Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. D. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

A PROCESSES MIGRATION MECHANISM FOR DISTRIBUTED SIMULATION PROGRAMS

Mateus Augusto F. C. Junqueira

Federal University of Itajuba
Itajuba, MG 37500-000, BRA

Emerson Assis de Carvalho

Federal Universisty of Itajuba
Itajuba, MG 37500-000, BRA

Edmilson Marmo Moreira

Federal University of Itajuba
Itajuba, MG 37500-000, BRA

Otavio Augusto S. Carpinteiro

Federal University of Itajuba
Itajuba, MG 37500-000, BRA

ABSTRACT

This paper presents migration mechanism for distributed simulation programs based on Time Warp protocol and a dynamic load balancing algorithm developed to evaluate the processes migration mechanism. During a migration, the processes migration mechanism finishes and recreates all simulation processes. Experimental results were achieved using the Message Passing Interface (MPI) and its resources. The results show that the proposed migration mechanism, along with the load balancing algorithm, can improve the simulation programs performance.

1 INTRODUCTION

The process developing of new projects needs development support tools. It is important to identify the most appropriate approach for a system development in order to predict how would be its behavior. In some situations, support tools are more suitable because prototyping may be difficult or inappropriate in according to time and cost (Fujimoto 2000).

With computer simulation is possible to model existing and/or conceptual systems. A simulation is a real-world process representation over time, with the generation of an artificial system history that allows infering real system characteristics (Banks, Carson, Nelson, and Nicol 2008). There are complex real-world problems that can only be analyzed through simulations (Law 2007). Simulations application examples are: military, entertainment such as games, networks area (hardware, software and/or protocols evaluation), digital circuits and computer systems development, transport such as traffic control, manufacturing production planning etc (Kassab, Tun, Arora, King, Ahmed, Miskovic, Cope, Vadhwana, Bello, Sevdalis, et al. 2011, Banks, Carson, Nelson, and Nicol 2008, Wainer G A 2011, Fujimoto 2000).

A Distributed System (DS) is a system formed by an independent computers collection (nodes), collaborating with each other and appear to its users as a single system. The scheduling processes is responsible for the processes allocation and reallocation, with the goal of balancing the load and achieve a better performance (Tanenbaum and Steen 2006).

To use a DS, the simulation program must be paralized which leads to difficulties absents in a sequential program, such as the need for LPs synchronization and load balancing. The synchronization is a topic widely studied due to the difficulty in reliably coordinating the LPs logical clocks. This difficulty results of the occurrence of inconsistencies that arise during simulation, called causality errors (Lamport 1978). In a sequential simulation, there is no need for explicit synchronization because, in sequential algorithms, the events ordering occurs naturally. There are two approaches to perform processes synchronization: the

conservative and the **optimistic**. The conservative is the one that prevents the causality errors occurrence. The optimistic, which allows these errors occurrence and presents mechanisms to treat them.

The Time Warp (TW) (Jefferson 1985) is the better know optimistic protocol for distributed simulations synchronization. The key concepts and mechanisms, such as rollback, anti-messages, local and global weather control simulation, used by optimistic protocols, were initially introduced by TW which each logical process (LP) can run any received event. If ocurrs a causality error (straggler message), it must undo the computation to a previous consistent state (rollback procedure). If the undone computation done by the LP sent messages to other LPs, the rollback procedure must cancel these messages (anti-messages mechanism).

To migrate an LP in execution means transferring its state between two computers in a DS, allowing the LP running can continue on the target computer from the same point where it was suspended on the source computer. The processes migration is essential for dynamic load balancing mechanisms, as it allows the LPs distribution modification at runtime. It may be implemented in kernel mode, with less portability, or in user mode, which allows for greater portability.

With the processes migration comes the need for dynamic scheduling algorithms that perform load balancing in real time, i.e., during the program execution in an attempt to balance the load. Static balancing is limited to certain applications types, being unable to deal with dynamic load variations present in many distributed applications (De Grande and Boukerche 2011).

This paper presents a mechanism to perform the collective processes migration in distributed simulation applications running on the TW protocol. In order to evaluate the proposed migration mechanism, an algorithm for dynamic load balancing, which used the migration engine to perform LPs dynamic allocation was developed.

The proposed migration mechanism may be useful in situations where the modeled system changes over time resulting in unbalance and fall distributed simulation performance due to increased stragglers messages and consequently anti-messages. It may also be useful for distributed systems subject to external loads, do not belonging to simulation program, which may generate unbalance in the simulation program.

2 RELATED WORK

There are many techniques used to optimize the distributed simulations performance. Among these techniques, those that stand out are the techniques for load balancing (Carothers and Fujimoto 2000). The literature presents many scheduling algorithms for distributed simulation, with different concepts and operating mechanisms (Reiher and Jefferson 1990, Burdorf and Marti 1993, Carothers and Fujimoto 2000, Low 2002, Jiang, Anane, and Theodoropoulos 2004, Peschlow, Honecker, and Martini 2007, El Ajaltouni, Boukerche, and Zhang 2008, De Grande and Boukerche 2011, Meraji, Zhang, and Tropper 2010, Chen, Lu, Yao, Peng, and Wu 2011). Many of them are dynamic approaches and use migration mechanisms to accomplish the processes distribution.

