Optimized Federate Migration for Large-Scale HLA-based Simulations *

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

Abstract

Federate migration is a fundamental mechanism for large-scale distributed simulations. It provides the means for simulation load-balancing and thus improves the simulation's overall performance. Given its importance for simulations, several federate migration approaches have been proposed in the literature. Some approaches freeze the entire simulation, others use third-party mechanisms to transport data, and others make use of unnecessary communication and computing. Thus, in order to minimize the time spent on federate migration, we introduce a simulation agent that manages the migration steps, as well as the migrating federate's communication with the other simulation entities. The use of the agent simplifies the message management transparently and avoids redundant usage of network and computing resources. We demonstrate through simulation experiments that our approach decreases federate migration latency, improving the performance of HLA simulations that run over large-scale environments.

1. Introduction

High Level Architecture (HLA) is a standard initially defined by the United States Department of Defense in order to perform distributed simulations for military purposes, and it became an open IEEE standard in 2000 [9]. HLA introduces interoperability and reusability in distributed simulations. Interoperability is achieved when the same set of standards are used in all elements that compose a simulation, and reusability stems from the separation of simulations into independent, stand-alone components. According to the HLA specification [9], a simulation, or federation, is composed of federates and a Run Time Infrastructure (RTI). The RTI provides management services that organize the distributed simulation, coordinating operations and data exchange in order to keep the simulation's consistency. The RTI needs to conform to the HLA specification, but it is not part of the specification.

However, the HLA framework is not appropriate for large-scale distributed simulations because it does not solve issues of scalability, dynamic load-balancing and fault tolerance. If scalability is a system's ability to provide services of a certain quality level when its load increases, HLA does not deal with communication overhead, network latency, service discovery, or resources management. HLA also does not transfer processes between resources, which would balance the system load on heterogeneous distributed resources and consequently increase simulation's performance. HLA does not address fault-tolerance, so the entire simulation can be lost if a single failure occurs in the distributed system. Such issues must therefore be solved in order for large-scale HLA simulations to be supported.

Many approaches are found in the literature proposing to solve some of the issues in large-scale distributed simulation. Some approaches, such as those introduced by [16] [13] [18] [5] [12], attempt to solve scalability issues. Other approaches focus on load balancing issues, such as [16] [14] [7] [6]. Finally, the works of [7] aim to provide fault tolerance in large-scale HLA simulations.

Focusing on load balance and fault tolerance, federate migration i as a tool that generates dynamic deployment of federates on distributed systems. Such deployment provides the means for solving issues of distributed resource and federate heterogeneity, lack of resource reliability, and fault unpredictability.

Therefore, besides being essential in large-scale HLA simulations, federate migration influences the simulation's performance in proportion to the time spent on it. The interactivity of federates in a simulation leads to an interdependency among them. Even if the entire simulation does not stop during a federate migration, a set of federates dependent on the migrating federate will stop. As a result, federate migration must be improved in order to minimize its latency in simulations.

Given the importance of federate migration, several approaches have been presented in the literature [7] [17] [2]

*This is work is partially supported by NSERC, the Canada Research Chair program, ORF funds, and EAR Research Award.

[15] [11] [10]. These approaches aim to maintain transparency, simulation's consistency, and simulation's high performance. Transparency, which consists of hiding migration from the simulation code, is achieved by minimizing modifications in the HLA architecture. Consistency is kept by preventing events from being lost or duplicated during the migration process. Meanwhile, performance is dependent on how long and how many resources are spent during a migration.

In this paper, we therefore propose a new federate migration scheme that requires less time and resources. Based on previous approaches, our scheme ofers higher simulation performance by preventing unnecessary processing and message exchange. In order to maintain certain transparency, keep consistency and minimize the simulation's overload, we propose adding to our migration scheme a simulation agent that manages federate migration transparently to avoid unnecessary messages and computing.

The remainder of the paper is organized as follows. In section 2, the related work is described and existing issues are identified. In section 3, the federate migration using simulation agents is introduced and described. In section 4, four test-case groups of experiments are outlined, and the experimental results are shown and discussed. Section 5 presents the conclusion and directions for future work.

2. Related Work

Migration is ken in several areas, such as process migration and mobile agents' migration in parallel and distributed computing. A mobile agent, composed of an execution code and data, is expected to migrate autonomously, to adapt to different environments, and to recover its execution state seamlessly [3]. Thus, in essence, mobile agents are interoperable, and they consequently improve transparency in distributed systems by minimizing the migration effects on other systems' peers.

