. However, the balancing of resources at this scope is not able to detect imbalances or utilize all available computation power of shared resources outside the domain of the local CLB. Thus, an inter-domain balancing is introduced to allow the communication and the load analysis between between different domains.

Local Scope: in the local balancing, the redistribution occurs as an exchange of load between the resources. Similarly to the technique proposed in [8], the balancing is divided in monitoring, redistribution, and migration. Following the local balancing algorithm, a CLB collects load information from managed resources, filters it, and classifies the resources as overloaded or underloaded. After that, the CLB analyzes pairs of resources; it identifies migration moves according to the load differences between an overloaded resource and an underloaded resource. As presented in Algorithm 1, this pair-match analysis uses some load

Algorithm 1. Pair-Match Evaluation Algorithm

```
Require: src_rsc, dst_rsc
  if dst_rsc < min then
    if number_fed(src_rsc) > 1 then
      create_migration_move(src_rsc, dst_rsc)
    else if number_fed(src_rsc) = 1 & src_rsc > (min * 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
```

rules to determine the need of federate migration. After the moves are generated, they are sent to their respective federate to perform the needed load changes.

Inter-Domain Scope: in the inter-domain balancing, the redistribution manages the load differences between domains in order to gradually achieve a system-wide balancing, avoiding synchronization, overhead, and one point of failure. In this case, the redistribution is performed between pairs of CLBs. In order to facilitate the balancing when the system is overloaded, the inder-domain is triggered by an overloaded CLB, in a push fashion. Through the balancing scheme, an overloaded CLB selects a neighbour CLB that represents the proper location to send local load. Consequently, in a inter-domain balancing, the receiver CLB faces redistribution as an addition or submission of new load to its domain.

In this work, a CLB is configured with a list of neighbour CLBs when it initiates. This list does not include all balancing CLBs, but every CLB needs to be connected with other CLB in order to perform inter-domain balancing and exchange load. As a result, the balancing effectiveness and time to achieve even load distribution grows proportionally with the connectivity degree of CLBs in the system.

According to Algorithm 2, the inter-domain balancing is necessarily performed after the local balancing. Through this policy, more priority is given to local analysis to enforce that an external balancing is triggered only if the local redistribution algorithm has balanced intra-domain load or it cannot improve the local load distribution. Thus, the inter-domain is initialized if the local balancing does not detect load imbalances that are large enough to generate federate migrations.

Furthermore, a distribution factor is introduced to influence the frequency in which the inter-domain balancing is initialized. In order to regulate the relations between CLBs, this factor is introduced to reflect the need to perform distributed balancing, indicating the degree of imbalances among domains. The factor increases as the number of migration moves increases, what evidences imbalances between two CLBs and the need for more inter-domain load changes. As initial value, the factor starts in the balancing system with a high value to force inter-relations between CLBs; the initialization value comprises 100%. At the end of each balancing cycle, the relation factor needs to be adjusted in order to conform

Algorithm 2. Distributed Dynamic Load Balancing Algorithm

```
loop
  loads ⇐ query_MDS()
  current_loads ⇐ filter_MDS_data(loads)
  current_loads ⇐ normalize_loads(current_loads, benchmark)
  overload_cand ⇐ select_overload(current_loads)
  spec_loads ⇐ request_LLBs(overload_cand)
  mng_loads ⇐ filter(current_loads, spec_loads)
  mean, bds ⇐ calculate_mean_bds(mng_loads)
  over, under ⇐ select(mng_loads, mean, bds)
  mig_moves ⇐ redistribute_local(mng_loads)
  send_migration_moves(mig_moves)
  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
    overloaded_resources ⇐ filter_resources(extStD, localStD)
    overloaded_resources ⇐ order_resources(selectionParameter)
    federates ⇐ select(spec_loads, overloaded)
  end if
  send_to_neighbour(overloaded_resources, federates)
  migration_moves ⇐ wait_for_migration_moves()
  send_migration_moves(migration_moves)
  adjust_factor(relFactor, overloaded_resources, migration_moves)
  wait( Δ )
end loop
```

with the real load status between a CLB and its neighbour CLBs. The adjustment, as described in Function 1, is realized by using the migration success rate, which comprises the ratio between the number of generated migration moves and the number of resource candidates for migration ($extmig/extcadrsc$).