A well known problem is related to the LPs migration in TW, since each LP has a large number of saved states (necessary in case of rollbacks). A radical solution would be to perform a rollback to the global simulation clock (GVT), which would eliminate all the saved states and would not have the states migration overhead. However, this approach injures the optimistic simulation principle. Proposed solutions that address these problems can be found in (Carothers and Fujimoto 2000, Reiher and Jefferson 1990).

Numerous operating systems such as AMOEBA (Mullender, van Rossum, Tanenbaum, van Renesse, and van Staveren 1990) and Mosix (Barak, Guday, and Wheeler 1993) offer migration progress facilities. However those are not adequate systems for distributed simulation programs based on the Time Warp because of the simulation programs particularities. The scheduler in this case should consider metrics commonly used for balancing simulation programs and not just issues such as CPU and memory use.

The works in (Reiher and Jefferson 1990, Carothers and Fujimoto 2000, Chanchio and Sun 1996) present algorithms to dynamically balance the simulation migrating only logical processes, i.e., only the work load are transmitted over the network to a pre-existing process on another node avoiding the dynamic

creation overhead of the processes. This approach therefore requires the creation of many processes at the initiation of the simulation in each node. Moreover most of the processes must remain idle to, in due course, receive the logical process as defined by balancing algorithm.

Although this strategy avoids the dynamic creating overhead of the processes, it can occur, for a large amount of idle processes, higher overload due to the idle processes management such as, for example, the CPU usage and memory management by the operating system, particularly in external load conditions, i.e., loads not belonging to the distributed simulation program.

To avoid this overhead, we propose in this paper a migration process mechanism that really implements the migration, with the dynamic balancing capability of the distributed simulation using a balancing algorithm, suitably chosen, not defined by the migration mechanism.

The proposed mechanism is able to dynamically obtain the necessary information for applying the chosen balancing algorithm and is highly transparent as regards the particularities of programming languages, communication libraries and platforms allowing high portability.

3 PROCESSES COLLECTIVE MIGRATION MECHANISM IN DISTRIBUTED SIMULATION

The migration mechanism was projected considering the user mode, which allows for portability on different platforms, thus, favoring the execution in heterogeneous systems. This project was perfomed in order to avoid any dependency of any programming language, any communication library and any platform.

The mechanism is based on the collective migration concept, where, during a migration, all simulation LPs are terminated and then restarted in suitable hosts. Eventually, the suitable host for an LP may be the same host where it has already been found. A master LP controls the entire migration process, being responsible for the following activities:

1. **Apply the static balancing algorithm:** responsible for the simulation LP initial mapping;
2. **Start or restart the LPs:** in response to the balancing algorithm, the master LP restart the LPs in suitable hosts. All the LPs are created simultaneously;
3. **Start a timer:** over the timer interval, the master LP remains idle so the simulation is performed without interference;
4. **Start taking performance metrics:** the master LP requests to all simulation LPs to collect information, in according to the need of the used dynamic balancing algorithm;
5. **Receiving information about the simulation:** after reporting to LPs to obtain data on the simulation, the master must wait until it receives data from all LPs. According to the information, the master can still decide whether any termination condition is satisfied. In this case, all simulation LPs receive a message informing they must close;
6. **Apply the dynamic balancing algorithm:** with the received information, the master applies the algorithm for dynamic balancing. The balancing algorithm should indicate whether there will be migration and, if so, it also inform where each LP must be restarted;
7. **Notify migration:** if the balancing algorithm indicate a need for migration, all LPs must be informed they will sufer a migration;
8. **Wait for the LPs termination:** in case of migration, all simulation LPs should shut down and restarted. However, to maintain variables consistency and messages that may still being exchanged between the LPs, the LP master must wait for the termination of the all simulation LPs. Immediately before they close, the simulation LPs inform the master they are closing;
9. **Return to pass 2.**

The steps followed by Master LP are shown in Figure 1 a).

For a proper synchronization with the master LP, simulation LPs need some additional features. The main one is the use of a synchronization thread, present in all simulation LPs, responsible for exchanging