Similarly, in the literature, process migration involves the transfer of an executable code and data. Moreover, as proposed by Artsy and Finkel in [1], a process migration is accomplished in three phases: negotiation, transfer, and establishment. The negotiation phase entails the decision to receive a process and allocate resources for it. In the transfer phase, the process's virtual space and its communication links at the source are copied to the destination. As in the transfer phase, the process execution in the establishment phase is recovered, and the involved parts are informed about its completion.

Federate migration approaches mix both process and mobile agent migration techniques, following the phases of process migration and incorporating the interoperability and transparency features of mobile agents. However, such approaches differ in terms of how the information is transferred, how the federate is stopped, and how much the simulation is modified.

Lüthi and Groβmann [7] developed a solution for large-scale distributed simulations using HLA and their own resource-sharing system. In their simulation system, a simulation manager and a differentiated federate (communication federate) are used in the migration process. The manager administers the federate's startup at the new location, as well as the saved state's transfer. The communication federate transmits orders to the simulation's federate in order to save and restore the state, and thus freezes the entire simulation.

Zajac et. al. [17] present a system for running HLA simulations over Grid environments. In order to introduce such a system, a Migration Library is proposed as a working interface between the HLA simulation and the Grid services. The Migration Library facilitates access to the HLA API, helps to save, and restore states and provides an interface between the author's system and the federate's code. The federate state is saved and restored through the HLA specification methods, and the transfer of the federate's state is accomplished using the GridFTP.

Cai et. al. [2] developed a Load Management System (LMS) to support the execution of HLA-based large-scale distributed simulations. The most important feature in the proposed system is federate migration. Such a migration is accomplished by freezing the entire simulation in order to avoid consistency issues. Moreover, Grid FTP services are used to transport the migrating federate's state.

All the presented solutions use the save and restore methods provided by the HLA standard. Although these methods guarantee simulation consistency by not allowing messages to be exchanged during the migration, they freeze the entire simulation. Therefore, to minimize migration latency, a freeze-free migration is required.

The approach presented by Tan and Lim [10] accomplishes load balance of HLA simulations. Its architecture consists of a federate wrapper and a load distribution system. The load distribution system monitors federates and determines federate migration. The federate wrapper controls the federate's execution through SugarCubes[1], which stops and resumes a federate, and JavaGo[2], which migrates a process to keep its state. Moreover, the messages received during migration are stored in queues and transmitted through publication and subscription to a special simulation region.

A common drawback in the previous approaches is the requirement of third-party mechanisms. The use of an external tool to transfer data increases overhead for the migration process [15]. Peer-to-peer communication thus aims for migration.

[1] http://homepage.mac.com/jeanferdinandsusini/SugarCubes.html
[2] http://homepage.mac.com/t.sekiguchi/javago/index.html

Yuan et. al. [15] introduce a migration mechanism that is based on SimKernel and that minimizes migration latency by focusing only on application-level federate migration. During the migration process, the federate's state is saved and transferred together with its code and a shadow federate is initiated on the destination node. The proposed solution is freeze-free and does not use third-party mechanisms. However, the simulation needs to be designed according to the SimKernel framework, and processing in order to eliminate duplicated events is necessary for event consistency.

Tan et. al. [11] introduce HLA federate migration with strong mobility - migration keeping process state. In order to minimize migration latency, the proposed system is freeze-free for stopping a federate, ans saving and restoring its state, uses peer-to-peer communication for data transfer, and ensures event consistency. The solution creates a new federate that joins the federation, restores its state, and publishes and subscribes for the same objects as before, while applying mechanisms for event consistency.

In the approaches of both Yuan et. al. [15] and Tam and Lim [10], unnecessary messages must be sent in order to manage each federate. This management involves the joining of a new federate, the resignation of the old federate, and publications and subscriptions to same objects. Duplicated messages are thus sent to both federates, and additional computing is required to eliminate these messages. Thus, in order to avoid unnecessary management and computing, we introduce a simulation agent that improves the migration process.

3. Federate Migration with Simulation Agents

The simulation agent is introduced in the simulation in order to facilitate federate migration management and to decrease latency. Decreasing migration latency improves simulation performance running on a large-scale distributed environment. Thus, our federate migration process is based on some concepts presented by previous federate migration approaches, such as avoiding third-party mechanisms for data transfer or using shadow federates.

In this paper, the simulation agent creates a shadow agent that participates in the management of the federate migration actions. As depicted in figure 1, the simulation agent works as a intermediate layer between the federate and the RTI, managing all the communication and eventually required migrations. Each federate, as well as the RTI, has an agent for communication. The simulation components access the agent's interface in order to transmit or receive messages. The simulation agent works as a communication layer in the architecture, and consequently, federates and the RTI call on its methods instead of directly accessing the network layer.

