Dynamic Load Balancing Schemes for Large-Scale HLA-based Simulations

by

Robson Eduardo De Grande

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Ph.D. degree in
Computer Science

School of Electrical Engineering and Computer Science
Faculty of Computer Science
University of Ottawa

Abstract

Dynamic balancing of computation and communication load is vital for the execution stability and performance of distributed, parallel simulations deployed on shared, unreliable resources of large-scale environments. High Level Architecture (HLA) based simulations can experience a decrease in performance due to imbalances that are produced initially and/or during run-time. These imbalances are generated by the dynamic load changes of distributed simulations or by unknown, non-managed background processes resulting from the non-dedication of shared resources. Due to the dynamic execution characteristics of elements that compose distributed simulation applications, the computational load and interaction dependencies of each simulation entity change during run-time. These dynamic changes lead to an irregular load and communication distribution, which increases overhead of resources and execution delays. A static partitioning of load is limited to deterministic applications and is incapable of predicting the dynamic changes caused by distributed applications or by external background processes. Due to the relevance in dynamically balancing load for distributed simulations, many balancing approaches have been proposed in order to offer a sub-optimal balancing solution, but they are limited to certain simulation aspects, specific to determined applications, or unaware of HLA-based simulation characteristics. Therefore, schemes for balancing the communication and computational load during the execution of distributed simulations are devised, adopting a hierarchical architecture. First, in order to enable the development of such balancing schemes, a migration technique is also employed to perform reliable and low-latency simulation load transfers. Then, a centralized balancing scheme is designed; this scheme employs local and cluster monitoring mechanisms in order to observe the distributed load changes and identify imbalances, and it uses load reallocation policies to determine a distribution of load and minimize imbalances. As a measure to overcome the drawbacks of this scheme, such as bottlenecks, overheads, global synchronization, and single point of failure, a distributed redistribution algorithm is designed. Extensions of the distributed balancing scheme are also developed to improve the detection of and the reaction to load imbalances. These extensions introduce communication delay detection, migration latency awareness, self-adaptation, and load oscillation prediction in the load redistribution algorithm. Such developed balancing systems successfully improved the use of shared resources and increased distributed simulations' performance.

List of Publications

Referred Journal Papers

- R. E. De Grande, A. Boukerche. Measuring and Analyzing Migration Delay for the Computational Load Balancing of Distributed Virtual Simulations. *IEEE Transactions on Instrumentation and Measurement*. To Appear.

- R. E. De Grande, A. Boukerche, and H. M. S. Ramadan. Measuring Communication Delay for Dynamic Balancing Strategies of Distributed Virtual Simulations. *IEEE Transactions on Instrumentation and Measurement*. 60(11):3559-3569, November 2011. doi: 10.1109/TIM.2011.2161143.

- R. E. De Grande; A. Boukerche, and H. M. S. Ramadan. Decreasing Communication Latency through Dynamic Measurement, Analysis, and Partitioning for Distributed Virtual Simulations. *IEEE Transactions on Instrumentation and Measurement*. 60(1):81-92, January 2011. doi: 10.1109/TIM.2010.2065730.

- R. E. De Grande and A. Boukerche. 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, January 2011. doi=10.1016/j.jpdc.2010.04.001.

- R. E. De Grande, A. Boukerche, and H. Ramadan. Distributed re-arrangement scheme for balancing computational load and minimizing communication delays in HLA-based simulations. *Journal of Concurrency and Computation: Practice and Experience*. 2011. doi: 10.1002/cpe.1807. John Wiley & Sons, Ltd. ISSN:1532-0634.

- E. E. Ajaltouni, M. Zhang, A. Boukerche, and R. E. De Grande. An Adaptive Dynamic Load Balancing Technique for Grid-Based Large Scale Distributed Simulations. *Journal of Interconnection Networks*. Pages 391-419, 2009.

Referred Conference Papers

- R. E. De Grande and A. Boukerche. Predictive Dynamic Load Balancing for Large-Scale HLA-based Simulations. In *Proceedings of the IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications*. Pages 4-11, September 2011. doi: 10.1109/DS-RT.2011.17

- R. E. De Grande and A. Boukerche. Dynamic load redistribution based on migration latency analysis for distributed virtual simulations. In *Proceedings of the IEEE International Workshop on Haptic Audio Visual Environments and Games*. Pages 88-93, October 2011. doi: 10.1109/HAVE.2011.6088397

- R. E. De Grande and A. Boukerche. Distributed dynamic balancing of communication load for large-scale HLA-based simulations. In *Proceedings of the 2010 IEEE Symposium on Computers and Communications*. Pages 1109-1114, June 2010. doi: 10.1109/ISCC.2010.5546621

- R. E. De Grande and A. Boukerche. Self-Adaptive Dynamic Load Balancing for Large-Scale HLA-Based Simulations. In *Proceedings of the IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications*. Pages 14-21, October 2010. doi: 10.1109/DS-RT.2010.11

- R. E. De Grande and A. Boukerche. A redistribution scheme centred on communication delay for distributed virtual simulations. In *Proceedings of the IEEE International Symposium on Haptic Audio-Visual Environments and Games*. Pages 1-6, October 2010. doi: 10.1109/HAVE.2010.5623959

- R. E. De Grande and A. Boukerche. A dynamic, distributed, hierarchical load balancing for HLA-based simulations on large-scale environments. In *Proceedings of the 16th international Euro-Par conference on Parallel processing*. Pages 242-253, 2010. ISBN:3-642-15276-7 978-3-642-15276-4

- R. E. De Grande and A. Boukerche. Dynamic Load Balancing Using Grid Services for HLA-Based Simulations on Large-Scale Distributed Systems. In *Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*. Pages 175-183, October 2009. doi: 10.1109/DS-RT.2009.33

- R. E. De Grande and A. Boukerche. Dynamic partitioning of distributed virtual simulations for reducing communication load. In *Proceedings of theIEEE International Workshop on Haptic Audio visual Environments and Games*. Pages 176-181, November 2009. doi: 10.1109/HAVE.2009.5356113

- A. Boukerche and R. Grande. Optimized Federate Migration for Large-Scale HLA-Based Simulations. In *Proceedings of the 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*. Pages 227-235, October 2008. doi: 10.1109/DS-RT.2008.29

Acknowledgements

I am greatly thankful to my supervisor Dr. Azzedine Boukerche, who always provided guidance, encouragement, and financial support with professionalism and moral throughout the entire period of my Ph.D. thesis work. This thesis work would not have been possible without his advice and support.

I would like to thank my friends and colleagues that I have met during my Ph.D. at the PARADISE laboratory. They have always offered me help whenever I needed, turning my Ph.D. work more pleasant and productive. Special thanks to Dr. Richard Pazzi, Kent (Yun-feng Gu), Cristiano G. de Rezende, Zhenxia Zhang, Mario Fanelli, and many others for the moments of fun and discussion, advice, and sometimes encouragement.

I owe my deepest gratitude to my wife Priscila M. L. De Grande, who has been supportive during all my Ph.D. She has been always by me side in good and bad moments of the Ph.D. work, comforting and providing me any help she could offer at any time.

I am also truly grateful to my family, my parents, and brother. They have been always helpful with whatever they could assist during the journey through my Ph.D., even with the long distance as great obstacle that limited our contact.

Contents

List of Tables

List of Figures

Glossary of Terms

AGIM Aggregation Interest Management

AOIM Area of Interest Management

API Application Programming Interface

BEEP Blocks Extensible Exchange Protocol

CAT Cluster Advance Time

CAC Cluster Adaptation Component

CCM Contractual Computing Mechanism

CLB Cluster Load Balancer

CMI Communication Monitoring Interface

CPC Cluster-to-Processor Communication

CPU Central Processing Unity

DDM Data Distribution Management

DDMS Data Distribution Management Service

DMS Declaration Management Service

DMSO Defense Modeling and Simulation Office

EWMA Exponential Weighed Moving Average.

FedExec Federation Executive process

FMI Federate Management Interface

FMS Federate Management Service

FOM Federation Object Model

FT Fault-Tolerant

FTP File Transfer Protocol

FT-RSS Fault-Tolerant Resource Sharing System

GA Grid Agent

GCE Grid Computing Environment

GDSA Grid-based Distributed Simulation Architecture

GGF Global Grid Forum

G-HLAM Grid HLA Management System

GIS Grid Information Service

GPDS Grid-Based Parallel and Distributed Simulation environment

GRAM Grid Resource Allocation Manager

GridFTP Reliable Data Transfer

GSI Grid Security Infrastructure

GVT Global Virtual Time

HLA High Level Architecture

IEEE Institute of Electrical and Electronics Engineers

IPC Inter-Processor Communication

inQ In Queue

LFS Local Factory Service

libRTI Run-Time Infrastructure library

LLB Local Load Balancer

LM Load Manager

LMoI Local Monitoring Interface

LMiI Local Migration Interface

LMS Load Management System

LP Logical Process

LRC Local Run-Time Infrastructure Component

LS Local Service

LVT Least Virtual Time

MDS Monitoring and Discovery Service

MIS Monitoring Information Service

MM Migration Manager

NVE Network Virtual Environment

OCM-G OMIS-Compliant Monitoring system for the Grid

OGSA Open Grid Services Architecture

OMT Object Model Template

OOMS Object and Ownership Management Service

outQ Out Queue

PAT Processor Advance Time

PEDS Parallel Discrete Event Simulation

PJPS Resigning, Joining, Publishing and Subscribing HLA calls

PPC Processor-to-Processor Communication

PSDE Parallel and Distributed Simulation Environment

QoS Quality of Service

RAC Resource Adaptation Component

RBAC Role-based Access Control

RSL Resource Specification Language

RSS Resource Sharing System

RTI Run-Time Infrastructure

RTIExec Run-Time Infrastructure Executive process

SimEvent Simulation Event

SimKernel Simulation Kernel

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

SOHR Service Oriented HLA RTI

TMS Time Management Service

VTP Virtual Time Progress

WebMDS Web Service Monitoring and Discovery Service

WSIM Web Services Internet Management

XMSF Extensible Modeling and Simulation Framework

WS Web Service

WSDL Web Service Definition Language

WS-GRAM Web Service Grid Resource Allocation Manager

WSRF Web Service Resource Framework

Chapter 1

Introduction

Distributed simulations have been receiving increasing interest from industry and academia. In large-scale scenarios, High Level Architecture (HLA) based distributed simulations have been extensively used to determine situations that might influence a behaviour or produce situations, in which would be costly or harmful in real life, or to measure reactions or train professionals. These simulations are employed in several areas, such as simulations of wireless and mobile networks to identify issues of and test new communication protocols [18, 17]; flight simulation to test the effect of atmosphere on flights [101], to test the flight of specific aircraft [89], to observe the combat between aircrafts [92], or to evaluate novel air traffic management techniques [6]; fire fight simulation to prevent of combat forest fire [29]; supply chain simulation to model and analyze complex scenarios [21] or to observe symbiotic simulation control for supply chain [34]; simulation of complex manufacturing systems with emphasis on logistics [70]; container terminal training simulation to improve the learning of operators using 3D simulations [32]; launch vehicle testing simulations [103]; and hardware-in-the-loop simulation for modern weapons development [84].

Since such distributed simulations might grow to large-scale dimensions, issues can affect the performance of their execution or preclude the execution of these simulations. Generally, these issues are load imbalances caused by the uneven arrangement of simuations' load and by dynamic load changes that can occur during run time. Uneven allocation of load stems from a lack of knowledge about distributed applications or shared available resources. This irregular partitioning leads to some resources to be overloaded

while others are underloaded. Moreover, distributed applications might present unpredictable run-time load changes. In such cases, a static initial load partitioning barely reaches efficient resource usage because it cannot optimally distribute load based on a deterministic assessment of the simulations' characteristics. Other than the existence of uneven load distribution, factors inherent to large-scale distributed systems must be considered, such as the heterogeneity of resources and the presence of external background load. Therefore, in order to provide a more efficient use of shared resources and to increase the performance of large-scale distributed HLA-based simulations, a dynamic balancing of computation and communication load is devised.

A distributed simulation consists of a set of interactive, independent simulation entities that perform simulation processing over a group of shared resources. The distribution of such entities on resources may lead to causality simulation inconsistencies, thus corrupting the simulation's result. To organize the interactions between simulation entities, the HLA framework is employed. The HLA standard includes rules and services, which maintain the communication order of simulations. However, the HLA standard is limited to the management of only distributed simulations, and it is unaware of the issues caused by distributing simulations on unreliable, non-dedicated, shared resources; one example being load imbalances and failures.

1.1 Thesis Statement

Since HLA-based simulations can undergo performance loss, load balancing becomes a crucial mechanism that reduces execution simulation time through the increase of even consumption of resources. In order to react properly to run-time load changes, a dynamic load redistribution scheme needs to be devised. Moreover, because simulations are deployed on distributed environments that might be large in scale, the devised balancing scheme has to be aware of heterogeneity and non-dedication of resources, requiring third-party monitoring tools and benchmarks. Therefore, a general purpose, distributed, dynamic balancing scheme for HLA-based simulations, as well as its extensions, is proposed and developed.