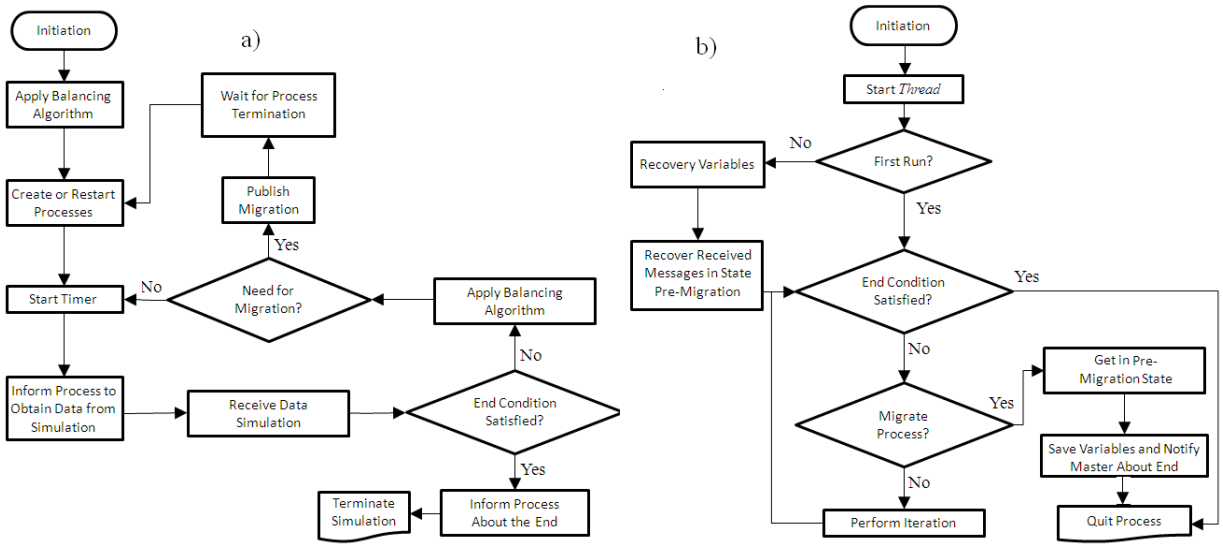

Figure 1: a) Master LP and b) Main Thread Flowcharts

messages with the master LP, for obtaining the necessary metrics to dynamic balancing algorithm and the interaction with the main thread, the one that runs the simulation.

Thus, some additional steps are necessary for the main thread of the simulation LPs. They are:

1. **Initialize the synchronization thread:** before entering the main simulation loop, the synchronism thread should be initiated;
2. **Check if it is the first LP running:** if it is not the first run, the LP has gone through migrations and should have its variables and messages, received during the pre-migration state, recovered. The variables are retrieved from files (in the file server) previously created to store its values. Similarly, messages received during the pre-migration are retrieved from files. These files are saved in a file server, thus independent in which host the LP is restarted, it will have access to the files;
3. **Check the termination condition:** a variable is responsible for informing if the simulation stopping criterion was satisfied. The synchronization thread is responsible for changing this variable when the master LP inform the simulation end;
4. **Check if there is need for migration**: another variable is responsible for informing if the LP will undergo a migration. The synchronization thread is also responsible for changing this variable when the master LP announces a migration. When informed of a migration, the LP enters in the pre-migration state, that will be discussed in Section 3.1. When the LP go out of pre-migration state, it will be able to close and just before its termination, it must shall notify the Master LP it is ready to quit;
5. **Perform iteration:** since the LP verifies that the termination condition is not satisfied and it will not go through a migration, it performs normally the simulation iteration;
6. **Return to pass 3.**

The Figure 1 b) presents the operation flowchart of the main thread of the simulation LPs.

The synchronism thread allows communication between the master LP and simulation LPs. Its function is to receive requests from the master LP and apply them to the simulation LP. It is responsible for the following activities:

1. **Check for messages to the LP:** if there are no messages, the thread should sleep for a chosen period of time which depends, for example, of the dynamic of the simulated system;

2. **Identify received instruction:** if the instruction received from the master is a message stating the need for migration or termination condition, the thread must modify the corresponding variable and then quit;
3. **Get data from simulation:** if the instruction received is not a message for migration or termination condition, then the thread collects simulation data, necessary to the balancing algorithm, and sends them to LP master;
4. **Return to pass 1.**

The figure 2 presents synchronism thread flowchart.

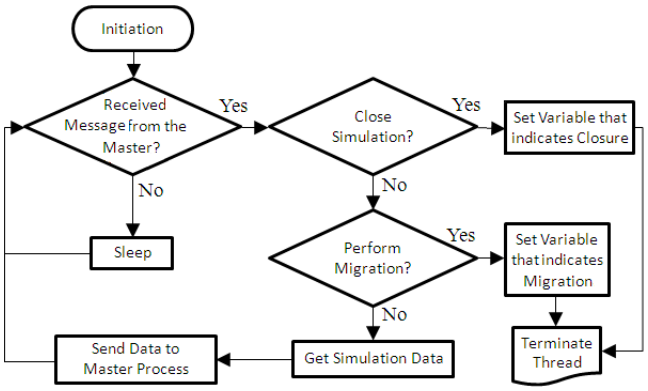

Figure 2: Synchronism Tread Flowchart

3.1 Pre-Migration State

Upon being notified they will undergo migration, the simulation LPs can not quit immediately, as this would lead to loss of messages in transit on the network. To address this situation, the concept of pre-migration state was created. Basically, a LP in this state only receives messages and no longer performs the simulation.