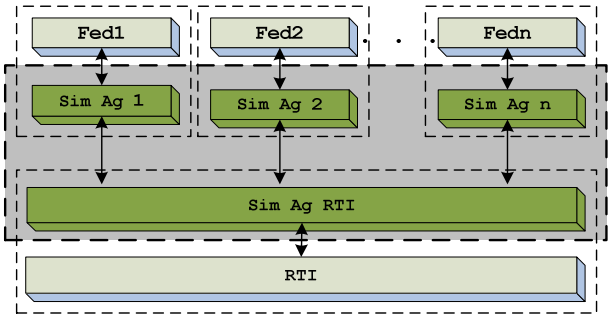

Figure 1. General view of simulation agent in HLA simulation

3.1. Simulation Agent Architecture

As explained in figure 2, the simulation agent is placed between the network layer and the simulation layer. The agent does not determine when federate migration should happen. It only transfers the federate to the location requested by the load balance system. To achieve migration, the simulation agent launches another simulation agent at the remote location, and they communicate to perform migration. The agents' architecture is composed of a Communication Manager and a migration manager that work independent of each other.

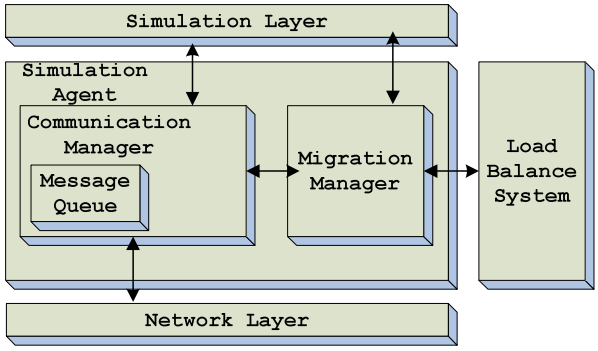

Figure 2. Architecture of simulation agent

Because HLA simulations access the simulation agent in order to perform communication, unique identifiers are assigned to simulation entities by the simulation agent to forward messages properly to them. Also, when a migration process is not required, the simulation agent simply forwards the message to the communication layer; conversely, if a migration process is required, the message is forward to the federate or the RTI.

The main task of the Load Balance System in our design is to determine federate migration in order to distribute load over the distributed system's nodes. Based on system analyses, it triggers migration and determines where it must migrate to.

The communication manager is responsible for maintaining the federate's communication while the federate is migrating. It forwards messages to the federate and stores them when they cannot be processed because the federate is moving to another host. As seen in 2, the communication manager contains a queue that is used as the storage structure for such messages. The communication manager interacts with the federate and the network layer to forward messages, store them, and restore them at the federate's new location. Furthermore, whenever a migration process is started in the simulation agent, the communication manager is triggered by the migration manager in order to store the incoming messages.

The migration manager is responsible for all the actions taken during or for migration. It triggers the communication manager and requests the federate's simulation state by calling the *federateSave* method. The migration manager launches the simulation manager at the remote node designated by the load balancing system, and communicates with its respective migration manager at the new location node in order to transmit the federate's state and the received messages. Finally, the migration manager acts at the remote location to restore the federate's simulation status at the same point it stopped for migration.

3.2. Save and Restore State Methods

Although transparency should be an integral part of the design, some modifications in HLA and its federates are required in order to minimize the latency caused by federate migration. The HLA specification [9] provides two interfaces called *federationSave* and *federationRestore*, which are respectively used to save and restore the entire simulation state. Because they introduce global synchronization of the simulation, their usage decreases the simulation performance badly. There also exists the transparent state-saving mechanism introduced by Santoro and Quaglia [8], but this mechanism stops the federate's run abruptly, leading to simulation inconsistencies.

Thus, as a matter of simplification, such interfaces and mechanism are not used, and new methods, like those proposed in previous works [10] [2] [15] [11], are introduced to save and restore a specific federate state without interfering with the rest of the simulation. In our design, the new methods are called *federateSave* and *federateRestore*. The methods' functioning is determined by whoever designs the federate, so inconsistencies are prevented during the saving process, and computing is avoided when restoring.

In a federate migration scenario, during migration and after the *federateSave* method is called, the migrating federate terminates its execution because it stops computing the received messages and producing new events to be sent to other federates. Unlike the approaches of Tan and Lim [10] and Yuan et. al. [15], which keep both old and new federates running in the simulation, the federate in our solution terminates without resigning the federation. This is because the simulation agent manages all communication. As a result, the federate's termination requires less computing and memory resources from the host where it is running, improving the performance of other federates on the same host.