1.2 Motivation

HLA framework does not provide any management of resources for distributed discrete event simulations, and load imbalances might really impose substantial performance loss to distributed simulations, preventing or making unfeasible the execution of such simulations based on their deployment on certain resources. Consequently, a resource management system that controls and organizes the load of distributed optimistic and conservative simulations is needed. However, resource management systems, such as Grid Services [39], only provide mechanisms for monitoring distributed resources and applications, so a general purpose balancing scheme is required.

Due to the importance and the complexity of solving the problem of balancing distributed load, load balancing is a recurrent problem in many areas. Since balancing load belongs to a np-hard complexity class of problems, there is not a solution that can optimally solve the redistribution problem in polynomial time. In the case of balancing computational load in a set of non-dedicated resources, a distributed multiple knapsack problem, which is a variant of and can be reduced to a bin packing or knapsack [43] problem, raises to be solved in order to achieve a optimal solution the load distribution. In the case of balancing communication load, the analysis and placement of distributed parts according to their interaction load is comparable or reduced to the quadratic assignment problem (QAP) [43], which is another np-hard problem. Therefore, heuristics are constantly and extensively used to provide feasible sub-optimal solutions that achieve performance gain above an acceptable threshold.

The balancing schemes that detect and react to load imbalances during run-time are based on heuristics in order to offer a load redistribution response in feasible time. The balancing schemes developed previously for distributed simulations are limited and do not deal with all the distributed environment characteristics or all simulations. These schemes observe application-specific or environment-specific aspects in their design and cannot be applied to balance load of any simulation in any set of resources. As a result, in order to provide a balancing scheme that effectively works for any HLA-based simulation and in configuration of resources, these limitations need to be avoided by integrating all the characteristics of distributed environments and simulations in a load redistribution algorithm.

1.3 Objectives

Balancing systems are introduced to overcome the limitations of the previous balancing designs and to present timely responses to communication and computational load imbalances. Such balancing systems are developed through observation and analysis of characteristics of distributed environments that might influence their effectiveness in reorganizing load. Third-party tools, such as Grid Services, are used to enabled the detection of global load of a resource and consequently allowing the identification of external background that might cause overload in resources. Resource benchmarks are used in the balancing systems to determine and normalize the differences in computational capacities among resources. In both cases, heterogeneity of resources and external load can mislead the reallocation of resources and generate imbalances instead of improving the utilization of resources for performance gain. The characteristics of simulations are also considered in the design of these proposed systems to keep simulations consistent while their load are transferred between resources, which generally consist in providing efficient federate migration. Thus, with such measures, better awareness of environment load status and HLA-based simulations is introduced when balancing load and higher simulation performance gain is provided.

1.4 Contribution

In order to enable the design of the devised balancing systems, a two-phase federate migration is first developed. With such migration technique, modifications on simulation load distributed are enabled, and migration latency is minimized, providing means to the balancing systems to present high responsiveness to imbalances. Based on such a federate migration procedure, a general-purpose centralized dynamic balancing scheme is initially developed. In this initial system, the simulation load is redistributed based on the detected imbalances; the detection and redistribution mechanisms are aware of heterogeneity and non-dedication of resources. This awareness is introduced through the access to monitoring information from Grid Services and the use of benchmarks to normalize load. This centralized design effectively and efficiently reacts to simulation load imbalances, but it is based on global view of the system's load status in a central balancing element. In order to avoid the drawbacks of a centralized balancing system, the same

system is redesigned in a distributed architecture. In the distributed balancing scheme, the redistribution algorithm is conducted by a set of balancing elements to balance the load of certain environment region. In this scheme, the load is properly reorganized according to the load imbalances, but some balancing efficiency is lost due to the lack of global view of entire system's load status. Using the distributed balancing scheme as core, extensions are also developed to improve the balancing efficiency according to some aspects arising form redistributing load or certain specific simulation load characteristics, such as migration delay, communication delay, self-adaptation of load balancing, and prediction of load imbalances.

1.5 Outline

The remainder of this thesis is organized as follows. Chapter 2 presents the topic's background information and the related work. The topic's background information includes the description of the HLA standard and its limitations, the main characteristics of Grid Services, which are extensively employed in the proposed balancing system, and the architectures for enabling HLA simulations in large-scale environments through Grid Services. The related work describes the existing solutions for redistributing load dynamically, as well as their drawbacks. In Chapter 3, the federate migration protocol, essential for the development of balancing schemes, is delineated. In Chapter 4, the centralized hierarchical three-phase scheme is described, and experiments are outlined and discussed. In Chapter 5, the distributed balancing scheme is delineated, and experiments for evaluating the scheme are demonstrated. In Chapter 6, extensions for the distributed balancing scheme are described, and experimental results are detailed and discussed. Finally, Chapter 7 presents the thesis conclusion and the directions for future research work.

Chapter 2

Background and Related Work

Background information is presented in order to contextualize the thesis topic, providing means to understand the motivations and challenging issues in designing dynamic load balancing solutions for large-scale HLA-based simulations. An overall description of High Level Architecture is used to explain the limitations of the simulation design framework related to the load imbalance aspects. Due to the importance of the tools for managing distributed resources and applications, Grid Computing Services are described together with HLA framework; such tools are essential for monitoring, accessing remote resources in a distributed, shared environment. Architectures supporting large-scale HLA-based simulations are also described to list the existent solutions for enabling HLA simulation and coordinating their execution in large-scale environments; basically, these architectures attempt to overcome the main obstacles in executing large-scale by allowing the discovery and the interaction of simulation federates in these environments. The related work presents previous approaches for coordinating federate migration and load balancing distributed simulations to described these solutions' drawback and identify the challenging issues.

2.1 High Level Architecture

High Level Architecture was developed to provide a general purpose standard method to design and coordinate distributed simulations. The standard was initially developed by the United States Department of Defence to allow the execution of distributed, par-

allel simulations for military purposes. In 2000, the HLA specification became an open IEEE standard [3] and became a recommended process for developing interoperable parallel simulations. Prevent causality inconsistencies and provide interoperability and reusability in simulations' design, management mechanisms and design principles were introduced. Interoperability corresponds to the capacity of many elements to operate together, which is produced by imposing a singular interaction standard. Re-usability stems from the separation of simulations into independent, stand-alone components, which can be employed in other simulation contexts without any modification. An HLA-based simulation is called federation and is composed of interactive, independent components called federates, which interact according to specification-defined communication methods to avoid causality inconsistencies. Essentially, the HLA standard consists of rules, an object model template, and an interface specification.

The HLA rules restrict the actions of federates by defining responsibilities regarding federations and federates in simulations. These rules enforce compliance with the specification in order to avoid inconsistencies. They determine the design of simulations by guiding the development into the HLA specification aspects. Such compliance aspects require the existence of a Federation Object Model documented according to a Object Model Template to define the simulation objects, simulation object exchange among federates restricted by the Run-Time Infrastructure and in accordance with the HLA interface specification, the access and modifications of simulation objects delimited by pre-established ownership criteria, and federate's coordination of its local time.

The Object Model Template (OMT) provides a common framework for the communication inside HLA simulations. The template defines all the information exchanged during simulation execution. The OMT describes simulation shared objects, attributes, and interactions. The description determines the format and syntax of information that needs to be recorded or exchanged, enabling and facilitating interoperability among simulations and reusability of simulation components. A OMT consists of a Federation Object Model, which relates the data for the whole federation, and a Simulation Object Model, which involves the data of a single federate.

The Interface Specification defines the HLA management services and describes the APIs to access them. Since simulations are coordinated according to these services, every federate needs to access such APIs in order to interact with or exchange information

to other federates. The management services allow the organization and coordination of simulation elements properly and are classified in 6 management areas. The areas correspond to Federation Management, Declaration Management, Object Management, Ownership Management, Data Distribution Management, and Time Management. These services are responsible for filtering and restricting the the simulation interactions, limiting the design concerns of application developers just to issues regarding simulation modelling.

As shown in Figure 2.1, the RTI middleware is composed of an RTI Executive process (RTIExec), a Federation Executive process (FedExec), and the RTI library (libRTI) [3]. The RTIExec coordinates Federations and FedExecs. A FedExec is responsible for managing the simulation of one federation, which comprises the organization of the life cycle of all federates within a simulation. The libRTI embraces the mechanisms employed to manage distributed simulations and provides them according to the Interface Specification as the Management Services; the library is made available to a federate according to the HLA interface specification. The management services are fundamental in HLA simulations and are accessed by federates to coordinate every operation and data exchange. Thus, all communication between federates is observed and coordinated by the RTI, and federates are enabled to access the services through the Local RTI (LRC), which contains all libRTI interfaces.

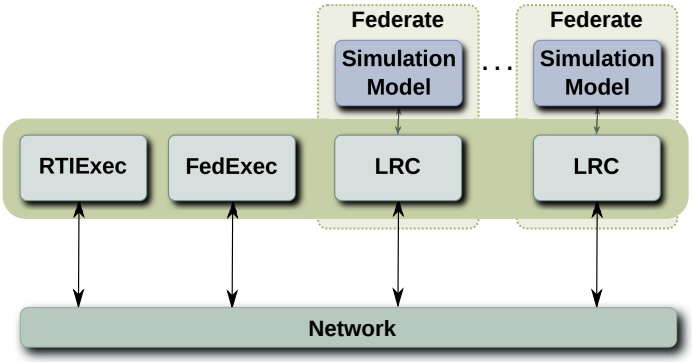


Figure 2.1: HLA General Architecture.

Even though HLA provides an RTI service called Data Distribution Management (DDM), which presents mechanisms for minimizing the communication overhead, the HLA standard is unable to offer any solution to balance the computational load and communication delays. DDM introduces data exchange methods, such as publish and

subscribe, and interest management techniques, which reduce the amount of information that is transmitted between federates by restricting communication to only relevant data. DDM decreases the network overhead caused by the simulation, allowing higher available bandwidth (throughput), but it is unaware of external processes that might also be consuming network resources or network distances that can generate high communication latency for a simulation. The HLA standard does not present any mechanism or method for controlling the distributed simulation load placed on shared resources or handling the shared resources on which simulations run. HLA relies on the fact that all available resources are dedicated, show reliability, and present enough computing power and communication capacity for simulation applications. Because the performance of HLA simulations can be jeopardized by only one overloaded resource, load balancing systems are required to improve performance through the maximization of resource utilization and minimization of communication delays.

2.2 Grid Computing

Resource management systems are essential for administering distributed resources; the Grid system is extensively employed to organize distributed applications that run on shared resources through the provisioning of various services. According to Foster [40], Grid computing involves the organization of resources, individuals, and institutions, establishing a coordinated resource-sharing system aimed at flexibility and security. Grid computing refers to the combination of computing resources that belong to multiple administrative domains to executive and achieve a common final goal. In order to reach such a goal, the Grid is recognized as a distributed system composed of non-interactive processes and a large amount of files and data exchanges. This characterizes Grid computing as a non-conventional high performance computing system in which the resources that belong to the system are loosely coupled, heterogeneous, and geographically dispersed. With varied magnitude, the Grid provides transparency for the access to its resources, enabling and simplifying the submission and management of workload by presenting a super virtual computer view to end users. However, in reality this system uses its computing servers to perform highly large and complex tasks. In contrast with the traditional notion of supercomputers, Grids relies on loosely-coupled computers and

processes applications that use message passing that mostly depends on conventional network interfaces for internal communication.

The Open Grid Services Architecture (OGSA) [39] was designed within the Global Grid Forum (GGF) to provide an architecture for a service-oriented grid computing environment. Such environment was developed to be a solution employed in businesses and scientific applications. Such system and applications require interoperable, reusable, and portable components that can be used to build the envisaged management system. In order to enable these components, standardization is provided by the OGSA, which addresses the behaviours and capabilities that are major concerns in grid computing systems. Based on several Web service technologies, OGSA provides standardized methods that combine stateful web services – distinctly Web Service Definition Language (WSDL) and Simple Object Access Protocol (SOAP) – in the Grid architecture. With the service-oriented architecture, OGSA assures interoperability for different types of resources and applications on heterogeneous systems. The OGSA grid is described in terms of the capabilities documented in its specification: infrastructure services, execution management services, data services, resource management services, security services, self-management services, and information services [39]. Therefore, Grid services encompass the monitoring of resources and distributed applications, the scheduling and allocation of resources, and the identification of application requirements [68]. Due to the features described in its specification, OGSA has been adopted as a grid architecture design by a variety of grid projects, including the Globus Alliance. Globus Toolkit [1] became the *de facto* middleware standard for Grid computing.