Upon receiving notification it will undergo migration, the LP must inform all other simulation LPs, that it entered the pre-migration state by following these steps:

1. **Notification of the other LPs:** sends a message to all LPs stating that entered the pre-migration state;
2. **Receiving messages:** if the incoming message is a simulation message, as the LP is in the pre-migration state, the message will not be treated, but only stored (in the file server) for later recovery. If the message is a notification of pre-migration from another LP, the LP receiver must update a vector that indicates which other LPs are already in a pre-migration state;
3. **Leave the pre-migration state:** when the LP verify that all other simulation LPs are already in the pre-migration state, he must leave the pre-migration state and quit.

The figure 3 presents the pre-migration state flowchart.

Assuming that the communication channel guarantees the ordering of messages (FIFO), there are no loss of messages when an LP quit because, after his departure from the pre-migration state, all other simulation LPs will know its situation, and thus not send messages to those LP.

If the communication channel can not guarantee the ordering of messages (no FIFO), the procedure does not guarantee that all messages are received by the recipient before it shuts down. The figure 4 a) shows how a message may be losted in the network if the communication channel is not FIFO.

In the figure 4 a) there are two processes in the simulation and the message loss occurs due to the presence of transient messages over the network, which arise due to temporal non ordering of messages

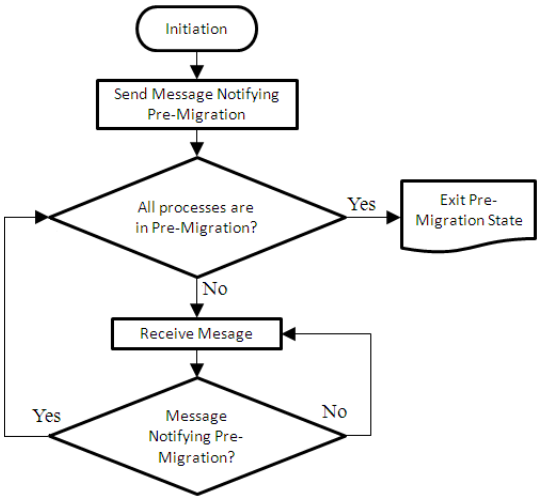

Figure 3: Pre-Migration State Flowchart

through the communication channel. Before entering in the pre-migration state, the process 1 sends a simulation message to the process 2. When the process 2 enters in the pre-migration state (due to notification of the master process), it informs the process 1, which, in turn, enters into the pre-migration state and informs the process 2, the latter, therefore, closes. The simulation message, previously sent by process 1, is lost, because reaches the process 2 when it is already closed.

To solve this problem, it is used a counter of sent and received messages. Whenever a message is sent to an LP, the counter associated with this LP is incremented. Similarly, when an LP receives a message, a counter associated with the sender LP is incremented.

When an LP send a message notifying its entry in the pre-migration state, it appends in this message the number of sent messages to the receptor LP. The receptor LP, before leaving the pre-migration state, checks if the counter value of the received messages of the simulation is equal to the received value as attachment in the entry notification in the pre-migration state for each LP. If equal for all LPs, the LP comes out of pre-migration state. Otherwise, the LP must wait until all messages are received. The figure 4 b) shows this mechanism.

4 EXPERIMENTAL RESULTS

results

There were performed influence analysis of the migration mechanism in the increasing number of messages stragglers and anti-messages, the time for LPs to restart and the time spent to save and recover variables. The objective of these analyzes were to assess the cost of the proposed collective migration mechanism.

A third analysis was performed, aiming to measure the performance gain when using an algorithm for dynamic load balancing along with the proposed collective migration mechanism.

4.1 Physical Infrastructure

For the experimental results, a computational cluster was used with the following technical specifications:

- Five computers with the following features:
  - Intel(R) Processor Core(TM)2 Quad CPU 2.66 GHz;
  - Cache Memory L2 8Mb;
  - Front Side Bus 1066 Mhz;