3.3. Saving and Restoring Messages

As stated earlier, our federate migration approach accomplishes freeze-free federate *save* and *restore* actions. Even though this technique minimizes migration's latency, it complicates the migration process and requires additional mechanisms for simulation consistency. Thus, messages must be saved in a data structure, transmitted to the federate at the new location, and restored, manipulated, and processed in a consistent order.

Because all the exchanged simulation messages have to pass through the network layer, they can be treated as simple messages; the migration process in this step of saving and restoring messages can be accomplished in such a way that federates are visualized as processes. Thus, our approach eliminates all the consistency issues that arise when migration is treated on the simulation layer.

Because the entire simulation is not suspended for migration, other federates continue to send messages, which must be saved to be sent and accessed to properly restore the migrated federate. Following this technique, Yuan et. al. [15] save all the messages in a queue called *inQ*, which is sent together with the federate's state. Similarly, Tan and Lim [10] use several queues to save and restore incoming messages. Tan et. al. [11] keep both federates running while saving the incoming messages, which are then merged in the migrated federate.

Regarding simplicity in our approach, all the received messages are saved in a queue to be processed post-facto. The queue works much like the combination of the approaches *mailbox* and *full protocol*, which are used for process migration, as described by Heymann et. al. [4]. The simulation agent is responsible for saving messages in its queue and flushing them to the other simulation agent at the remote host. The messages are sent only after all the references to the federate's address are updated.

The next steps in the migration process are sending and restoring the received messages at the new location. These steps are accomplished differently than in the solutions pro-

posed in the literature [15] [10] [11]. In our approach, after all messages are saved, and after it is certain that no other message will be received, the simulation agent sends the messages to the respective simulation agent on the remote location. The agent transmits the messages only after it receives a notification message from the simulation agent at the new location. The simulation agent at the new location sends messages to other elements to update their addresses with the migrating federate's new location, and the simulation agent at the old location is informed after the procedure is completed. Then, after the notification message is received, the old simulation agent packs all the received messages and sends them to the new simulation agent. The new agent inserts all messages in its queue so they are processed first by the new federate.

3.4. Federate Migration Protocol

As shown in the sequence diagram in figure 3, the federate migration procedure starts when the migration manager receives a migration call from the load balance system. After the migration manager is called, it takes five well-defined and ordered actions to perform the migration consistently. First, the migration manager launches a new simulation agent at the location determined by the load balance system. This remote simulation agent is launched with a flag set to *migrating*. The federate's code is sent to the launched simulation agent that locally initiates the federate in restore mode. The federate initializes and waits for the saved state and the messages received while its migration was performed. In the simulation agent at the new location, the migration manager informs all the migration managers involved in the simulation's elements about the address change. The agent also incites the communication manager to store any messages in its queue, so that these can be merged with other messages received by the communication manager at the old location.

Secondly, in the old location and at the same time, the migration manager launches the simulation agent remotely and informs the communication manager that a migration process has just started. The communication manager stops forwarding messages to the federate and starts storing them in its queue.

In the third step, the migration manager calls the *federateSave* method from the federate in order to halt the federate's run and to retrieve the federate's state. The federate, after completing all processing in its simulation loop, saves all simulation variables that correspond to its state, answers to the manager with them, and halts its run. The local migration manager stores the state temporally while it waits for the remote migration manager's request for the state.

In the fourth step, the migration manager at the remote location requests the federate's state when the migrating

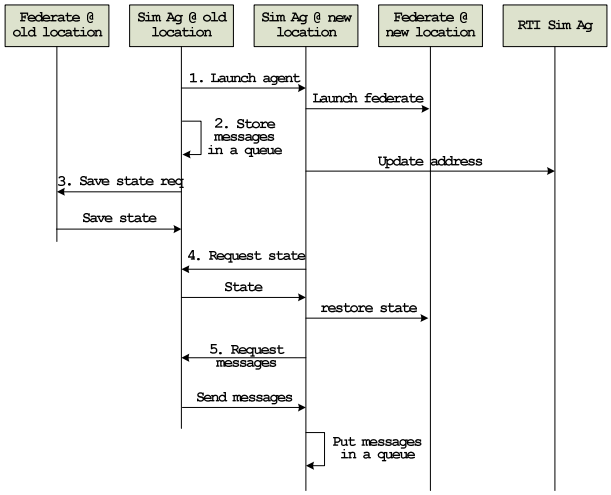

Figure 3. Federate migration protocol using the simulation agent

federate is ready to restore its own state. After the manager receives the state, it passes the state to the federate by calling the *federateRestore* method from the federate. Although the state request is shown here as the fourth step in this migration protocol, it can occur whenever the federate has completely launched at the new location.