Formerly, Grid systems have defined components that are essential for enabling Grid computing to user applications. These components consist of user interface for allowing the access to the distributed system, security for maintaining the integrity of resources and applications, workload management for determining available resources, scheduler for assigning application jobs to a resource (computing server), data management for transferring, exchanging, and accessing data, resource management for monitoring and submitting jobs on specific remote resources, information service for equipping directory services to identify resources and their load status. When applying OGSA to define such components, combining resource interconnection resource techniques to solve complex problems emerge: composability of resources in systems; dynamic aspects of systems

(load balancing – jobs assigned to resources and jobs finalizing their execution); seamlessness (reduction of the need of external intervention for management); and greater generality, ubiquity, and mobility of computing resources and applications. Thus, to comply with the defined components and techniques, Grid services are basically delimited and organized according to Figure 2.2. In the architecture, the Access Interface provides simple means for users to access resources and services and launch applications. This interface employs security aspects, such as Public Key Infrastructure, to authenticate and authorize the access and maintain confidentiality and data integrity in the system. The Index Service and Monitoring provides Monitoring and Discovery Service (WebMDS) which keeps a database that contains the status of resources and applications. Data Replication and Data Management uses secure and reliable mechanisms to move or make accessible data or application modules to the nodes where the application's jobs are designated to execute. This service employs Reliable Data Transfer (GridFTP) to transfer data with authentication, authorization, and delegation of access. The Job Submission service uses Grid Resource Allocation Manager (WS-GRAM), which provides means to launch jobs with delegation on a particular resource, check their execution status, and retrieve their results. The WS-GRAM allows the utilization of schedulers, which identifies available resources to assign jobs to run, determining the most appropriate resources to use in the Grid system. However, these services do not resolve load imbalance issues although they do provide basic instruments for balancing systems. Thus, they can be accessed by a balancing system to retrieve the load of computing resources and to perform the load transfers.

2.3 Architectures Supporting Large-Scale HLA-based Simulations

Even though HLA provides an RTI service called Data Distribution Management (DDM), which presents mechanisms for minimizing the communication overhead, the HLA standard is unable to offer any solution to balance the computational load and communication delays. Even though HLA provide means to accomplish distributed simulations, it offers little support to large-scale distributed simulations [104]. The HLA standard does not present any mechanism or method for controlling the distributed simulation load placed

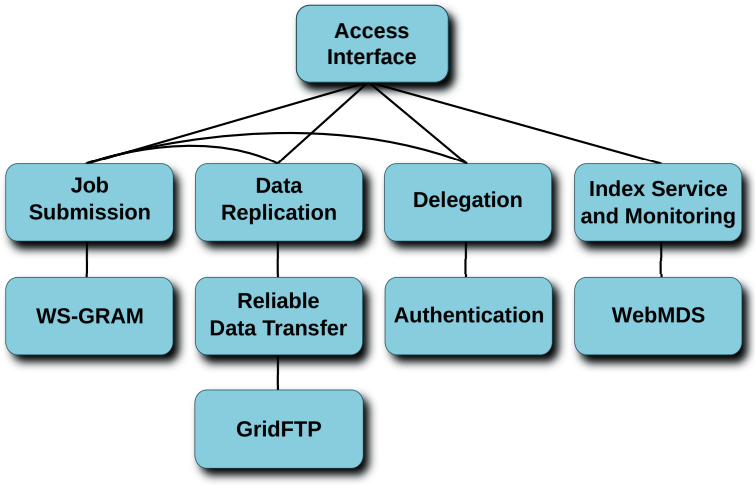


Figure 2.2: Grid Services Conceptual Architecture

on shared resources or handling the shared resources on which simulations run. HLA relies on the fact that all available resources are dedicated, show reliability, and present enough computing power and communication capacity for simulation applications. Because the performance of HLA simulations can be jeopardized by only one overloaded resource or a single local failure, management systems are required to coordinate the deploy of simulation federates on resources of certain distributed environment.

The presence of unreliable resources on distributed environments grows proportionally to the environments. This unreliability originates from the chances of a failure occurring in distributed systems, which increase with the amount of components that constitute the environment. In the case of HLA simulations, it is assumed that the distributed environment on where the simulation federates are deployed is faultless [76]. No formal failure model is included in the HLA [59] for eventual issues regarding fault-tolerance. Since large effort is applied to supply and organize resources for large-scale computation needs, the management of these resources is required to provide processing robustness and reliability [53] to simulations. Without such a system, the whole distributed simulation can easily fail due to a single fault in a resource.

On the other hand, the non-dedication of resources and the limitations of their processing and communication power capability significantly condition the simulation performance. In distributed simulations, the HLA standard totally relies on the availability of distributed systems resources [98]. Because of this assumption, HLA simulations do

not fit in large-scale simulations whose resources do not have complete availability and reliability. Even though the HLA standard presents mechanisms to coordinate distributed simulations, it does not provide tools to manage the execution of a simulation on distributed resources [99]. HLA standard does not support federate migration, which is essential for allowing dynamic setup of federates on shared resources [78] [58].

The OGSA Grid services can be used to introduce fault-tolerance in the distributed simulation [98] and can allow the management of simulation load according to resources capabilities. With a very general architecture, OGSA does not differentiate specific distinct services and are suitable for supporting HLA simulations. Grid computing presents ubiquity of peer resources, and Grid has a robust and transparent architecture that allows resources to discover each other over wide area networks without control of the infrastructure [72]. By using Grid systems, HLA federates can be viewed as system resources because they exist as simulation resources with complete transparency to each other. Since HLA does not offer any dynamic discovery mechanism for the simulation and its elements [98] and rely on common configuration files that specify the location of controlling components [100], Grid services can provide the dynamic discovery of simulation elements. Grid services also cab enable the dynamic reconfiguration of simulation load, which is not present in HLA because it does not offer automatic deployment and migration of applications (federates) [99]. Because federates are not independent and may only communicate via the RTI that has to keep track of every migration, general purpose load management systems, such as Grid systems, can not be uniquely used [58]. Therefore, a management system that integrates or employs the Grid services capability is necessary fro enabling large-scale HLA simulations.

2.3.1 Resource Sharing System

The Resource Sharing System (RSS) and the Fault-Tolerant RSS (FT-RSS) are both devised to manage HLA-based simulations on large-scale environments. This management comprises the monitoring of simulation load and the detection of faults during run-time. According to the issue detected, the system reacts and performs modifications on a simulations or on the distribution of their entities (federates).

2.3.1.1 Load Balancing in the Resource Sharing System

As the main part of the developed system, the RSS manages the execution of HLA federates over a distributed system by managing the load of each system's node [58]. The system organizes the load by transferring federates from one system node to another in order to introduce load balance around all the distributed resources. Thus, the RSS provides mechanisms to identify when it is necessary to transfer simulation components, to accomplish their migration and to configure the behaviour of balancing the load. Moreover, users can control the availability of their computers in the simulation system. Therefore, the manager, following the rules imposed by users, can administrate the simulation, and the users' computers receive load not more than what was specified by them.

In order to provide such load management, the RSS is composed by a manager, clients and a communication federate, as present in the figure 2.3. The manager administrates the load over the distributed system's nodes and is the users' interface to the simulation system. The clients are the users' interface to the RSS, and they manage the simulation's federates that are running on their respective client node. The communication federate is the bridge between the manager and the simulation federates. This special federate passes the manager's requests to simulation's federate by translating them into the RTI interactions.

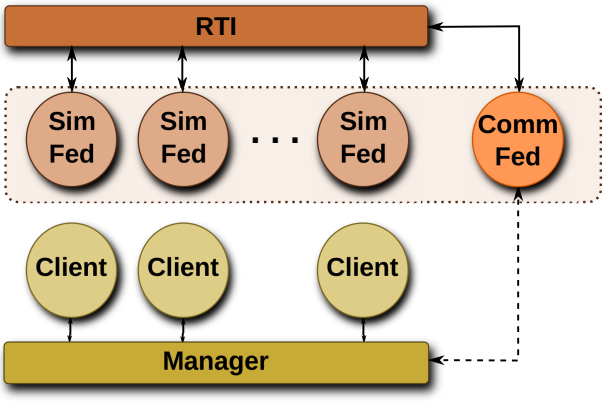


Figure 2.3: General architecture of RSS.

In the developed system, load migration at the federate level is employed to keep the consistency on RTI communication level. As requested by the RSS manager, federates

save their state on a file and are restart using the saved state file on another node. Moreover, in order to minimize modifications on simulations, the simulation federates do not communicate with the manager directly. All the communication between the manager and the simulation federates is performed via the so-called communication federate that uses RTI to communicate with the other federates. Acting as a common federate, the communication federate uses the RTI interactions to forward the manager's requests to a specific federate and accomplish the federate migration.

In the implementation of the system, the RTI, the RSS manager, and the communication federate are located on a server node. Each node receives a number of federates and a client program, which manages the execution of a set of federates and helps the migration. The RSS manager is the central component of the RSS. It processes incoming requests from the client programs, including connection and disconnection requests. The manager performs migration decisions for simulation federates and controls the migration operations.

In order to accomplish the load balance, the manager keeps track of client nodes joining the RSS by mapping them and distributing federates among the available nodes. The manager issues migration calls to a communication federate, which communicates the migration directions to federates via the RTI. With the use of a communication federate, time management conflicts during migration of a federate are avoided. The communication federate is viewed as part of the HLA federation to reduce the adaptation effort for the simulation federates. Non-RTI communication with the RSS manager is integrated in the simulation via the communication federate. However, not all the communication involving the RSS manager is realized via RTI because the manager does not participate in the simulation.

The RSS presents major drawbacks related to its design or to the mechanisms employed to redistribute load. The entire system relies on a centralized design in which only one element aggregates all the information and decides about load redistribution. This characteristic imposes major scalability obstacles to the system. Moreover, the system does not provide any details about the balancing algorithm, so no observation can be applied to measure its balancing efficiency. The system employs the interfaces available by HLA to save and restore the migrating federate. This approach freezes the entire simulation, imposes considerable overhead on simulations, and substantially limits the

balancing responsiveness.

2.3.1.2 Fault Tolerance in the Resource Sharing System

Based on the resource sharing system (RSS) depicted in 2.3, a FT-RSS is devised to allow individual configuration of fault tolerance mechanisms through a FT configuration tool [59]. The system aims to integrate, administrate, and execute fault tolerance mechanisms into HLA federates and federations. The choice of fault tolerant mechanisms is dependent on the specific simulation requirements based on failure assumptions for federates and computing nodes of a distributed system. Also, not every federate of an HLA federation will have the same requirements with respect to dependability and fault tolerance.

In order to introduce a configurable fault tolerant system into RSS, RSS needs to be extended by inserting fault tolerance configuration files, extending the RSS manager to a FT-RSS manager to control error detection, error processing, and migration according to the configuration file, adding support checks on clients to detect errors and process them, and introducing a FT monitor to allow observation of the error and failures. Thus, the framework consists of a FT configurator, a set of FT code modules and interfaces, and a fault tolerant resource sharing system (RSS). FT requirements determine the choice of error detection and error processing mechanisms for federates. A FT monitor can be used to visualize detected errors and initiated error processing mechanisms.

The fault tolerance configurations are divided in aspects that can be configured for individual federates and aspects related to whole federation. The aspects that are specific to a federate are configuration of migration possibilities, configuration of error detection mechanisms, and configuration of error processing methods. Fault tolerance configurator specifies the migration behaviour through a node pool that identifies node addresses that can be migrated to, a list of restrictions that prevents the migration to some nodes, and a list of operating system assignments whose federates can be executed on.

The detection of errors is observed only on simulation federates execution according to the following basic aspects. First, federate execution errors are not considered since they hardly can be identified. Second, fail-stop failures are easily determined through functional or structural checking. Third, partial failure is another option that consists of separating the federate's failures into several failure regions. Finally, the replication checking is used for error detection and error processing by grouping federates in redun-

dancy regions that detect errors by absolute and relative majority decision, pair decision, and unanimity decision.

The error processing configuration is divided into four basic options that are described as follows. The first option is the "no error processing" that does not apply any processing about detected errors. Another option is to stop only the federates involved with a failure. The forward error recovery is another option that restarts the execution of a federate with its initial state or using a specified valid state. The last option is the backward error recovery that recovers the federate using a previously saved state. In this case, timing checkpoints are performed periodically, computed using an underlying failure model, or triggered by certain simulation events.

2.3.2 Load Management System

Cai et. al. [24] developed a Load Management System (LMS) to support the execution of HLA-based large-scale distributed simulations over geographically distributed computing resources efficiently and effectively. LMS should provide mechanisms for load-balancing, faut-tolerance, coordination of simulation execution and security. They combine an existing resource sharing system, in this case Grids, with the Run Time Infrastructure (RTI) of the HLA.

Their approach is different from the system reported in [58], which was not built using any existing resource sharing system. That system requires more effort for development and may not be able to take full advantage of the Grid because all the resource management must be implemented. The authors use Globus toolkit [38] [37] to enable Grid Computing because of its availability, completeness and popularity. The major modules in Globus that are used in LMS are Grid Security Infrastructure (GSI), Grid Resource Allocation Manager (GRAM) and Grid Information Service (GIS). Therefore, LMS manages resources over the Grid for distributed simulations. While the RTI is the infrastructure for supporting the execution of the simulation federation, LMS manages the resources where the simulation executes, as shown in the figure 2.4.