$$fac_n = -(\alpha_{n-1}^2) + \alpha_{n-1} + \alpha_{n-1} * fac_{n-1}, n > 0 \quad (1)$$

In order for the inter-domain balancing to be performed, inter-relations between CLBs are introduced, as described in Algorithm 2. These inter-relations reflects the balancing needs of each CLB, inducing the balancing system to proactively react to internal overload and to properly trigger the detection of inter-domain imbalances. Consequently, based on the average load and standard deviation analysis, the inter-relations between a local CLB and an external CLB develop the following situations:

– The first situation is identified through the inequality $CLB_{external}.load < CLB_{local}.load + \alpha$. This comparison evidences that, in average, the remote CLB (set of resources) presents less load than the local CLB, and it determines the need of moving load to remote resources in order to achieve a more even distribution of load in the entire balanced system.

- The second situation is represented by the following two comparisons $CLB_{external}.load \geq CLB_{local}.load + \alpha$ and $CLB_{external}.StD \geq CLB_{local}.StD + \beta$. Such analysis show that the remote CLB presents more load than the local CLB, but it presents a larger standard deviation for its average than the local CLB does. The standard deviation evidences the variation of load based on the average, and consequently how well the load is distributed on the resources. Therefore, even presenting more average load, there might be remote resources able to receive federates from the local set of resources.
- The third situation, described by the formulas $CLB_{external}.load \geq CLB_{local}.load + \alpha$ and $CLB_{external}.StD \leq CLB_{local}.StD+\beta$, evidences the case in which the local CLB presents less load than the external CLB, containing a larger variation of load. This standard deviation shows that the existence of underloaded resources at the remote location is unlikely. Thus, no load is transferred from the local CLB to the remote CLB, or based on further analyses, the local CLB can request load in order to equilibrate more the distribution of load.

As described in Algorithm 2, the general balancing scheme attempts to normalize the load locally in a domain before any external analysis is initialized; consequently, the local balancing is performed first in order to detect internal imbalances, and after further analysis is realized to detect the need for external balancing. For both balancing scopes, data is required in order to detect load discrepancies that indicate the presence of imbalances.

According to the three situations described previously and to Algorithm 2, the inter-domain redistribution algorithm detects load imbalances and select resources candidates for further remote balancing analysis. In the first step, an analysis of retrieved neighbour CLB's load is performed. Initially, an ordered list of underloaded CLBs is created; the list contains all CLBs with less load than the local CLB. If this list is not empty, the first underloaded CLB is chosen to receive federates from the local CLB. Instead of selecting only the first underloaded CLB, the balancing scheme can distribute the additional local load to a set of CLBs. The overloaded resource candidates and their respective federate candidates are selected based on the load of a chosen CLB. If the list of underloaded CLBs is empty, a filtering is applied to generate another list of CLBs, which present a standard deviation value larger than the local CLB does. This condition indicates that neighbour CLBs might contain underloaded resources that are hidden by the calculation of average load of the set of resources in a domain and can contribute to evenly redistribute the load. After, the parameter $(CLB_{external}.load + CLB_{local}.load)/(CLB_{external}.load + CLB_{local}.load)$ is employed to order the filtered list of CLBs. The first CLB in this list is selected, and the overloaded resource candidates are identified for migration move based on its average load.

After the overloaded resources are received by the remote neighbour CLB, it initializes its analysis to accept the receiving load. As with the load balancing, the remote CLB performs pair-match comparisons in order to identify possible load transfers. First, the CLB orders the receiving resources descendingly and its

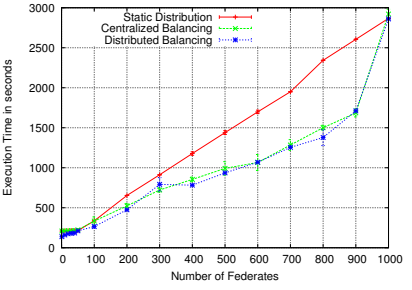
local resources ascendingly. After the ordering, each external resource is matched with a local underloaded resource through load comparisons, as described in Algorithm 1. If load difference between the resources evidence imbalance, a federate migration move is created, using the federate that was previously selected by the triggering CLB. Finally, a set of migration moves is produced as a result of the external resources analysis, and then this set of migrations are sent back to the calling CLB to be processed properly.