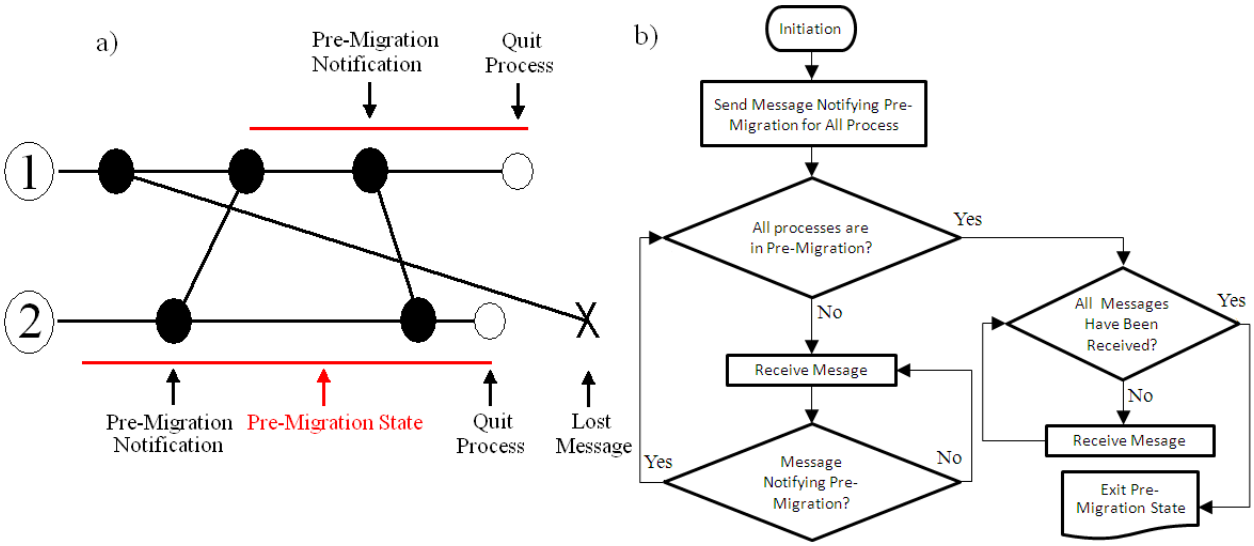

Figure 4: a) Transient Message Problem; b) Flowchart to Threat the Transient Message Problem in the Pre-Migration State

  - RAM Memory: 2 GBytes.
- Communication channel with the following specifications:
  - Ethernet Network 100 Mbits/s.
  - *Switch*: 3Com Baseline Switch 2948-SFP Plus 48-Port Gigabit (3CBL SG48).

The implementations were performed in C language, with the OpenMPI library, version 1.5.4 (Gabriel, Fagg, and Bosilca 2004). The used operating system was the Linux Fedora 13. The file system used was the NFS (Network File System).

With the use of the MPI standard, which guarantees the messages ordering, the solution use of the transient message problem was not necessary.

4.2 Influence of Migration in the Number of Stragglers Messages and Anti-Messages. LPs Restart Time

It is important to point out, for this analysis, the balancing algorithm was not used. The goal is just to evaluate the migration impact in relation to the number of stragglers messages and anti-messages.

For this analysis, we considered simulation models where the LPs had equal probability of event generation and the same virtual logic time to treat the simulation events. The models were simulated with and without the use of the migration mechanism. For simulations with migration, it was used a timer (60 seconds) that initialized the migration process.

Table 1 shows the results in terms of percent increase over the original implementation of TW.

The results show that there is a very small increase in the number of messages stragglerss, which shows that the proposed mechanism have a small impact on the simulation. Moreover, the mechanism is scalable, and as the number of LPs increases, there is a small increase in the number of messages stragglers and anti-messages in this experiments.

4.3 LPs Restart Time

This section presents a factor that directly influences the whole mechanism performance. The time it takes to migrate an LP is a period in which no simulation occurs of fact, and therefore represents a loss in distributed simulation.

Table 1: Influence in Number of Stragglers Mesagens and Anti-Mensagens ($\bar{x} \pm DP$, $n = 10$)

Processs Number	Increase Msg. Strag.(%)	Increase Anti-Msg.(%)	Restart Time (s)
2	0.0301 ± 0.0028	0.210 ± 0.019	0.157 ± 0.032
4	0.0349 ± 0.0030	0.215 ± 0.017	0.903 ± 0.034
6	0.0447 ± 0.0035	0.224 ± 0.023	1.156 ± 0.041
8	0.0546 ± 0.0032	0.239 ± 0.025	1.251 ± 0.025
10	0.0635 ± 0.0042	0.264 ± 0.025	1.302 ± 0.059

From the moment in which an LP enters in the pre-migration state, it stops the simulation and returns to simulate only after the migration and recovery of its data. So, the spent time to restart was considered the period between the entry in the pre-migration state until the simulation restart, without taking into account the time for saving and recovering the variables. The total migration cost can be obtained by summing the times of the LPs restart, recovery and rescue of variables (discussed in Section 4.4).

The number of simulation LPs affects the restart time, because there is an increased amount of incoming and outgoing messages in the pre-migration state. Moreover, another factor that influenced the restart time is the performance of the MPI function used for creating LPs, which has its runtime directly affected by the number of created LPs.

The experiment consisted in migrating LPs and measure the time to restart them. Ten experiments were performed in order to obtain the mean ($\bar{x}$) and standard deviation (*SD*) for each situation. The table 1 summarizes the experimental results.

According to the results, the restart time increases as the number of LPs increases. It was possible to find that the spent time by the function that dynamically creates LPs was mostly responsible for the restart time. This times are depending of the openMPI performance.

4.4 Variables Rescue and Recovery Time

The time to recovery the variables (LVT, future events list, sent messages list and stored local states list) was the time required to obtain the data from the file server, allocate memory and assign their values. On the other hand, the spent time to rescue variables was the time to send data to file server. Figure 5 shows the results, where the x-axis is the amount of recovered data and y-axis, the spent time.