As depicted in the figure 2.5, LMS consists of two sub-systems: job management sub-system (LMSJobs, LMSHandlers, and the load manager) and resource management sub-system (LMSClient and LMSServer). The job management monitors the execution of federates and performs load-balancing using Grid services like resource discovery and

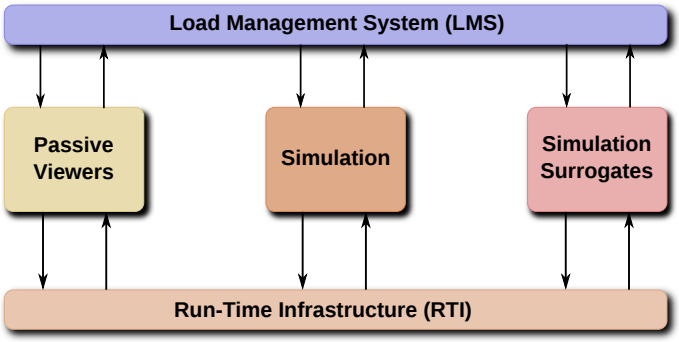


Figure 2.4: Load Management System interacting with High Level Architecture.

monitoring.

For each federation execution, LMS creates a Load Manager (LM) which contains several instances of LMSJobs and an LMSClient and assigns federates to computing resources to balance the load. LMSJob corresponds to the federate, and LMSClient gets a request from LM and passes it to the LMSServer which gets information about resources, requests for resources, starts or restarts a job. In the system, GSI is used to authenticate the client connection, GIS is used to obtain information about available resources, and GRAM is used to start execution of jobs remotely.

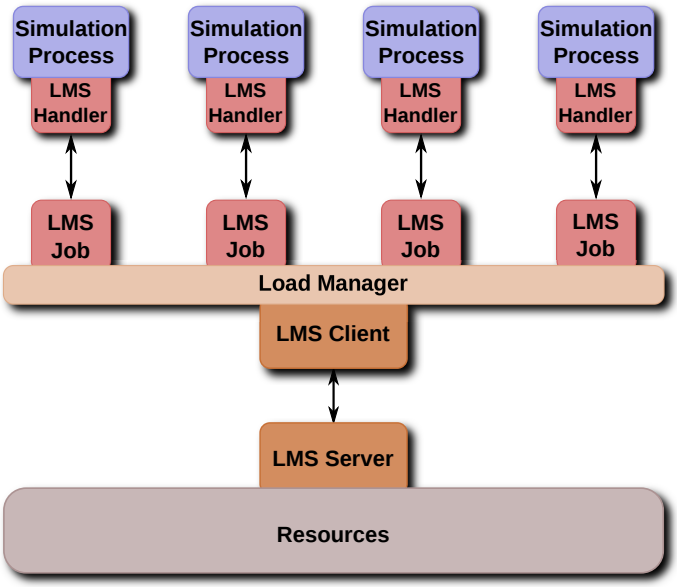


Figure 2.5: General architecture of Load Management System.

Regarding load balancing mechanism, when the system needs to migrate a federate, the LMSJob sends a message to the corresponding LMSHandler to suspend the job. Using the interface provided by the LMSHandler, a federate can detect the condition and save the state of simulation objects into a file. The LMSHandler then sends the state information to the corresponding LMSJob. The LM sends restart job request to the LMSServer. LMSServer restarts the job execution at a selected host. LMSHandler needs to receive a state info message from the corresponding LMSJob. After the connection information is passed to the LMSHandler, the LMSServer restarts the job. In order to transfer the state saved by LMSHandler, GSI FTP services are used in the system to authenticate the access to resources increasing the security over data. However, the use of a third party to transport data brings an overhead on the communication.

In normal execution, LM starts a job (federate) by asking LMSServer to allocate a resource. Once the federate is executed, an LMSHandler will be created. The LMSServer makes connections to the LM by sending a connect request message. After receiving the message, LM creates the corresponding LMSJob. The created LMSJob will then inform the LMSHandler to start the execution of the federate. To suspend a job, the LM sends a suspend job message to LMSJob which sends a suspend federate message to LMSHandler. After receiving the message, LMSHandler will save the federate's state and send the state info to LMSJob.

2.3.3 SimKernel

Yuan et. al. [96] presents the SimKernel framework aiming to facilitate the simulation's design and to overcome problems of running parallel and distributed simulations using the RTI. The HLA standard introduces difficulties for users to write simulation applications. Also, the HLA does not provide tools for load balancing of Grid resources. Thus, the framework introduces a simpler simulation modelling and facilitates the federate's checkpointing.

The SimKernel framework consists of three major components: a federate execution model, named SimKernel; an user interface that allows modellers to specify essential simulation information at process level; and an automatic code generation tool that translates the modeller's design into executable codes. A new layer between the user and the HLA simulation is introduced to translate the user's simulation design into a

pseudo-code whose calls are translated to RTI calls in order to be processed.

According to the authors [96], their framework is composed by logical processes (LPs) and messages, which respectively correspond to the modelling of physical processes and the messages models of their interactions. Comparing this model to the HLA simulations, the LPs are mapped as federates and their communication (messages) are mapped as RTI interactions through an automatic code generation component. An associated library code is also presented as a complement of the middleware to translate from the authors' model to the HLA standard, such as described in the figure 2.6. After the SimKernel model is parsed, an intermediate configuration file is produced and used to generate the federate's code.

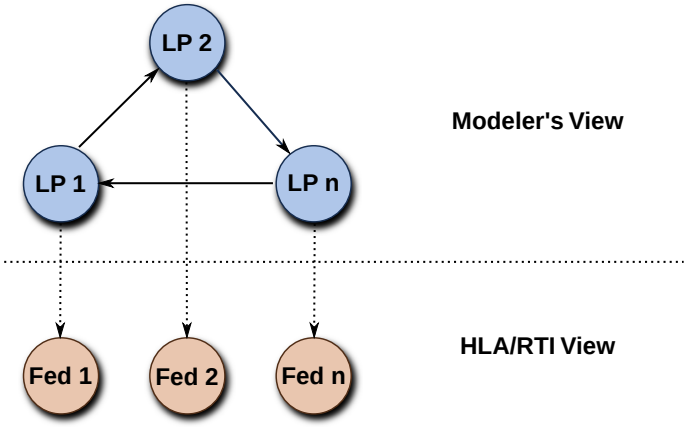


Figure 2.6: Federate design in the SimKernel approach.

According to the modelling approach and as described in Figure 2.7, the events in the simulation are defined in a SimEvent subclass and they are processed by a method called consume in the SimEvent class. Moreover, the SimKernel contains two queues called inQ and outQ which are used to manage the incoming and for sending events (interactions). These queues store all the interactions that arrive in the SimKernel and those events that will be sent to the RTI. An interaction is stored into the outQ if it needs to be sent to the RTI, and an interaction is stored into the inQ if it needs to be sent to the same SimKernel element. Whenever a HLA interaction is received, it is converted to a SimEvent and stored in to the inQ. Before events are stored in outQ to be sent to the RTI, they are converted into RTI interactions.

In order to keep the simulation consistent, the SimKernel manage the time based on

the HLA event-based time management. SimKernel negotiates the time advance with the RTI to be able to consume the interactions in the inQ. Also, DDM is used in the SimKernel to filter the messages to be transmitted.

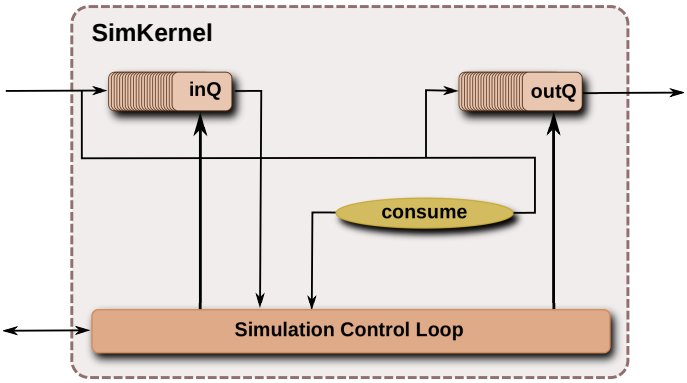


Figure 2.7: General architecture of SimKernel.

As an extension to the SimKernel architecture, Yuan et. al. [97] introduce a migration mechanism to decrease migration overhead by avoiding global synchronization and use of third party mechanisms. The use of HLA standard interfaces (federationSave and federationRestore) imposes major migration latencies into simulations with the synchronization of entire federation. Non-migrating federates are required to participate in the federation save and restore process for every migration request. As a result, the overhead caused by a migration is proportional to the number of federates in the simulation.

A shadow object concept is employed in the migration approach for SimKernel. The shadow object approach is used in avatar migration [47]. In order to migrate the federate, the federate's state is saved and transferred together with its code and a shadow federate is started on the destination node. The messages received at the old host are sent only when the migrating federate restore its subscription to its event interests, and the duplicates are removed from the inQ. In order to migrate a federate, the transfer of the program's code and essential execution state is required. With the SimKernel framework's standard library is placed at the destination host, the migrating federate can be dynamically reconstructed at the destination with the LP specifications and the received events stored.

The SimKernel system for migration consists of two subsystems, named simulation subsystem and load management subsystem, as shown in Figure 2.8. The SimKernel

uses the HLA/RTI and the Load Manager (LM). The LM determines the destination host for the federate to be migrated. The LMClients at the source and destination hosts communicates until a successful migration is achieved. LMClient performs three major tasks: it monitors the load level at the host and reports to LM; on receiving a migration request from the LM, the LMClient migrates the selected federate using a protocol; and the LMClient creates, suspends, communicates with, and destroys federates when necessary.

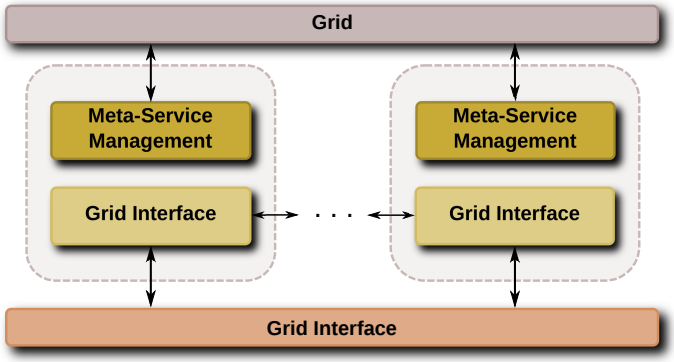


Figure 2.8: Extended SimKernel architecture for federate migration.

2.3.4 Grid HLA Management System

Grid HLA Management System (G-HLAM) manages HLA-based interactive simulations that run over Grid environments, supporting migration and providing monitoring mechanisms to simulation applications [79] [77] [80]. The system acts as an interface to Grid services and prepares them to be accessed by simulation applications [75] and introduces mechanisms that can be used for load balance of simulations and for fault-tolerance of unreliable Grid resources [78]. The Grid services help to overcome the HLA requirement of explicit description of data and event objects before the simulation starts [88]. Its design is based on OGSA and keeps backward compatibility with legacy HLA codes [78]. In G-HLAM's design all its services are provided as Grid services which benefits interoperability and service discovery.

The system was first presented in [98], defining the system's main characteristics and the functionality of the most important components: service discovery, backward

compatibility with legacy HLA code, load balance, fault-tolerance, and simulation consistency. In [22], the system's major realm features were delimited as user-interactive simulations and user visualization. To enable such features, the proposed system aims to provide monitoring, distributed data access, and resource management.

As presented in Figure 2.9 [79], the G-HLAM architecture is composed in general by a Broker Service, a Performance Service, and a Registry Service. The Broker Service organizes the simulation management by determining the best resource to receive a federate [76]. The Performance Service determines the load redistribution. The Registry Service keeps information about local services which perform migration, monitor federates and benchmark the simulation on behalf of the Broker Service. In order to gather information about federates, the Broker accesses the Registry Service and monitoring services, which includes the Main Service Manager and Service Managers (Application Monitoring Services). The Broker Service's work is directly related to the simulation since it groups federates that tend to communicate intensively [76].

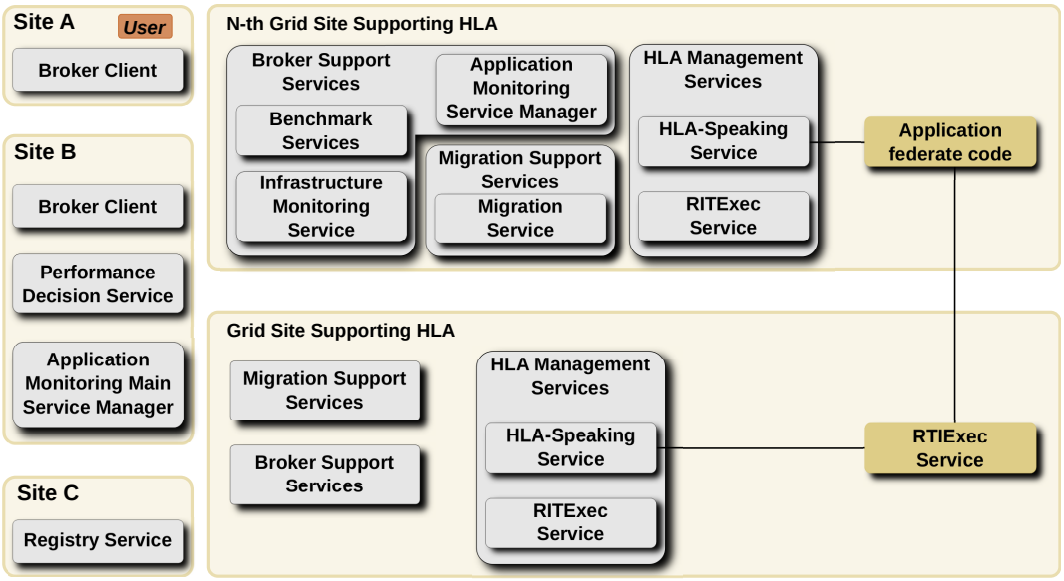


Figure 2.9: General architecture of poland's framework.