4 Experiments and Results

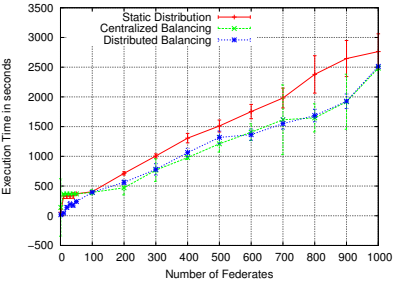
Experiments have been realized in order to evaluate the performance of the proposed distributed balancing scheme when compared with the centralized approach [8]. The experiments were run on a distributed environment composed by two computing clusters that were connected by a fast-Ethernet network link. One cluster was composed of 24 computing servers connected with a Myrinet optical network that provided data transmission up to 2 gigabits per second, and each server presented a Quadicore 2.40GHz Intel(R) Xeon(R) CPU and 8 gigabytes of RAM memory. The other cluster consisted of 32 computing servers connected with a gigabit Ethernet network, and each server contained a Core 2 Duo 3.4 GHz Intel(R) Xeon(R) CPU and 2 gigabytes of RAM. Linux operating system was installed in both clusters, and the HLA architecture with RTI version 1.3 was employed to coordinate the experimental simulations.

In the experiments, all the components were evenly allocated on the distributed system. Each LLB was set in every computing server, and one CLB was placed in each cluster. Moreover, the simulation elements were evenly distributed on all the shared environment, and one server was dedicated to run the HLA RTI executive. The simulation scenario comprised the coordination of training operations in two-dimension routing space. Two teams of interactive tanks were coordinated by federates in time-stepped simulations. The simulation consisted of 1 to 1000 federates, and each federate controlled the positioning of 1 tank. The movement of each tank was calculated by a federate, which published the tank's positioning and subscribed for a interest region. The movement calculation consisted in computing intensive processing that completely consumed a processor's resources for a whole simulation time step. External background load comprised a process that completely consumed the processing time of three CPU cores. For dynamic scenarios, in each 40 seconds, the simulation load was randomly selected to decreased its processing intensity to simple calculations, which determined the movement based on speed and space each 40 seconds, while the external load was randomly moved among resources each 40 seconds.

Four experiment have been accomplished, two evaluates the performance gain provided by the proposed balancing scheme and two evaluates the reaction capacity of the scheme. In the performance gain analysis presented in Figures 2a and 3a, the distributed balancing is compared with a centralized balancing technique, and with a baseline. As shown in the graphs, the distributed balancing approach presents a performance improvement similar to the centralized approach, even



(a) Constant Load

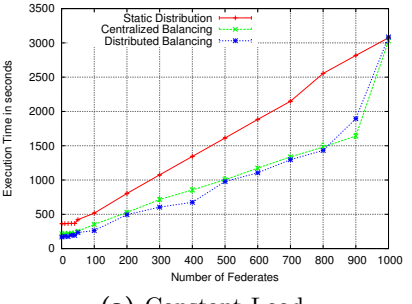


(b) Dynamic Load Changes

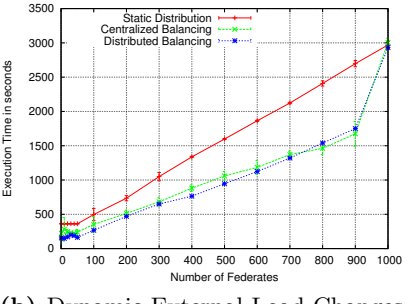
Fig. 2. Performance Gain and Reaction Analysis Between the Distributed Balancing and the Centralized Balancing for an Increasing Number of Federates

with the analysis with an external background load. However, the distributed redistribution presents larger number of migrations, which is shown in Figure 4a 4c respectively. This increase of migrations is caused by the inter-domain relations between the two CLBs, which produced intra-domain load imbalances in both local and remote domains.