Finally, in the fifth step, the migration manager at the new location requests the messages that were received during migration. This occurs when the address references to the federate are updated successfully. Such messages are transferred to the federate in the same order that they were received by the communication manager at the old location. After the messages are received at the new location, the migration manager passes them to the communication manager, which inserts them at the beginning of its queue. Next, the migration manager confirms that the restoring phase is complete, and that the federate can take over its processing by entering in its simulation loop. In addition, the communication manager also starts forwarding messages to the federate, processing those in queue first.

After all these steps are accomplished, the simulation agent at the old location finishes its run.

4. Experimental Results

Experiments were run to evaluate the benefits of our approach. For the experiments, we ran all simulations in a cluster composed by 32 nodes. Each node is composed by a Intel Core 2 Duo Xeon processor CPU that runs at 3.4 GHz and 2 gigabytes of DIMM DDR memory RAM, and they are

connected to each other through a gigabit ethernet network. Also, each node was intalled with the Linux operating system, and our experiments were run using HLA with the RTI version 1.3 performing communication through TCP/IP connections. As a result, the speed of our network presented communication latency from being an issue in our experiments.

Even though the experiments were run on a cluster with a high-speed network and not on a large-scale distributed system, the results were useful for the large-scale context. The proposed federate migration aims to minimize the computing and specially the communication used during migration. Thus, the results are relevant because they reflect a latency improvement for our experiments' environment, and they reveal more important implications for a larger scale environment.

Moreover, as our benchmark, an HLA simulation coded in C was used to conduct experiments and analyze the performance of our approach. The scenario for our experiments was a simulation of training operations of two teams of interactive tanks in a routing space. The tank effectuates random movements in two-dimensional space that is within range of its original location. In order to accomplish such simulation, each federate must publish the position update and subscribe to the simulated space area that is related to the tank's new position. Thus, even though the scenario is quite simple, the number of publication and subscription messages - that is, the communication load - is large, due the number of federates and the simulation's tanks.

In general, the federate migration procedure occurs when the simulation agent is triggered through a migration call at a given moment during the simulation. Such a call was induced in our experiments for purpose of analysis. In all simulation experiments, the simulation elements are distributed in proportion to computing resources. Moreover, to study the migration pattern more accurately, the call was triggered only once in each simulation run.

In order to provide trustworthy results, each plotted point in our graphs represents the average of 33 runs. Thus, the confidence intervals were calculated using the z-distribution at a confidence level of 95%.

To evaluate our approach, experiments were clustered in four test-case groups. In the first group, an analysis was made over the performance results of our federate migration approach. In the second test-case group, following the same approach presented by Tan and Lim [10], our solution was compared with existing federate migration approaches. The third group involved comparisons between a federate migration approach that makes use of RTI *resign*, *join*, *publish* and *subscribe* calls and a federate migration mechanism that avoids these calls, as pointed out in our solution. The fourth test case group highlights the benefits that federate migration introduces in an HLA simulation.

4.1. Federate Migration Performance Analysis

In this test case, the performance of the federate steps is analyzed in order to assess the delays that each one introduces to the entire migration process.

Figure 4 shows the federate migration's performance and some of the steps that were described earlier. In the graph, performance is analyzed for an increasing number of federates. The steps shown in the graph include launching the simulation agent and the federate in a remote node, transferring the federate's state, restoring the federate state, and transferring and restoring the received messages. The migration time increases slowly as the number of federates increases. The reason for this tendency is that the number of messages exchanged by federates while the migrating federate's run is not yet restored increases with the number of federates. This pattern is clearly identified in figure 4, which only shows how the step of retrieving and restoring messages changes significantly with the number of federates. The migration time increases to 0.012188 seconds, the time spent to retrieve and restore messages, when the number of federates increases from 2 to 15 in the simulation.

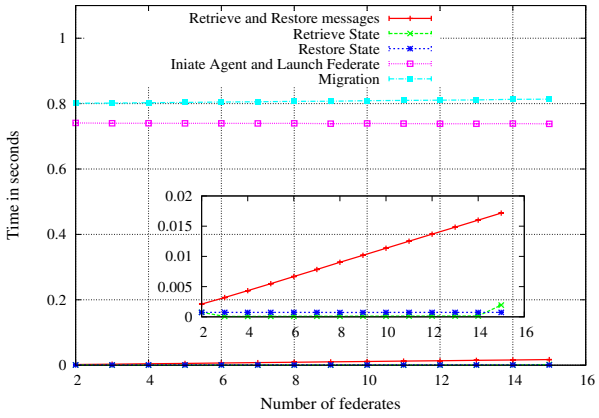

Figure 4. Federate migration performance overview with an increasing number of federates

Figure 5 shows that the number of objects did not influence the migration time in our experiments. Unlike the results seen in figure 4, the time spent retrieving and restoring received messages does not change in figure 5 because the number of federates was the same throughout the entire experiment. Because the amount of information to save the object's information is small in our implementation, the number of objects does present significant computing and

communication overhead in the simulation, as shown in the figure 5. Focusing on the retrieving state step, there is a slight increase in this curve as the number of federates increases. The curve's jitter is caused by the network oscillations, made noticeable by the tiny amount of data transferred. A more complex simulation scenario and a slower network could considerably increase this step's latency. As a result, all the curves presented in figure 5 show an imperceptible change in the federate migration time generated by data transfer.