The system's architecture is also composed of HLA Management Services, Broker Support Services and Migration Support Services. The HLA Management Services comprise the HLA-Speaking and the RTIExec services which manage and interface federates and the RTIExec. The Broker Supporting Services provide information needed to mi-

grate load. The Migration Support Services perform federate migration improving simulation performance and introducing fault-tolerance [74]. Local Monitors and Migration Services are responsible for federate run feed-back and check-pointing. Integrating the Broker Service, the Application Monitoring Services and the Benchmark Services provide monitoring and benchmark information about the simulation components, as depicted in Figure 2.9. The monitoring mechanism is composed of an application monitoring system called OMIS-Compliant Monitoring system for the Grid (OCM-G) [9], which is a support subsystem of G-HLAM that provides information to Broker Service.

The OCM-G Monitor [9] is a distributed and decentralized mechanism that monitors Grid applications during run-time. This monitor consists of a per-host Local Monitors, per-site Service Managers, and a Main Service Manager. The Local Monitor resides in a host and monitors the Grid applications locally. The Service Manager is responsible for gathering data from a set of Local Monitors. The Main Service Manager collects the information from the remote Service Managers and delivers to performance analysis tools.

The monitoring subsystem is composed of a Performance Decision Service, a Main Service Manager, and local Service Managers [74]. The local Service Manager monitors applications on the site where it resides. In order to collect data about HLA applications, it is inserted a wrapper between the federate and the RTI/Federate Ambassadors which will send application monitoring data to OCM-G transparently. The Main Service Manager collects information from each Service Manager and passes the gathered data to Performance Decision Service which triggers the Broker Service to start a migration process. The Performance Decision Service keeps a graph of connections and accesses Benchmarks Services to compare and keep minimum communication requirements.

HLA-Speaking is the most important component in the system because it manages the federates' run by starting, saving, and restoring them [78][99][98]. Because of the HLA-Speaking Service role, it acts as an interface for G-HLAM to control the simulation components [77]. The service manages many federates' execution locally and directly in a site, and it performs federate migration. It incorporates a Globus GRAM component called Resource Specification Language (RSL) [1] to describe and identify the exact federate that needs to be managed [78]. All the HLA-Speaking Service actions are accom-

[1] http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html

plished through the GridHLAController Library which performs the communication with the control federate. It locally sets up, saves, and restores a federate on the resources, and it performs all these tasks by accessing the GridHLAController library which is a layer between the G-HLAM and the simulation.

The GridHLAController Library facilitates and makes the process of federate migration and management more transparent [99]. The library encapsulates all the migration management service process in method calls, interfaces the HLA-Speaking service to the user code, and uses the native HLA methods for saving and restoring the federate state. The Library provides functions for setup, for RTI connection, for checking external migration or restore requests, for checking internal RTI saving/restoring requests, and for saving and restoring user-specific values [80].

As presented in the figure, the system is accessed by the user through the Broker Client, which requests the Broker Service for the application deployment [78]. The Broker Service, accessing the Registry Service, determines a HLA-Speaking Service from a list and an RTIExec Service to run the submitted application. After that, the application is submitted to a HLA-Speaking Service that starts it in a host locally.

The deployment of applications occurs when federates are submitted to the simulation system or when federate migrations are requested. Migration in the system is classified in three levels: RTI layer migration, Federation layer migration and Federate layer migration [100] [76]. The RTI layer migration refers to the publication of the RTIExec and RTI Libraries as Grid services, which can be discovered by federates to initialize their run and the federation [88] [100]. The Federation layer migration comprehends the communication between federates, which is accomplished through GridFTP [88] [100]. The Federate layer migration comprises the registration, the discovery and data transfer of interactive applications (federates) [100]. In this layer, the HLA events are translated to Web services standard to be transmitted, GridFTP will perform communication, and HLA data and events will be described dynamically through Object Description Repository Service.

In order to keep the process transparent and avoid simulation failures, the migration implemented in G-HLAM is focused on the RTI layer migration [76] and provides simulation and system management services as Grid services. Moreover, in the G-HLAM, the migration logic is transferred from the Migration service to the Broker service because a

global view of the system facilitates to determine the best location to set up or to migrate a federate [78]. Also, several migrations are performed together in order to minimize the time spent with freezing the entire simulation.

Migration is accomplished through Migration Service calls to HLA-Speaking Services [78]. The HLA-Speaking Service interacts directly with federates and with another HLA-Speaking Service to start and stop de federate's execution, save and restore their execution state. One call is made to create a new HLA-Speaking service on the remote resource, and another one is made to trigger the HLA-Speaking service to save federates' state. The new HLA-Speaking Service generates a control federate that creates the control federation. The old HLA-Speaking Service, by accessing the GridHLAController Library, requests to save all the federates it controls. The new HLA-Speaking Service sets the context needed to run the federate code. After such federates are checkpointed, G-HLAM sends their state to the new HLA-Speaking Service. After that, the federates' codes are transferred to the remote location. At the end, the new HLA-Speaking Service restores them from the saved state at the remote location by calling the GridHLAController Library.

In order to save and transfer the federate's state, it is used GridFTP services for reliable data transfer in Grids and the Federation Management API to save and restore the simulation's state [100] [98] [99]. The HLA API freezes the entire simulation to perform the migration process of only one federate. It uses freezing solution in order to keep the simulation consistent without modifying the RTI implementation.

2.3.5 HLA_GRID

HLAGrid is a distributed simulation framework which uses a Federate-Proxy-RTI architecture allowing resources on a Grid to be utilized on demand through its services. The architecture also supports federation discovery, security of the simulator logic, and flexible federation construction [94]. Moreover, the authors aim to provide services to support HLA-based distributed simulation on Grids and a flexible construction of federations. Their proposed system aims to support the provision of RTI services on demand, to aid federation dynamic discovery, to introduce a standard HLA API, to allow communication through firewalls, and to present a hierarchical federation architecture.

The proxy is the main component in the framework architecture and acts on behalf

of a federate hiding the heterogeneity of simulators, simulators' execution platforms, and communication channel of simulators with the RTI. RTI services are presented as Grid services, and they are called through Grid service invocation, which increases the security, scalability, and coordinated management, as depicted in Figure 2.10 [94]. Also, the architecture supports federation dynamic discovery and hierarchical federation, and it introduces reusability by being compliant with the HLA standards.

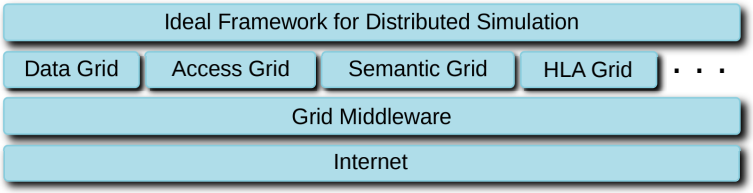


Figure 2.10: Layered architecture for HLA_GRID framework.

The proxy concept, used in the framework structure, was first introduced by Xie et. al. [95]. In this work, the system presented is a framework that uses a federate-proxy-RTI architecture as a backbone to run HLA simulations over Grid systems. This architecture is composed by a remote proxy that acts as a middleware between federates and the RTI in the simulation. The proxy provides communication transparency and simulation component reusability; thus, the federate runs in the client side containing only the simulation model, and this architecture facilitates federate migration, as well as the backbone's migration [95].

The main goals in designing the backbone are to introduce a standard HLA API according to Defense Modeling and Simulation Office's (DMSO) implementation in order to maintain interoperability and reusability aspects, and to solve communication limitations caused by firewalls. According to the authors' design, federates are separated from the RTIExec and FedExec by the proxy that executes remotely, as presented in the figure 2.11 [95]. The proxy forwards the federate's calls on behalf of it, and a federate's proxy communicates with a remote federate's proxy accessing the RTI. The communication between federates and RTIExec is made through the Grid-enabled HLA API that invokes Grid services. The Grid services provide means to create RTI through a persistent RTI service, to discover federations through a persistent indexing service and to access simulation's services. Moreover, the framework is composed by clients, resources and

Grid services. Federates locate on the client side which provides the standard DMSO HLA API and uses the Grid services invocations through Grid-enabled API. The proxy is located in the resource side, and it forwards the federate's calls to the RTI after it translates back the Grid service invocations to RTI invocations.

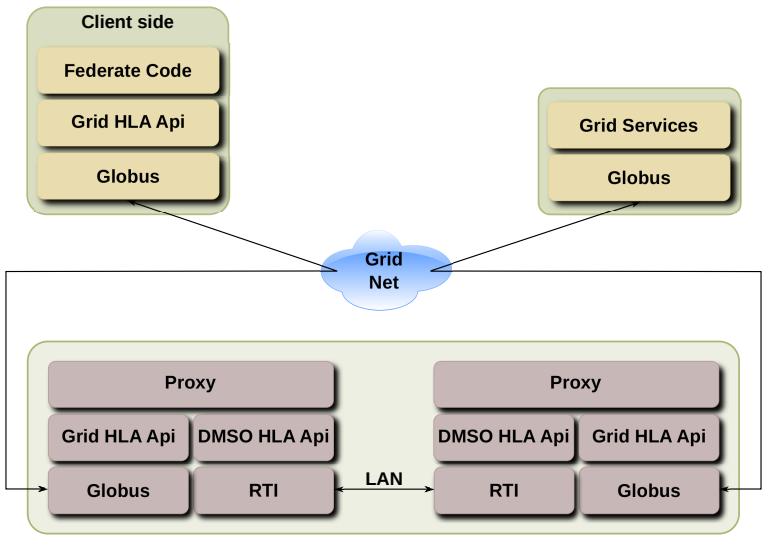


Figure 2.11: General architecture of HLA_GRID system.

In the same way, the HLAGrid framework identifies HLA simulation (RTI) services as Grid services. Such usage was first described by Zong et. al. [105] who presented a framework that used Grid services to run large-scale distributed simulations and to provide services of dynamic discovery and resource usage. It overcomes the the HLA simulation communication issue, which needs to be established by explicitly configuring the simulation components' endpoints. Moreover, the framework keeps HLA RTIExec managed by RTI services, provides Federate services that encapsulate the simulation models, and introduce a central Index service for these Grid services. Thus, complex simulations can be created by assembling services that are discovered through Grid services.

The framework proposed by Zong et. al. [105] is composed by an Index Service, a RTI Factory Service and a Federate Factory Service. Such services are implemented based on Globus Toolkit (OGSA) and made available as Grid Services. The Index Service maps the federations to handle RTI Services, answers queries from federate clients and updates status by the RTI service. The RTI Factory Service generates RTI Service instances, and each instance is responsible for managing and collecting information about the execution

of an RTIExec process. The RTI Service is registered by itself in to the Index Service when it is created. The Federate Factory Service generates Federate Service instances, and each instance creates federate processes, provides federate status information, or terminates a federate's run. It also provides discovery services in the Index Service, and it can be grouped to form more complex distributed simulations.

As stated previously, the HLAGrid framework is structured as a Federate-Proxy-RTI architecture [94]. This architecture divides the simulation in local federate codes and remote execution of RTIExec and FedExec, and a entity called proxy is introduce to bridge the communication between the simulation parts, acting on behalf of client's federate code. The federate communicates with remote proxies through Grid services by accessing Grid-enabled HLA API that translates federate's calls. Because of the use of Grid services to create RTI and discover federation, RTI can be created, discovered, and used on demand. Moreover, users can join the federation locally, remotely and hierarchically. The architecture provides users transparency and simulation interoperability and reusability, and transparent federate migration is facilitated. However, the proposed framework adds overhead in the communication due to the use of Grid services and the proxy.

As presented in the figure, the framework is composed by a client and resources, and they are connected by a Grid network [94]. The HLA API is in the client side and is the communication interface to the Grid, and the client and resource sides contain a Grid-enabled API that converts the federate-RTI communication into Grid invocations and conversely. Consequently, all the calls in the RTIAmbassador and the callbacks in the FederateAmbassador are performed through the remote Grid services invocations. The proxy separates the client and the RTI, keeping the simulation logic at the client side, and it provides only federation and time management services to the simulation. The use of a proxy facilitates design of freeze free federate migrations because it can respond to the RTI during the migration process. The proxy, acting on behalf of the client, forwards its communication to the RTI through a local area network (LAN). The Grid services also are used to create RTI and discovering federation by using RTI service factories that create instances of RTI services and indexing services that identify the handles for corresponding RTI service instances.