The reaction analysis observed the capacity of the proposed balancing scheme to detect dynamic imbalances and to generate the proper load modifications. As shown in Figures 2b and 3b, the distributed approach reduced the simulation execution time equivalently to the centralized approach. In Figure 2b, the oscillations in the curves are caused by the instability of the simulation load, which constantly changed during run-time and were detected accordingly. As in the case of performance gain analysis, the distributed balancing presented considerable larger number of migrations than the centralized balancing, as depicted in Figures 4b and 4d respectively. However, in all the experiments, the curves converge to the same point when simulation contain 1000 federates; this convergence occurs due to the load saturation of the distributed system, reflecting in a few migrations.



(a) Constant Load



(b) Dynamic External Load Changes

Fig. 3. Performance Gain and Reaction Analysis Between the Distributed Balancing and the Centralized Balancing with External Background Load

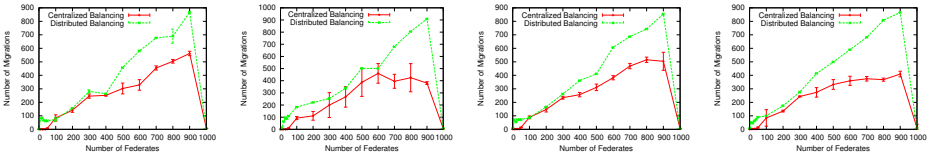


Fig. 4. Comparison of Number of Migrations for Each Experiment

5 Conclusion

In this paper, a distributed, dynamic balancing scheme is proposed for large-scale HLA-based simulations. This scheme aims to introduce aspects that were not addressed by the centralized approach, such as to provide fault tolerance, to minimize overhead, and to avoid bottlenecks. The proposed balancing scheme constantly monitors, redistributes, and moves load that can dynamically change during run-time. Grid services are employed in the design to obtain information and to perform federate migrations. The experiments showed that the proposed scheme presents performance gain similar to centralized approach's, but it uses a considerable larger number of migrations to achieve the same results. Thus, as future work, further analysis needs to be realized in order to adjust the inter-domain relations and to minimize the number of migrations, and additional larger scale experiments are necessary to evaluate the synchronization overhead caused by the centralized approach.

References

1. Ajaltouni, E.E., Boukerche, A., Zhang, M.: An efficient dynamic load balancing scheme for distributed simulations on a grid infrastructure. In: Proc. of the Int. Symposium on Distributed Simulation and Real-Time Applications, pp. 61–68. IEEE Comp. Society, Los Alamitos (2008)
2. Avril, H., Tropper, C.: The dynamic load balancing of clustered time warp for logic simulation. In: Proc. of the Workshop on Parallel and Distributed Simulation, pp. 20–27. IEEE Comp. Society, Los Alamitos (1996)
3. Bononi, L., Bracuto, M., D'Angelo, G., Donatiello, L.: An adaptive load balancing middleware for distributed simulation. In: Workshop on Middleware and Performance (WOMP), pp. 864–872 (2006)
4. Boukerche, A.: An adaptive partitioning algorithm for conservative parallel simulation. In: Proc. of the Int. Parallel and Distributed Processing Symposium, pp. 133–138. IEEE Comp. Society, Los Alamitos (2001)
5. Boukerche, A., Das, S.K.: Dynamic load balancing strategies for conservative parallel simulations. In: Proc. of the Workshop on Parallel and Distributed Simulation, pp. 32–37. IEEE Comp. Society, Los Alamitos (1997)
6. Boukerche, A., Grande, R.E.D.: Optimized federate migration for large-scale hla-based simulations. In: Proc. of the Int. Symposium on Distributed Simulation and Real-Time Applications, pp. 227–235. IEEE Comp. Society, Los Alamitos (2008)