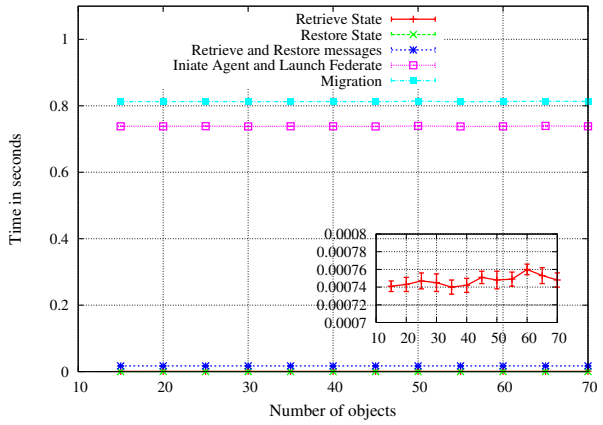

Figure 5. Federate migration performance overview with an increasing number of simulation objects

4.2. Performance Comparison

In order to compare our federate migration solution with other approaches, we followed the same performance experiment conducted by Zajac et. al. [17], Yuan et. al. [15], Tan and Lim [11], and Tan et. al. [10]. The number of federates in the simulation experiments varied from 2 to 6, but all of them were publishers and subscribers due to the tank movement scenario adopted in our experiments. Our scenario is slightly different from those of the cited authors because these authors used a simulation composed of one publisher and N-1 subscriber(s) in their scenario. Thus, our tank movement scenario was used in the comparison because it presents a complexity that is equal or greater than the complexity in other scenarios.

As shown in figure 6, our federate migration solution shows better performance results than those presented by Zajac et. al. [17], Yuan et. al. [15], Tan and Lim [11], and Tan et. al. [10]. When compared with Zajac et. al. [17], our approach shows a performance improvement of around 97%, revealing the large overhead brought to the simulation by a federation save/restore approach. Compared to Yuan et. al.'s approach [15], there is an improvement of around 95% in performance. The improvement is around 95.2% when compared to Tan and Lim's approach and around 35% compared to Tan et. al.'s [10]. Moreover, all previous comparisons except that with Yuan et. al.'s [15] show a considerable increase in migration latency. Such approaches show this increase in time either to using the save/restore federation technique or to spending time on unnecessary messages and computing. In the work of Yuan et. al. [15], the communication messages pass through *InQ* and *OutQ* queues, facilitating their management during migration. Similarly, our approach separates migration into simulation and communication realms to simplify the management of messages. This is exemplified by the 0.4% increase in our migration latency when the number of federates changes from 2 to 6 in the migration scenario. As a result, our solution scales better than the approaches presented by Zajac et. al. [17], Tan and Lim[11], and Tan et. al. [10].

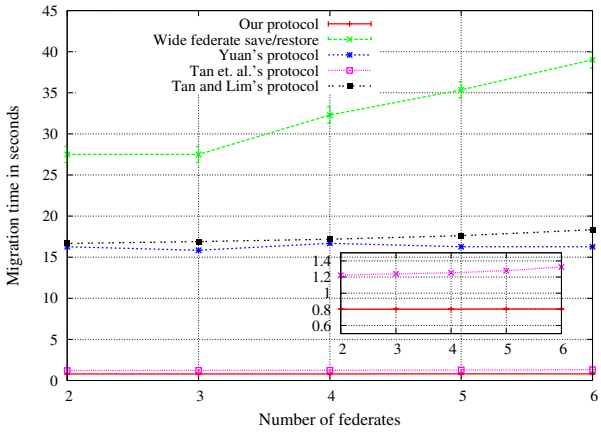

Figure 6. Performance comparison of federate migration designs

As stated above, we aim to produce federate migration that introduces minimal overhead to HLA simulations while maintaining transparency and simplicity. Thus, this section shows that the avoidance of resigning, joining, publishing and subscribing decreases federate migration latency. All messages received by the migrating federate are saved in order to be consumed afterwards, which easily maintains the simulation consistency. In order to identify this benefit, this test-case of experiments used a scenario composed of 15 federates that controlled 15 to 70 objects per simulation.