2.3.6 Grid-based Distributed Simulation Architecture

Grid-based Distributed Simulation Architecture (GDSA) is grid-based distributed architecture that supports fault-tolerant computing and QoS-based scheduling [104], and HLA simulations could be improved by integrating RTI services on GDSA. The six RTI services are wrapped as Grid services. The system focuses mainly on four pending problems in distributed system: scalability, communications, management mechanism, and QoS insurance computing environment. Contractual Computing Mechanism (CCM) is added to GDSA in order to provide QoS insurance for users. There is a three-layer QoS model to treat the QoS problems in different levels.

GDSA integrates Grid technologies in distributed simulations to provide scalability and uniform communication methods, and contractual computing mechanism (CCM) is added on its architecture to attend QoS requirements. CCM and QoS model help to improve efficiency and insurance of simulation environment. The use of Grid serviceSimple Object Access Protocol's to improve large-scale HLA simulations is limited on providing computing resources for simulation applications, and all the communication and control are accomplished through RTI. However, the integration of Grid services on HLA simulations has limitations in scalability, management, and QoS support. Thus, CCM is added to the architecture to overcome the mentioned limitations and provide insurance means for grid users.

Because the Grid technology is based on OGSA or WSRF, its communication and service provision is accomplished according to client/server architectures. Figure 2.12 shows the integration of RTI management services with the Grid technology. The use of Grid that introduces Service Oriented Architecture (SOA) has as the main feature scalability.

The communication with RTI is replaced since it presents many limitations in modern distributed large scale simulation applications. In GDSA, messages are passed in standard form of XML with the terms defined in SLA (Service Level Agreement) in order to negotiate minimum service requirements.

Moreover, GDSA introduces Meta-Service mechanism to deal with the management problem of high level simulation services over a grid system, and it employs a QoS-based scheduling mechanism, as shown in figure 2.13. Meta-Service is used to manage other grid resource services, and it has scalability characteristics since it is based on

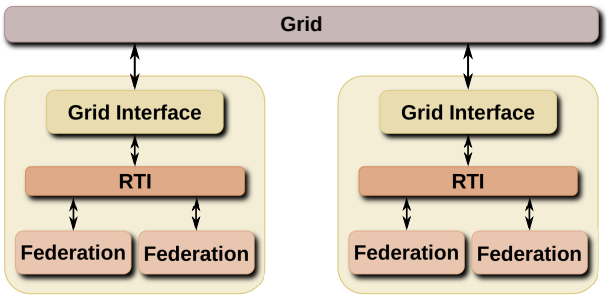


Figure 2.12: General architecture of GSDA.

service oriented management methods. Also, on Meta-Service architecture, QoS-based scheduling is achieved through Contractual Computing Model (CCM) and three-layer QoS protocols.

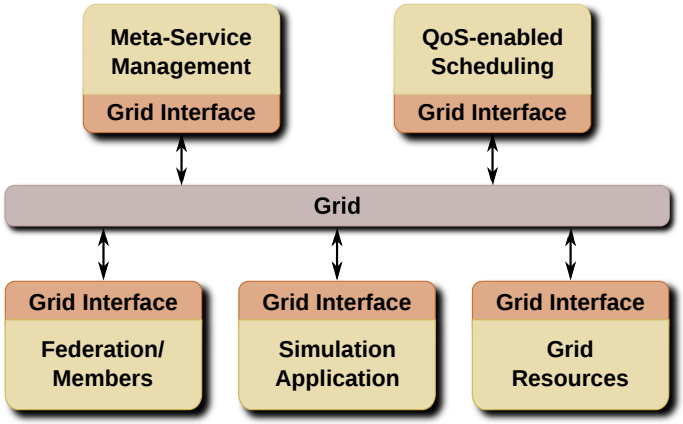


Figure 2.13: General architecture of GSDA.

In CCM, the QoS problems in distributed simulations are divided in three categories: base level, service level, and system level. Base level QoS guarantee is related to the guarantees of base operations inside a service. It inserts a contract that mainly focuses in a standard method and efficiency of job submission, advance reservation or pre-establishment of resource preferences that schedules resources according to WS-agreement, service parameterization that adds service describing terms and service communication terms in WS-agreement, data transfer assurance to keep efficient data transmission, and end-to-end application execution that uses RMI and XML to extended WS-agreement. The other category is the service level that includes QoS agreement-

based workflow integration and system integration based on SOA. The integration is necessary to make services work together and behave as expected. It uses auto negotiation, auto-binding and auto configuration. QoS guarantee works as a management system based on Meta-service and QoS guarantee platform; it provides monitoring and scheduling mechanism for a distributed system and SOA.

2.3.7 Grid-Based Parallel and Distributed Simulation Environment

Grid-Based Parallel and Distributed Simulation environment (GPDS) addresses distributed simulation problems like deficient computing powers, weakness in fault and security problem, and it supports transparency and scalability using Grid technologies [54] [53]. In order to solve deficient performance or computing power and issues related to faults and security, three services are introduced: automatic distribution service, dynamic migration service, and security service. The services provide a supply of computing resources and robustness to distributed simulations. The services are provided according to a 3-tier architecture which consists of clients at front end, interaction server at the middle, and a network of computing resources at back-end including Databases.

As depicted in the figure 2.14, the system's architecture can be presented as a layered design composed by the simulation, a communication middleware, GPDS manager, Grid services and resources. The Grid Computing Environment (GCE) is responsible by the management of system's resources by comprising modules of Grid services in it. The Network Virtual Environment (NVE) is an application and middleware to communicate efficiently.

The GPDS manager bridges NVE and GCE, and it is composed of a Grid Agent (GA) and Simulation Agent (SA) which allow Parallel and Distributed Simulation Environment (PSDE) and GCE to harmoniously interact to each other. Based on a client-server model, the client in client tier delivers an executable and standard input files of required simulation to the GPDS Manager in Server tier. The GA creates a process to allocate simulations to remote hosts and the SA. A simulation process in SA accomplishes dynamic migration services. Also, the SA manages the databases in Database tier while it cooperates with GA. The result of the simulation is delivered to the client by the SA.

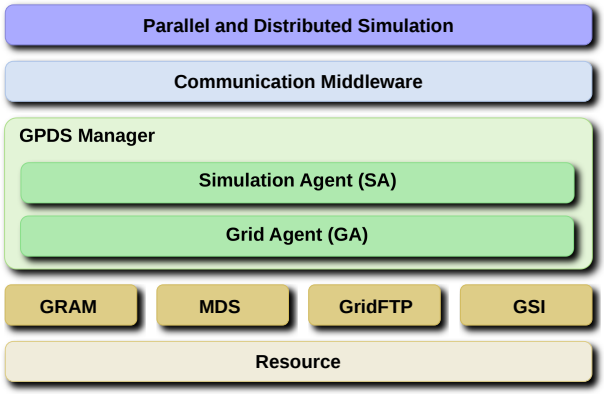


Figure 2.14: General architecture of GPDS.

GPDS manager also provides automatic distribution services by introducing dynamic migration strategies that transfer a computing from one host to another.

The HLA/RTI is used as simulation-specific middleware to provide stable communication and interoperability. Also, the migration on simulation view is accomplished through the usage of joining the migrated federate to the federation on the new server and resigning the old federate from the federation on the old server. However, a description about issues on the design of migration services for HLA components is not presented, such as how the federate state is saved and restored.

2.3.8 Extensible Modelling and Simulation Framework

The Extensible Modeling and Simulation Framework (XMSF) consists of a set of self-consistent standards, processes and practices that are used to model HLA compliant simulations [67]. XMSF employs a set of web-based technologies and services to enable a new generation of Internet-distributed applications to emerge, develop, and interoperate. The framework uses Web Services as a communication layer in its architecture, so it encapsulates RTI calls into SOAP packages, and in its implementation, it employs Blocks Extensible Exchange Protocol (BEEP) to transport the packages in a bi-directional communication channel.

The XMSF framework aims to enable a new generation of modeling and simulation applications to be developed through the use of web-based technologies. It also can be used as a bridge between forthcoming simulation requirements and web standards. Thus,

a long term goal of XMSF is to provide means to multiple federates use web services and be able to communicate to each other in a wide area network.

The RTI interfaces are formatted into SOAP packets to be transmitted using Blocks Extensible Exchange Protocol (BEEP). The use of SOAP provides cross-platform and cross-language solutions. Figure 2.15 shows the architecture and the functioning of the prototype that was developed based on XMSF. The prototype's federation is composed of two model elements that communicate to each other over the Internet in order to accomplish the simulation. Thus, the technology enables existing HLA compliant federates to be integrated easily over the Internet, including through most firewalls with minimal reconfiguration.

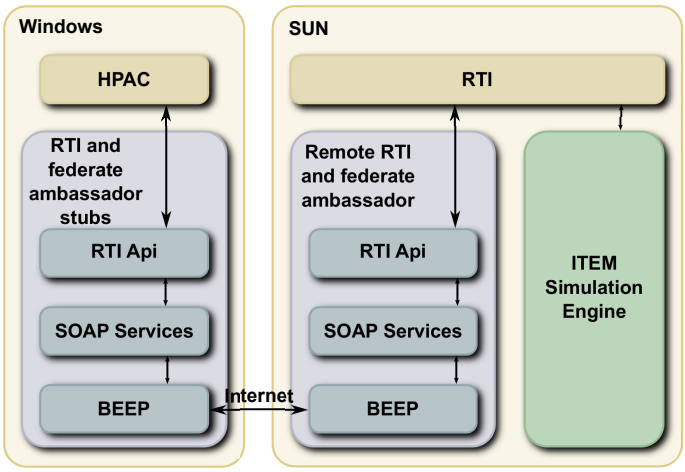


Figure 2.15: General architecture of XMSF.

Because Web services offer capabilities for scalable interoperation of heterogeneous software, XMSF expands the concepts of the HLA into the realm of Web services in [72]. Interoperability is introduced in HLA through Federation Object Model (FOM) and in web services through the creation of a common vocabulary. HLA FOM can be visualized as a common vocabulary in web services, and they can be expressed as an XML tagset to communicate to different applications.

In [66], the authors present a Web Services Internet Management (WSIM) architecture designed to achieve capabilities for interest management compatible with simulations using the HLA. The architecture uses the XMSF framework as its basis and extends the framework to treat interest management issues regarding the realm of web services. In

WSIM, capabilities are introduced to support both interest management and streaming updates from real time simulations.

Interest management is referred to as relevance filtering, data distribution management, or data subscription that reduces the received messages to a smaller, relevant set. The filtering reduces the network traffic and supports the user's need to define what information form the simulation will be displayed. The interest management in HLA is called Data Distribution Management (DDM). It takes the form of range specifications for generic dimensions not necessarily in geometric or geographic terms. Thus, WSIM extends XMSF by adding a Role-based Access Control (RBAC) component, an Area of Interest Management (AOIM) component, and an Aggregation Interest Management (AGIM) component into the system design.

2.3.9 HLA_GRID_REPAST

The HLA_GRID_REPAST manages large-scale distributed agent-based simulations over Grid systems [102] [87]. It acts as a middleware performing communication between parallel and distributed elements of a simulation. It is composed of the HLA_GRID [3, 12] and the HLA_REPAST [102] [64]. The first one is an architecture for harnessing HLA simulations over Grids. The second is a middleware to support agent-based distributed simulations with HLA-based services.

2.3.9.1 RePast System

The RePast system is a toolkit that provides several mechanisms and structures to manage and to develop simulations based on agent models [102] [64]. Such models are implementations of REPAST interface called SimModel [64]. The interface is composed of a Model and an Executive. The Model must make available an instance of Schedule class to the Executive in order to pass the sequence of events that need to be executed. The Models state changes as the events in the schedule are processed.

2.3.9.2 HLA_RePast

The HLA_RePast system manages simulations over distributed resources based on the RePast agent-toolkit [64]. Such a system allows the integration of several Java-based

simulation agent models. The system is a middleware introduced for connecting the sequential agent-based simulation toolkit RePast system to the HLA framework. The middleware manages the execution of a federation composed of several REPAST instances that interact using the RTI [102].

The HLA_RePast system maps the RePasts events to the HLAs events, so the state transitions can be detected [64]. In order to perform such a mapping, mechanisms are required to identify time-advance requests. As depicted in the fig 2.16 [64], the incoming events are stored in the external event buffer, and the buffer is flushed when the time advance is granted. Thus, the federation is completely synchronized transparently, as shown in the figure 2.17 [64].

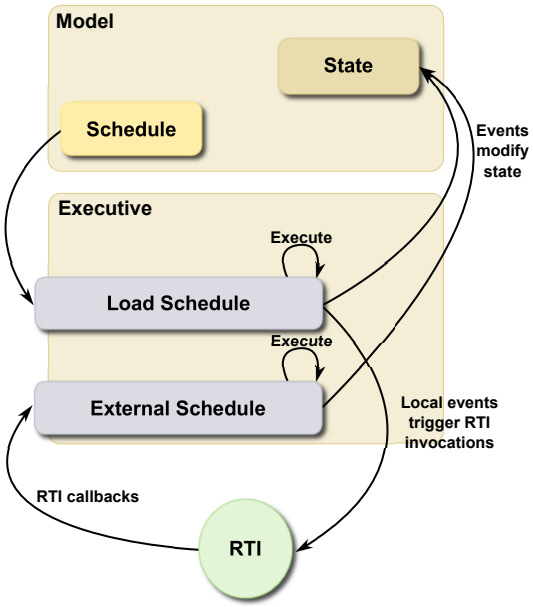


Figure 2.16: General architecture of HLA_RePast.