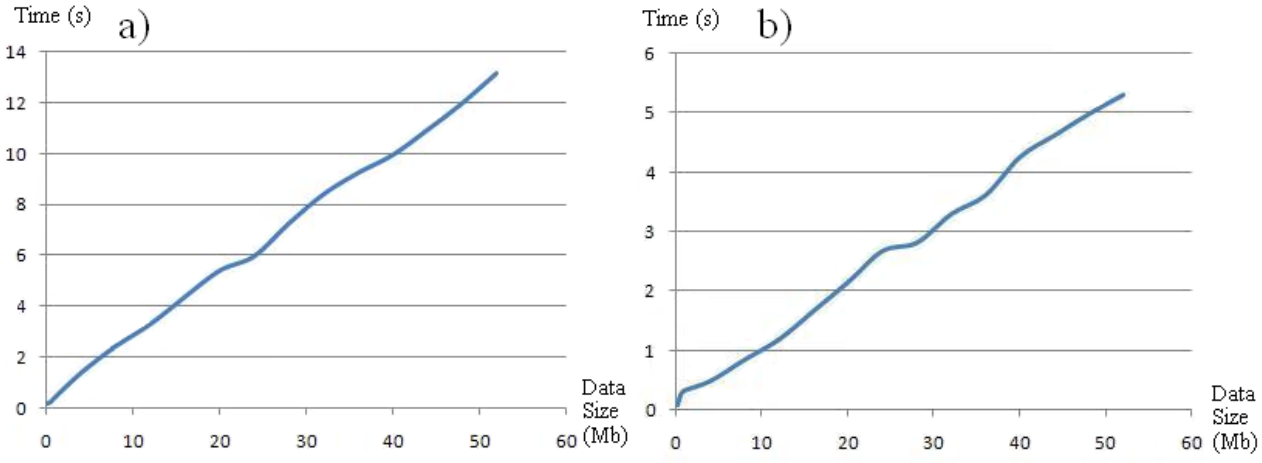

Figure 5: Spent Time to: a) Recover and b) Rescue Variables

Different sizes of data were considered. For each size, it was measured the time to rescue and recovery of variables (10 replications of the experiment were performed for each size ($n = 10$). The variation coefficient ($SD/\bar{x}$) for data recovery was less than 6% and less than 5% for the data rescue.

The time for variables recovery was greater than the time to rescue them. In both cases the cost is associated with the message exchange via communication network, around 4s for 50Mb (50/12.5). However, besides the transmition time of data across the network, the variables recovery time is affected by the time of dynamic memory allocation, which involved data structures such as lists. Experimentally, this spent time was sometimes greater than the spent time for the data transfer and it is the main reason for the difference observed between the periods of recovery and rescue.

Both the spent times contribute to the cost of migration and, therefore, should be considered. As can be seen, the spent time to rescue and recover variables, together, can reach values greater than 15 seconds, for data greater than 40 Mb. This spent time is much longer than the spent time to restart the LPs, as shown in Section 4.3. Thus, the importance of implementing an effective Garbage Collection mechanism, to keep small amounts of data, is evident, because the data recovery and rescue time is directly proportional to the data size.

4.5 Evaluation with Dynamic Balancing Algorithm

To perform the experiments, it was proposed an algorithm for dynamic load balancing, capable of working with different metrics for levels of load, i.e., the algorithm do not depend on how the metric is calculated. This algorithm used the proposed mechanism of collective migration.

First, the algorithm calculates the load of each processor, which is the sum of the allocated LPs charges in the processor. Then the algorithm computes some values that will be needed to determine whether there will be or not migration and which LPs will be migrated. The values are calculated as it follows:

1. **Determine the most and least loaded processors**;
2. **Determine what LP will be migrated**: The LP to be migrated is the slowest LP in the most loaded processor;
3. **Determine the current charge difference between the most loaded processor and the least loaded processor**: to also determine the future difference between these two processors if there is migration from the slowest LP from the over loaded processor to the least loaded processor. The future difference is an estimate and it is calculated by subtracting the load of the over loaded processor by load of the LP to be migrated, adding the charge of the LP to be migrated to the least loaded processor and subtracting the estimated load on the processor originally most loaded by the load of the originally least loaded processor;
4. **Calculate the mean, standard deviation,** $delta$ **and** *gamma* **of the processors load**: $delta$ is the subtraction of the mean by the standard deviation, while *gamma* is the sum of the mean by the standard deviation.