4.3. The Benefits of Avoiding Unnecessary Communication and Computing

For comparison purposes, figure 7 shows two curves that represent federate migration using resigning, joining, publishing and subscribing calls (PJPS) delimited by the HLA specification, and federate migration not using those calls (NPJPS). In the curves, it is possible to observe how much time our approach saves by not calling PJPS methods.

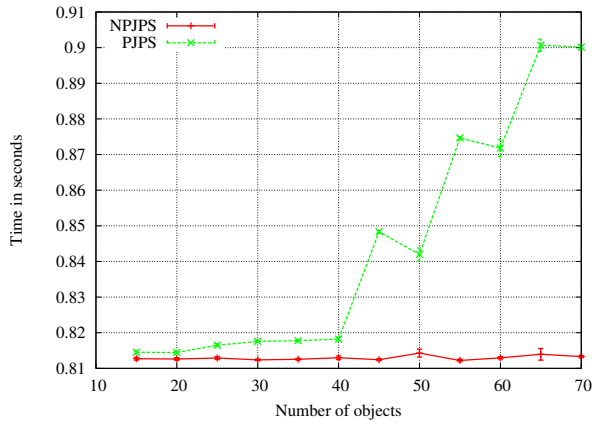

Figure 7. Comparison of PSPS and NPJPS federate migrations

It is noticeable in figure 7 that the NPJPS curve does not present significant changes when the number of objects increases, while the PJPS curve shows federate migration latency increasing clearly. This considerable difference occurs results from the communication and computation needed to publish objects and to subscribe to a region that is related to the published objects. For each tank's move in our scenario, the tank's position is published and it subscribes to an area of interest. Furthermore, the PJPS curve shows some points where migration latency slightly decreases, even with a an increasing number of objects. Because of the high precison of the time scale, even a jitter of milliseconds is noticeable in the results; this is shown when the number of objects increases from 45 to 50 and the federate migration latency decreases by 8 milliseconds. This jitter is caused by minor network oscillation.

Although the curve pictured in figure 7 shows a very small difference in time, it also reveals the importance of avoiding PJPS in federate migration for large-scale distributed simulations. The experiments were run on a cluster with a high-speed network, and communication did not compromise simulation performance. However, in large-scale simulations, simulation elements are widespread over the global network, and communication presents major issue for scalability. Thus, reducing the amount of unnecessary communication by avoiding PJPS calls in the federate migration minimizes overhead.

4.4. Federate Migration Benefits

This section shows how federate migration can benefit an HLA simulation. In the experiments, we ran time-stepped simulations composed of 10 federates controlling 30 objects (tanks). As a benchmark, we chose a cluster's node to run together with the federate's different loads in order to analyze the benefits of federate migration for load balance. A program was built to apply different loads on the node by launching a number of threads that were responsible for consuming computing resources.

First, to perform comparisons with a baseline, we ran a time-stepped HLA simulation without federate migration. We then ran our modified implementation, with federate migration taking place, and compared it with the original. In both cases, we ran our simulation for 500 time steps, and the migration was triggered after 10 simulation time steps.

As presented in figure 8, our federate migration implementation did not perform as well as the original simulation when the overhead on the node was less than 17%. The latency introduced by the migration was larger than the time the original simulation ran with overhead. However, when overhead increases, the simulation with federate migration presents better experimental results than the original, benefiting the entire simulation. Moreover, as the CPU load increases, the federate is migrated away from the heavily-loaded node. Our simulation therefore maintains a rather constant simulation time.

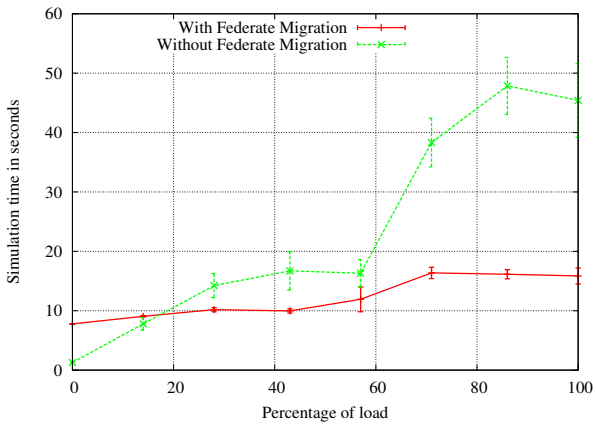

Figure 8. Benefit of federate migration while managing the simulation load

5. Conclusions and Future Work

In this paper, we have presented an optimized federate migration technique for HLA simulation aimed at large-scale distributed simulations. Federate migration is fundamental for balancing the load on distributed systems, and decreasing migration latency increases simulation performance. Thus, our objective has been to minimize federate migration latency. Our approach does not freeze the simulation, does not use third-party mechanisms, and avoids calling resign, join, publish and subscribe HLA methods in order to perform migration. Even though the experiments were not conducted in a large-scale environment, the results proved that a decrease in migration latency can greatly improve simulations in such an environment.