In HLA_RePast, local events are expressed as HLA events, notifying the middleware about state changes and converting from the Java events to HLA events [64]. Objects are used to make attributes public to entire simulation. Their variables, called attribute wrappers, are used to store the federates attributes and notify the RTI. Similarly to this approach, the HLA_AGENT [55] externalizes objects and agents by specifying them in the simulation FOM. The HLA_AGENT is a middleware that integrates the HLA framework to a toolkit, called SIM_AGENT, that supports agent-based architectures for

general purpose simulations.

2.3.9.3 HLA_GRID_REPAST

HLA_GRID_RePast was designed to run and manage large-scale distributed agent-based simulations on Grid environments [102] [87]. The proposed system acts as a middleware, integrating HLA_REPAST and HLA_GRID. Following the HLA_GRID's architecture, as shown in Figure 2.18 [102], the architecture of HLA_GRID_REPAST is basically composed of a client side and a proxy side. The client side contains the REPAST agent-based simulation code (the model), the HLA_REPAST, the Client RTI Ambassador and the Client Federate Ambassador Service. The proxy side contains the Proxy RTI Ambassador Service and the Proxy Federate Ambassador. Grid services are used to implement both Client and Proxy services, and web services are used to perform all the communication between RePast elements.

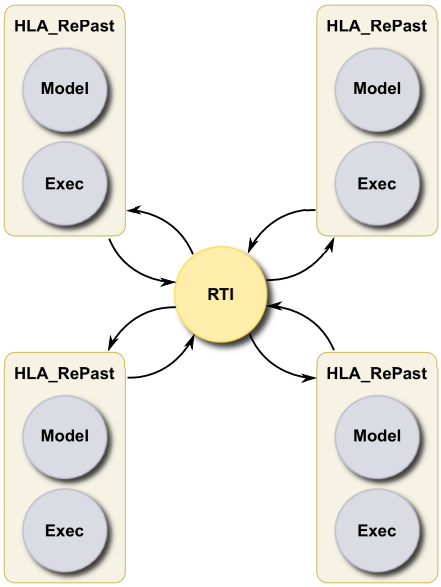


Figure 2.17: Management of HLA_RePast modules in a distributed simulation.

In a simulation, the RTI ambassador services are started, and the simulation federates access such services through the RTI proxy ambassador. Similarly, each federate initiates its federate ambassador service, registered with an RTI ambassador service, which is globally visible through the federate proxy ambassador. Moreover, the REPAST's

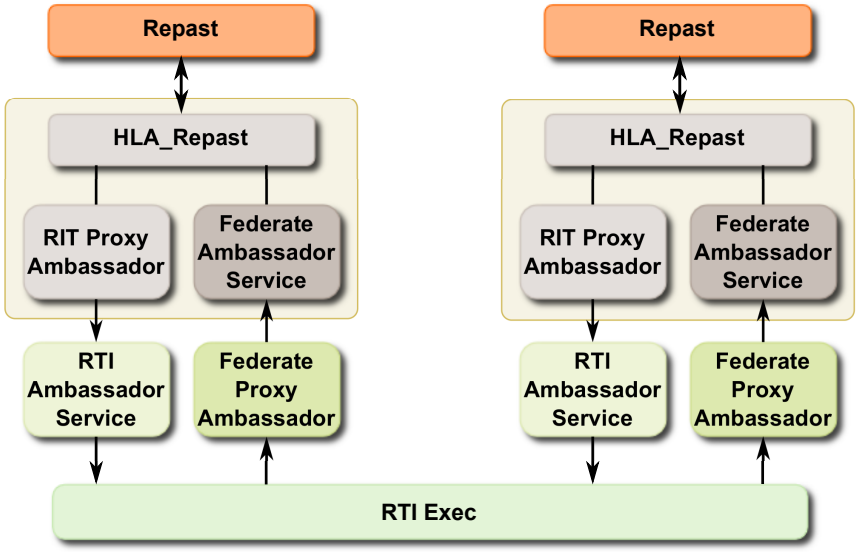


Figure 2.18: General Architecture of HLA_GRID_REPAST

events are monitored by HLA_REPAST, which makes RTI functions calls. Such calls are forwarded to the RTI proxy ambassador that translates to real calls to the RTI Ambassador Grid service. At the end, the RTI ambassador Grid service passes the call to the RTIExec. The RTIExec's communication is realized through the federate ambassador service, which forwards the call to the federate proxy ambassador. On the federate's side, the federate ambassador proxy passes the call to the HLA_REPAST, which converts it to REPAST events.

2.3.10 Service Oriented HLA RTI

Grid is employed to enable the communication between elements of large-scale HLA-based simulations. Service Oriented HLA RTI's (SOHR) framework implements the HLA RTI using Grid services. With a decoupled design of federates [30], the framework introduces modularity in the construction of a simulation by the dynamic, on-demand invocation of simulation services. The framework aims to increase the interoperability between simulation parts and improve flexibility in modelling simulations.

In order to qualify and distinguish the proposed architecture from the others, the authors introduce a classification for HLA-based simulations on the Grid [69]. According to

these classification, an approach can be Grid-facilitated, Grid-enabled, or Grid-oriented. These three categories define the degree of participation (or interference) of Grid services on HLA-based simulations for such solutions. In the Grid-facilitated approach, Grid services are employed to improve and facilitate the execution of HLA-based simulations while the real simulation communication is entirely performed through HLA RTI. In the Grid-enabled approach, Grid service interfaces are employed to externalize HLA-based simulations and to conduct simulations on the Grid. In the Grid-oriented approach, the HLA RTI is designed by using the Grid services and the communication is totally realized with these services invocations. The SOHR is classified as Grid-oriented since its main objective is to introduce a service oriented HLA RTI.

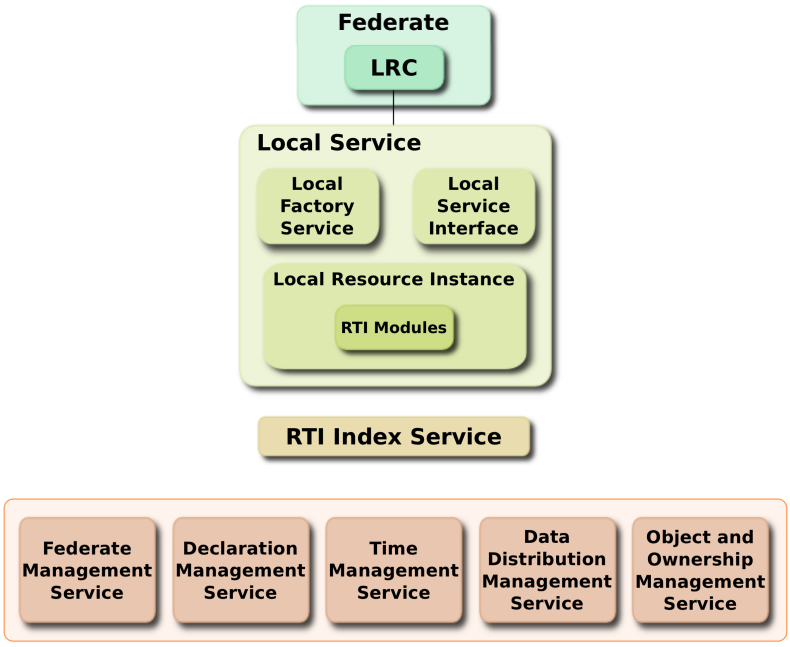


Figure 2.19: General Architectural Design of SOHR

As depicted in Figure 2.19, the HLA Local RTI Component (LRC) communicates with a Local Service (LS) in order to access other federates and the RTI. The LS acts as a broker for simulation federates. A LS comprehends a Local Factory Service (LFS), a Local Instance Service, and multiple Local Resource Instances (LRI). Basically, through a call to the LFS, a LRI is created for a federate. The federate communication with any other simulation elements through the LRI, which present 6 modules that reflect the RTI

services. The LSI contains a set of Grid service interfaces to access the RTI. In order for the federate communication through this components, its LRC translates its HLA calls to LIS Grid service invocations.

Remotely, the RTI Index Service acts as a registry of all available services, which represent each HLA RTI services. This element provides a dynamic discovery of such services in order to allow dynamic, on-demand assembly of simulation components. Moreover, the index service is responsible for creating and destroying federates through its invocations. Upon the request of a federate, the Index Service identify the management service requested. In the design, there are five management services that can be requested by a federate, and they correspond to the six HLA RTI service groups. The management services consist of Federate Management Service (FMS), Declaration Management Service (DMS), Object and Ownership Management Service (OOMS), Time Management Service (TMS), and Data Distribution Management Service (DDMS).

2.3.11 Summary

All the aforementioned architectures commonly define schemes that support large-scale HLA-based simulations by employing Grids services. Components are defined aiming at enabling the introduction of load balancing and fault-tolerance mechanisms for such simulations. Some architectures directly present elements with the objective of managing load and preventing failures. However, in such schemes, no load balancing mechanism or scheme is defined; just a rough description that delimits its existence, but no details that describe the inner functionality (definition of methods and techniques used to detect and react to imbalances).

2.4 Related Work

The related work is divided in two main streams: federate migration protocols and balancing systems for distributed simulations. Due to to the total dependency of balancing systems for distributed simulations on migration protocols, previous federate migration protocols are also described in this section. In order to enable the development of balancing schemes that efficiently improves the HLA-based simulations' performance, an

optimized federate migration protocol is design too. Observing and exploring the limitations of previous designs, the migration protocol and balancing schemes are created.

2.4.1 Balancing systems for Distributed Simulations

Many dynamic load balancing approaches have been proposed, aiming to solve uneven load partitioning problems for parallel discrete event simulations (PDES) distributed on shared resources and to therefore achieve certain improvements in performance. In the case of distributed simulations, the existing approaches assess the state of simulation applications or the characteristics of the distributed system where such applications run. These approaches attempt to identify load imbalance, perform load transfers, and decrease simulation execution time. Many aspects are considered in the design of these balancing approaches, such as monitoring metrics, resource heterogeneity, external background load, simulation computing load, simulation entities' interactions, and other simulation-specific characteristics, such as lookahead and virtual time progress. However, none of the solutions in the literature address the aspects that best suit HLA simulations running on large-scale distributed systems.

2.4.1.1 Balancing Schemes for Optimistic Simulations

Some schemes are designed based on the characteristics of optimistic parallel simulations [48, 41, 83]. In such balancing schemes, aspects specific to this type of simulations are used to identify load imbalances and define rearrangement of simulation entities, aiming to increase simulation execution time. The specific characteristics used in the balancing systems can be Global Virtual Time (GVT), Virtual Time Progress (VTP), or Least Virtual Time (LVT).

Glazer and Tropper [44] introduced the simulation advance rate as a metric to evaluate the imbalance of a optimistic distributed simulation. The rate is based on the simulation entities' time advance and their central processing unity (CPU) time consumption. Such measurement is also used to assign appropriate CPU time slice to each simulation entity when the balancing system is re-partitioning the load on the distributed system's resources. The proposed scheme consists of two phases: gathering CPU allocation and simulation advance time, and re-calculating a new load distribution. The load

is re-partitioned according to the advance time, so nodes with larger advance time are elected to receive more load, up to their capacity. Smoothed time slices are adopted in the balancing technique to avoid precipitated, abrupt resource re-allocation. Moreover, as pointed out by Jiang et al. [50], the Glazer and Tropper's scheme does no support redistribution of load for heterogeneous systems. Thus, Jiang et al. proposed an extension that incorporates weights based on constants to the redistribution formulas in order to consider heterogeneity aspects in calculation of the simulation advance rate. Such weights must be provided precisely to have a proper balancing. In both schemes, the external loads are not considered, and the metrics used to monitoring the system depends upon the simulation's code, which needs to produce the same load constantly for a correct load balancing.

Burdorf and Marti [23] developed a dynamic load balancing scheme for optimistic simulations based on the least virtual time (LVT) of each distributed resource. Starting from a static load partitioning, the dynamic balancing scheme, in a peer-to-peer fashion, collects the LVT of each resource through a transmitted vector. According to the values in the vector, a mean and a standard deviation are calculated. These two values are used to determine underloaded and overloaded resources, and the furthest behind simulation entity from the furthest behind resource is moved to the furthest ahead resource. The migration consists of moving the simulation entity, informing the other entities about the migration, storing all received messages during migration, and forwarding the messages in the GVT calculation. Therefore, the scheme does not consider heterogeneity issues and non-dedicated resources, its monitoring with sequential data gathering does not scale, its detection and re-distribution are based on simple, inaccurate comparisons, and its migration produces a large latency.