Calculated these values, it remains to determine whether or not there is a migration. A migration will be performed if the following two conditions are true:

1. **If there is a processor whose load is less than** $delta$ **or there is a processor whose load is greater than** *gamma*;
2. **If the differences, current and estimated future, between the processors loads, most and least loaded, obey any of the following rules**:
   (a) The estimated future difference is still greater than zero, i. e., even after removing the slowest LP of the originally most loaded processor and migrating to the originally least loaded processor, the most loaded processor will still have a greater load than the least loaded processor;

(b) If the load on the originally least loaded processor, after the migration estimate of the slowest LP, becomes greater than the load of the originally most loaded LP, the diference between the load of the originally least loaded processor and the originally most loaded processor can not be larger than half of the current difference. Thus, by the estimate, the originally most loaded processor will have a lower load than the originally least loaded processor. In this case, the migration is performed only in cases where the processors inversion most and least charged leads to the difference, according to estimated, is less than or equal to half of this current difference.

Determined that there will be migration, the slowest LP of the most loaded processor is migrated to the least loaded processor.

For the experiments, the metric used by the algorithm to determine the system load, was the feed rate metric of the of the LPs local clock (Burdorf and Marti 1993). The experiments were performed with different simulation models and with different load variations. As variations of load, the LPs had different probabilities of generating events and different times for processing the simulaions events. Some processors also suffered from outside loads to the simulation, caused by external simulation programs created for the purpose of consuming of processing resource. A timer (15 seconds), present in the master LP, called the balancing algorithm to analyze the system load.

The results of the dynamic algorithm were compared to the static balancing algorithm Round Robin (RR). For each model, were performed 10 simulations with the RR and 10 simulations with the dynamic algorithm, the results are based on the average of these runs.

The analyzes were based on two factors, the efficiency and the total execution time of the simulation. The efficiency is the ratio between the actual number of processed events, those which did not have rollbacks, by the total number of events processed.

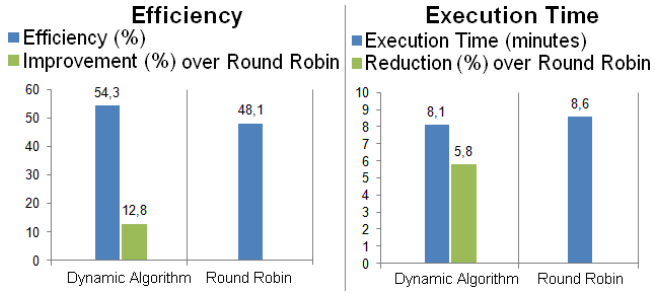

Figure 6: Dynamic algorithm versus RR

As can be seen in Figure 6, there were an improvement of nearly 13% in the simulation efficiency and a reduction of almost 6% compared to the total execution time.

5 CONCLUSION

The proposed migration mechanism is based on the use of a master LP who manages the simulation LPs. These, in turn, have two threads: the main that performs the simulation, and the synchronism, which communicates with the master LP to check the procedure that the LP must perform in the simulation at each time. This migration mechanism consist in the migration of all the LPs simultaneously. The mechanism is simple to be implemented, has a reasonable migration time and has little influence on the number of messages stragglers and anti-messages.

According to the results obtained, the proposed mechanism of migration enables a performance improvement in distributed simulations, with an attractive alternative when it comes to LPs migration for these environments.

It is important to emphasize the need for good memory management mechanism, due to the high time associated with saving and restoring the states of LPs. The tests were based on an implementation that used the communication library OpenMPI and the results reflect the resources offered by this library.

The major contribution of this work was the proposal of a mechanism for LPs migration for distributed simulation based on the use of the TW protocol with the ability to obtain, at runtime, information about the state of the simulation that can be used by a scheduling algorithm with highly portability. However some questions may arise about the mechanism such as: does the master process, the shared file server and/or the all-to-all communication required by the mechanism present a bottleneck in the simulation? Is the mechanism scalable with increasing number of processes?

The migration processes is a very extensive subject, and, in this paper, we give some direction in how to explore it to PDES systems. The results also provide some evidences that it may improve the distributed simulation performance.

ACKNOWLEDGMENTS

The authors would like to acknowledge the financial support provided by CAPES - Improvement Coordination of Higher Level Staff.

REFERENCES

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2008. *Discrete-Event System Simulation*. 4 ed. ndia: Prentice Hall.

Barak, A., S. Guday, and R. G. Wheeler. 1993. *The MOSIX Distributed Operating System: Load Balancing for UNIX*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Burdorf, C., and J. Marti. 1993. "Load balancing strategies for Time Warp on multi-user workstations". *The Computer Journal*.

Carothers, C. D., and R. Fujimoto. 2000. "Efficient Execution of Time Warp Programs on Heterogeneous, NOW Platforms". *IEEE Transactions on Parallel and Distributed Systems* PDS-11 (3): 299–317.

Chanchio, K., and X.-H. Sun. 1996. "Efficient process migration for parallel processing on non-dedicated networks of workstations". Technical Report ICASE 96-74, NASA (Hampton, VA US), Hampton.

Chen, L.-l., Y.-s. Lu, Y.-p. Yao, S.-l. Peng, and L.-d. Wu. 2011. "A Well-Balanced Time Warp System on Multi-Core Environments". In *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*, PADS '11, 1–9: IEEE Computer Society.