7. Boukerche, A., Tropper, C.: A static partitioning and mapping algorithm for conservative parallel simulations. In: Proc. of the Workshop on Parallel and Distributed Simulation, pp. 164–172. IEEE Comp. Society, Los Alamitos (1994)
8. Boukerche, A., Grande, R.E.D.: Dynamic load balancing using grid services for hla-based simulations on large-scale distributed systems. In: Proc. of the Int. Symposium on Distributed Simulation and Real Time Applications, pp. 175–183. IEEE Comp. Society, Los Alamitos (2009)
9. Burdorf, C., Marti, J.: Load balancing strategies for time warp on multi-user workstations. The Computer Journal 36(2), 168–176 (1993)
10. Cai, W., Turner, S.J., Zhao, H.: A load management system for running hla-based distributed simulations over the grid. In: Proc. of the Int. Workshop on Distributed Simulation and Real-Time Applications, pp. 7–14. IEEE Comp. Society, Los Alamitos (2002)
11. Carothers, C.D., Fujimoto, R.M.: Background execution of time warp programs. In: Proc. of the Workshop on Parallel and Distributed Simulation, pp. 12–19. IEEE Comp. Society, Los Alamitos (1996)
12. Deelman, E., Szymanski, B.K.: Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. In: Proc. of the Workshop on Parallel and Distributed Simulation, pp. 46–53. IEEE Comp. Society, Los Alamitos (1998)
13. Foster, I., Kesselman, C., Nick, J.M., Tuecke, S.: Grid services for distributed system integration. Computer 35(6), 37–46 (2002)
14. Gan, B.P., Low, Y.H., Jain, S., Turner, S.J., Hsu, W.C.W.J., Huang, S.Y.: Load balancing for conservative simulation on shared memory multiprocessor systems. In: Proc. of the Workshop on Parallel and Distributed Simulation, pp. 139–146. IEEE Comp. Society, Los Alamitos (2000)
15. Glazer, D.W., Tropper, C.: On process migration and load balancing in time warp. IEEE Transactions on Parallel and Distributed Systems 4(3), 318–327 (1993)
16. Grande, R.E.D., Boukerche, A.: Dynamic partitioning of distributed virtual simulations for reducing communication load. In: Proc. of the Int. Workshop on Haptic Audio visual Environments and Games. IEEE Comp. Society, Los Alamitos (2009)
17. Li, Z., Cai, W., Turner, S.J., Pan, K.: Federate migration in a service oriented hla rti. In: DS-RT 2007: Proceedings of the Int. Symposium on Distributed Simulation and Real-Time Applications, pp. 113–121. IEEE Comp. Society, Los Alamitos (2007)
18. Luthi, J., Grossmann, S.: The resource sharing system: dynamic federate mapping for hla-baseddistributed simulation. In: Proc. of the Workshop on Parallel and Distributed Simulation, pp. 91–98. IEEE Comp. Society, Los Alamitos (2001)
19. Peschlow, P., Honecker, H., Martini, P.: A flexible dynamic partitioning algorithm for optimistic distributed simulation. In: Proc. of the Workshop on Parallel and Distributed Simulation, pp. 219–228. IEEE Comp. Society, Los Alamitos (2007)
20. (SISC), S.I.S.C.: Ieee standard for modeling and simulation (m&s) high level architecture (hla) framework and rules. IEEE Comp. Society (September 2000)
21. Tan, G., Lim, K.C.: Load distribution services in hla. In: Proc. of Distributed Simulation and Real-time Applications, pp. 133–141. IEEE Comp. Society, Los Alamitos (2004)
22. Wilson, L.F., Shen, W.: Experiments in load migration and dynamic load balancing in speedes. In: Proc. of the Winter Simulation Conference, pp. 483–490. IEEE Comp. Society, Los Alamitos (1998)
23. Zajac, K., Bubak, M., Malawski, M., Sloot, P.: Towards a grid management system for hla-based interactive simulations. In: Proc. of the Int. Symposium on Distributed Simulation and Real-Time Applications, pp. 4–11. IEEE Comp. Society, Los Alamitos (2003)