Our results show the relevance of federate migration for an HLA simulation. Also, and most importantly, we have shown that avoiding PJPS methods diminishes federate migration time considerably, even in a high-speed network. Moreover, in order to decrease migration latency and to maintain simulation consistency, federate save and restore methods should be implemented by the simulation designer.

In future work, we will perform RTI migration in order to balance the simulation; accordingly, we will perform simulation experiments in large-scale distributed systems in order to analyze the advantages of federate migration for different system characteristics. Also, we will introduce a load balance system to interact with and trigger our federate migration technique. Such a system will determine migration based on network distance between nodes, communication density, resource load, and federate processing requirements.

References

[1] Y. Artsy and R. Finkel. Designing a process migration facility: the charlotte experience. *IEEE Computer*, 22(9):47–56, 1989.

[2] W. Cai, S. J. Turner, and H. Zhao. A load management system for running hla-based distributed simulations over the grid. In *DS-RT '02: Proceedings of the Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, page 7, Washington, DC, USA, 2002.

[3] S. Fnfrocken. Transparent migration of java-based mobile agents: capturing and reestablishing the state of java programs. In *Proceeding of the Second International Workshop on Mobile Agents*, pages 26–37, 1998.

[4] E. Heymann, F. Tinetti, and E. Luque. Preserving message integrity in dynamic process migration. In *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing*, pages 373–381, 1998.

[5] R. P. B. Katherine L. Morse, David L. Drake. Web enabling an rti an xmsf profile. In *Proceedings of the IEEE 2003 European Simulation Interoperability Workshop*, Stockholm, Sweden, June 2003.

[6] T.-D. Lee, S.-H. Yoo, and C.-S. Jeong. Design and implementation of gpds. In *Workshop on HLA-Based Distributed Simulation on the Grid in the 4th International Conference Computational Science (ICCS 2004)*, pages 873–880, Krakw, Poland, June 2004.

[7] J. Lüthi and S. Groβmann. The resource sharing system: dynamic federate mapping for hla-based distributed simulation. In *PADS '01: Proceedings of the fifteenth workshop on Parallel and distributed simulation*, pages 91–98, Washington, DC, USA, 2001.

[8] A. Santoro and F. Quaglia. Transparent state management for optimistic synchronization in the high level architecture. In *Proceeding of the Workshop on Principles of Advanced and Distributed Simulation*, pages 171–180, 2005.

[9] 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.

[10] G. Tan and K. C. Lim. Load distribution services in hla. In *Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 133–141, 2004.

[11] G. Tan, A. Persson, and R. Ayani. Hla federate migration. In *Proceedings of the 38th Annual Simulation Symposium (ANSS05)*, pages 243–250, Washington, USA, 2005.

[12] G. Theodoropoulos, Y. Zhang, D. Chen, R. Minson, S. J. Turner, W. Cai, Y. Xie, and B. Logan. Large scale distributed simulation on the grid. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, page 63, Washington, DC, USA, 2006.

[13] Y. Xie, Y. M. Teo, W. Cai, and S. J. Turner. Service provisioning for hla-based distributed simulation on the grid. In *19th IEEE/ACM/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2005)*, pages 282–291, Monterey, California, USA, June 2005.

[14] Z. Yuan, W. Cai, and M. Y. H. Low. A framework for executing parallel simulation using rti. In *DS-RT '03: Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications*, page 12, Washington, DC, USA, 2003.

[15] Z. Yuan, W. Cai, M. Y. H. Low, and S. J. Turner. Federate migration in hla-based simulation. In *Workshop on HLA-Based Distributed Simulation on the Grid in the 4th International Conference Computational Science (ICCS 2004)*, pages 856–864, Krakw, Poland, June 2004.

[16] K. Zajac, M. Bubak, M. Malawski, and P. Sloot. Towards a grid management system for hla-based interactive simulations. In *DS-RT '03: Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications*, page 4, Washington, DC, USA, 2003.

[17] K. Zajac, M. Bubak, M. Malawski, and P. M. A. Sloot. Execution and migration management of hla-based interactive simulations on the grid. In *PPAM*, pages 872–879, 2003.

[18] S. Zhu, Z. Du, and X. Chai. Gdsa: A grid-based distributed simulation architecture. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, page 66, Washington, DC, USA, 2006.