De Grande, R. E., and A. Boukerche. 2011. "Dynamic balancing of communication and computation load for HLA-based simulations on large-scale distributed systems". *Journal of Parallel and Distributed Computing* 71 (1): 40–52.

El Ajaltouni, E., A. Boukerche, and M. Zhang. 2008. "An Efficient Dynamic Load Balancing Scheme for Distributed Simulations on a Grid Infrastructure". In *Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium on*, 61–68.

Fujimoto, R. M. 2000. *Parallel and distributed simulation systems*. 1 ed. USA: John Wiley & Sons, Inc.

Gabriel, E., G. E. Fagg, and G. Bosilca. 2004, September. "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation". In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 97–104. Budapest, Hungary.

Jefferson, D. R. 1985. "Virtual time". *ACM Transactions on Programming Languages and Systems* 7 (3): 404–425.

Jiang, M., R. Anane, and G. Theodoropoulos. 2004. "Load balancing in distributed simulations on the grid". In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, Volume 4.

Kassab, E., J. K. Tun, S. Arora, D. King, K. Ahmed, D. Miskovic, A. Cope, B. Vadhwana, F. Bello, N. Sevdalis et al. 2011. "Blowing up the Barriers in Surgical Training: Exploring and Validating the Concept of Distributed Simulation". *Annals of surgery* 254 (6): 1059–1065.

Lamport, L. 1978. "Time, clocks, and the ordering of events in a distributed system". *Communications of the ACM* 21 (7): 558–565.

Law, A. M. 2007. *Simulation modeling and analysis*. 4 ed. USA: McGraw-Hill.

Low, M. Y. H. 2002. "Dynamic load-balancing for BSP time warp". In *In Proceedings of the 35th Annual Simulation Symposium*, SS'02, 267–274: IEEE Computer Society.

Meraji, S., W. Zhang, and C. Tropper. 2010. "A Multi-State Q-Learning Approach for the Dynamic Load Balancing of Time Warp". In *Proceedings of the 2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*, PADS '10, 142–149: IEEE Computer Society.

Mullender, S. J., G. van Rossum, A. S. Tanenbaum, R. van Renesse, and H. van Staveren. 1990. "Amoeba: A Distributed Operating System for the 1990s". *Computer* 23 (5): 44–53.

Peschlow, P., T. Honecker, and P. Martini. 2007. "A Flexible Dynamic Partitioning Algorithm for Optimistic Distributed Simulation". In *Principles of Advanced and Distributed Simulation, 2007. PADS '07. 21st International Workshop on*, 219–228.

Reiher, P. L., and D. Jefferson. 1990. "Virtual Time Based Dynamic Load Management In The Time Warp Operating System". *Transactions of the Society for Computer Simulation* 7 (2): 103–111.

Tanenbaum, A. S., and M. V. Steen. 2006. *Distributed systems: principles and paradigms*. 2 ed. Upper Saddle River: Prentice Hall.

Wainer G A, M. P. J. 2011. *Discret Event Simulation: Theory and Applications*. 2 ed. Tayleor and Francis Group.

AUTHOR BIOGRAPHIES

MATEUS AUGUSTO FAUSTINO CHAIB JUNQUEIRA is a doctorate student in eletrical engeneering in the Federal University of Itajuba in Brazil. He obtained his undergraduate in Computer Engeneering from the Federal University of Itajuba and master degree in Computer Science from the Federal University of Itajuba. His research interests include Distributed Systems and Eletromagnetism. His e-mail address is mateusafcj@gmail.com.

EMERSON ASSIS DE CARVALHO is a software engeneering of IBM and teacher of the University of Alfenas/MG. He obtained his undergraduate in Computer Science from the University of Alfenas and the master degree in Computer Science from Federal University of Itajuba. His research interests include Agile, Software Development Processes, Computer Networks, Operating Systems and Database. His e-mail address is assis.emerson@gmail.com.

EDMILSON MARMO MOREIRA is a Associate Professor of the Systems Engineering and Information Technology Institute at the Federal University of Itajuba in Brazil. He obtained his undergraduate in Computer Science from the University of Alfenas, masters and doctorate degrees, respectively, in Computer Science from the University of So Paulo. His research interests include Distributed Systems, Discrete Event Simulation and Optimistic protocols for distributed simulation. His e-mail address is edmarmo@unifei.edu.br.

OTAVIO AUGUSTO SALGADO CARPINTEIRO was born in Rio de Janeiro, RJ, Brazil. He has a B.Sc. degree in Mathematics, a B.Sc. in Music, and a M.Sc. in System and Computing Engineering, all of them from the Federal University of Rio de Janeiro, Brazil. He has a D.Phil. degree in Cognitive and Computer Science from the University of Sussex, UK. He has worked as a system analyst for many years, and presently, is an associate professor at the Federal University of Itajuba, MG, Brazil, in which he has done research, supervised graduate students, and taught courses in Computing Engineering. His email address is otavio@unifei.edu.br.