Schlagenhaft et. al. [82] devised a load balancing scheme based on a static partitioning and a dynamic load balancing mechanism. The static partitioning consists of grouping the simulation objects in simulation entities by identifying strongly connected regions in order to achieve minimal communication overhead. The dynamic balancing part employs the virtual time progress (VTP), which represents the average simulation speed in a resource, to determine load imbalances and minimize the number of simulation rollbacks. In order to determine migration moves, the VTP of a resource is compared with a predicted VTP (threshold), and the migration latency is considered to measure

the benefit of a migration since it inserts a large latency to the simulation. The migration is performed by a load mover that transfers a simulation cluster and informs other objects about the migration. The scheme does not detail about how migration moves are generated, it does not consider heterogeneous, non-dedicated resources, and its migration introduces a large latency.

Avril and Tropper [8] described a dynamic load balancing mechanism for clustered optimistic distributed simulations. The designed scheme aims at distributing evenly the load and consequently achieving a better resources' consumption. As balancing metrics, the communication dependencies and the simulation throughput are used to determine load moves. The throughput represents the number of non-rolled back messages processed by a simulation cluster. In iterations, the analysis of throughput in a resource allows to identify the overloaded and underloaded nodes and match possible migration pairs. Limited by a tolerance and according to the load to be moved, a set of clusters is elected for migration in the overloaded resource, and the resource with more communication with the destination resource, larger inter-processor communication (IPC), is chosen. Thus, the scheme performs load balancing based on load and considers communication, without detailing how migration is performed; the scheme also does not treat heterogeneity issues and external background load.

Carothers and Fujimoto [26] devised a dynamic re-allocation of non-dedicated resources for clustered optimistic distributed simulations. Such a scheme employs the processor advance time (PAT) as the monitoring metric and operates according to a processor allocation policy and a load balancing policy. PAT basically comprehends the amount of real time required to advance one unit of simulation time, deriving from the cluster advance time (CAT) and being collected periodically. The processor allocation policy determines a set of resources available for running simulations by comparing estimates of resources' CPU consumption with two thresholds. The load balancing policy identifies migration moves by comparing resources' PATs. A threshold is used to minimize the number of migrations, and an optimized migration technique is employed to move the cluster of simulation entities in a resource. This optimized migration procedure comprises decreasing the size of simulation entities' checkpointed states by rolling back the entire simulation to GVT. Thus, the scheme detects background load through CPU consumption and PAT, re-distributes the load through comparisons between PATs, and

moves load through clustered migration; however, heterogeneity aspects are not considered in the proposed scheme. An extended version of balancing approach is described in [27], which introduces TWFrac and Theta parameters to respectively insert awareness of external background load and heterogeneity of resources in the scheme. Besides being unclear about how resource load is measured and analyzed in the algorithm, this balancing approach shows limitations in determining load imbalances if heterogeneous simulations are managed.

Jiang et. al. [49], based on the Carothers and Fujimoto's [26] scheme, introduced a extended scheme in order to incorporate communication in the balancing system. As metrics for such an extended scheme, processor-to-processor communication (PPC) and cluster-to-processor communication (CPC) are gathered by a central component and used to determine migration. Based on a calculation using PPC and PAT, imbalances are determined through comparisons. CPC is integrated with CAT to select a simulation cluster, which has the highest factor, in order to perform migration. Moreover, a 4-layer framework is designed to incorporate Grid services in the devised system, so the services can be used to manage the distributed simulation entities accordingly. The proposed balancing scheme includes communication in a load balancing system previously proposed, and consequently the scheme inherits the same drawbacks regarding lack of support to heterogeneity of resources, large latency migration, and dependency on simulation code, which are added to the issue of merging communication and computation load in a single formula.

Deelman and Szymanski [33] devised a dynamic load balancing system for optimistic PDES that are specifically designed to solve spatial explicit problems. Extracting the advantages of these simulations' specific characteristics, the scheme balances a simulation, which is modelled as a ring where a simulation entity runs in a cell by moving simulation entity to its only two neighbours. The number of weighted unprocessed events of a simulation entity is employed as the load metric in the balancing system. This number is collected in every GVT calculation and is used to calculate the mean to be used to detect load imbalances. The imbalances are identified by determining the dominant load chains in the distribution and by comparing them with the mean. A dominant chain has its load re-distributed in its cells by migrating certain load from a cell to its neighbours cells. Backup information is kept in past cells for the occurrence possible rollbacks. As

a result, the proposed scheme, by performing a limited re-allocation of resources for a restrictive type of simulation applications, does not consider heterogeneity issues and external load in its design, presenting a limited simulation performance gain.

Choe and Tropper [31] described a flow control and a dynamic load balancing scheme to improve optimistic distributed simulations' performance. The flow control aims to minimize the differences of simulation entities' pace, and the dynamic load balancing attempts to re-allocate resources according to memory consumption and simulation speed. The flow control comprehends limiting the optimism of simulation by regulating the number messages allowed to be sent per simulation entity according to the obtained load metric. As load metric, the space-time product is calculated considering the memory consumption and the least LVT in a processor. Employing the same metric, the dynamic balancing scheme compares the system's mean plus two times its standard deviation with a resource load in order to determine migration moves. The migration consists of moving simulation entities from an overloaded resource to an underloaded one. Therefore, the scheme re-distributes the load by limiting the simulation optimism through a space-time product metric, but it does not detail how the simulation entities are moved and does not consider heterogeneous, non-dedicated resources in its scheme.

Low [57] proposed a dynamic load management scheme with a cost model based on the communication rate, the computation load, and the lookahead of simulations. The cost model involves calculating the GVT rate according to computation load, the communication, and lookahead. The scheme evaluates the load balance by re-calculating and summing the costs every simulation's superstep and by comparing the value with a threshold. Following a priority balancing order, computation load is re-distributed, simulation entities are re-organized to minimize communication, and simulation entities are moved to minimize lookahead differences. In the scheme, the load transfer mechanism is not mentioned, and the metrics for communication and computation are not delineated. In the devised scheme, neither heterogeneous resources nor external load are considered.

Wilson and Shen [91] developed a two-level, general-purpose, dynamic load balancing technique which adapts to load changes of discrete-event simulations. Adopting a two level scheme, such technique aims to distributed simulation load equally among available resources and minimize the communication between the simulation elements. In the first level, all processes are stopped in order for the load balancing system to collect CPU

consumption and pending load of a simulation. Then, based on this monitoring information, the need of load migration is identified, overloaded and underloaded resources are selected, migration moves are created according to the pending load of the system and to the amount of load processed by a processor. In the second level, load balancing is determined and performed through load migration. The scheme uses different policies to manage the communication and the computation load, and weights are assigned to loads so they can be transferred according to their complexities. The scheme does not detail how migration is accomplished and does not deal with issues regarding heterogeneity of resources and external background load.

Peschlow et. al. [71] devised an dynamic load balancing mechanism for optimistic simulations, considering both computation and communication imbalances. For monitoring, metrics related to computational load and communication load are used. CPU time consumed by simulations, simulation advancements, and processed events are the metrics for computational load and are used to determine imbalances. The interaction rates of simulation entities are metrics for communication load and are used to identify network overloads. Starting from an initial static partitioning, the proposed scheme performs the load redistribution in alternated cycles of communication balancing and computation balancing. Overloaded nodes are identified through the comparison between a node's capacity and its load, i. e., the time advance is compared with time of CPU utilization, similarly to Glazer and Tropper's advance rate [44]. Moreover, communication overloads are determined by analyzing the interaction rates of a simulation entity with remote entities. When re-partitioning the load, the balancing system transfers some load from a overloaded resource to a underloaded resource, without overloading the underloaded one. In order for the scheme to work properly, several parameters and thresholds are required to be provided, so precipitated load movements can be minimized. As a result, the proposed balancing scheme perceives the load changes of a resource, but as Glazer and Tropper's scheme [44], it is dependent on simulations' code.

A general purpose balancing scheme is jeopardized with the balancing designs totally based on optimistic simulations. Characteristics evidence by more generic metrics are needed to directly detect load imbalances instead of just determining the overload of resources according to the simulation execution pace.

2.4.1.2 Balancing Schemes for Conservative Simulations

Some other load balancing schemes developed for conservative simulations [41, 28, 65]. In such balancing schemes, lookahead is the simulation-specific metric used by some schemes to detect load imbalances. However, other schemes use metrics related to the current processing and communication load of shared resources in order to redistribution load dynamically.

Boukerche and Das [15] devised a balancing scheme for conservative simulations aiming to improve their performance as a consequence of evenly spreading the load and decreasing the number of simulation null messages. In their approach, the CPU-length for arriving messages is used as the load metric, and the overloaded and underloaded processors are identified through a comparison to a threshold. Their technique aims to minimize the number of null messages caused by load imbalances and attempts to decrease the communication overhead by grouping neighbour simulation entities in the same resource. Their scheme follows a two-level architecture to reduce the communication overhead caused by the balancing system. In order to perform the load changes, a simple three-phase migration algorithm is employed. In the scheme's design, either the heterogeneity aspects or the external load are not considered when re-distributing the load.

Xiao et al. [93] proposed a three-level scheduling algorithm based on the critical channel traversing to improve the event rate of conservative parallel simulations. The simulation entities with inter-dependencies are grouped in tasks, which are consequently stored in a centralized queue. A task scheduling algorithm is applied to select a task to run in a resource or in a set of resources, and this task is dictated by its input communication channels. The simulation entity scheduling occurs inside a task group and according to the task type by choosing simulation entities according to their communication dependencies (input channels). The event scheduling is conducted according to the conservative parallel simulation algorithm. The scheme is dependent on sparse connectivity channels and substantial lookahead to enable the scheduling and minimize the balancing overhead per simulation entity managed. Thus, the devised scheme balances the distributed load by minimizing communication and computational delays, but does not solve issues regarding heterogeneity and non-dedicated resources. Gan et. al. [42] propose a dynamic load balancing scheme for conservative distributed simulations

running on shared memory systems. Based on the an initial static partitioning, the scheme performs a scheduling of simulation entities to keep the simulation running at the same pace. The static partitioning is vital for the scheme's efficiency because it groups simulation objects with smaller lookaheads than a threshold in simulation entities to minimize communication latencies. Such a grouping and mapping mechanism is bounded by a cut-off parameter, which limits the lookahead. The dynamic balancing part manages a thread pool in which the simulation entities are stored in two queues to be run. A scheduling algorithm assigns priorities to simulation entities inversely proportional to their simulation times, so the differences among the simulation times are decreased. As a result, since the scheme is designed for shared memory systems, it does not treat heterogeneity, external background load, and scalability issues.

Boukerche and Tropper [19, 14, 16] devised a dynamic load partitioning system for conservative distributed simulations running on distributed environments. Such a scheme employs an heuristic technique to re-allocate resources for simulation entities. Based on thermodynamics, an annealing algorithm is used to determine a simulation's entropy, so migration moves are determine properly. According to estimated parameters, load and communication are considered in an iterative calculation for each processor. This calculation is used to compare with the distributed resources' mean to detect load imbalances and aims to maximize the calculated load value. Thresholds are employed in the scheme in order to minimize the search space for partitioning, to decrease the annealing algorithm's search iterations, and to determine when the simulation load is in equilibrium. Thus, the devised balancing scheme attempts to consider both load and communication in the same re-distribution calculation, whose efficiency is totally dependent on the application, but the scheme does not mention how monitoring and migration are performed, besides neglecting heterogeneity and non-dedication of resources.

Ajaltouni et. al. [4] proposed a load management scheme that monitors and balances computation and communications loads for peer-to-peer JXTA [2] simulations [20]. For monitoring, the system collects the CPU time consumed by each simulation entity and retrieves the communication of a simulation entity with other simulations entities, which is logged in a table. The re-distribution phase, influenced by priorities, alternates between cycles of communication balancing and computation balancing, and priorities are

[2] http://jxta.kenai.com/

used to coordinate the cycles alternation in order to adapt the balancing to the current simulation needs. The communication is balanced similarly to Peschlow et. al. [71], and the computation load is managed by comparing a node's load with the average load of the entire simulation. Additionally, the load of the simulation entity elected to be migrated is summed to the destination node load in order to check the creation of imbalances with load moves. In the scheme, the external background load and the heterogeneity of resources are not considered, and a simple, costly migration procedure is used, introducing global synchronization to simulations.

In these balancing designs, more generic metrics are employed, allowing the redistribution schemes to directly identify imbalances and react properly. However, such designs are limited since they do not fully consider the aspects of non-dedicated, shared resources on a large-scale environment, substantially compromising the detection and rearrangement of load.

2.4.1.3 Balancing Schemes for HLA-based Simulations

The balancing schemes developed for HLA-based simulations present architectures for integrating the schemes into the HLA framework, enabling transparent transferring of federates through migration.

Lüthi and Grossmann [60] devised a design called resource sharing system (RSS) to provide a dynamic load balancing system for HLA-based distributed simulations. The proposed system aims to produce minimal intervention in the HLA simulations, minimize the modifications into the HLA architecture, and avoid interference in the simulation applications. In the proposed scheme, only the federate migration mechanism is detailed, and the other load balancing elements are superficially mentioned in the architectural components. Basically, the system's architecture is composed of a RSS manager, which monitors the available computing resources and balances the load of HLA federates, and a communication federate, which is a modified federate that calls pre-defined HLA interfaces to perform federate migration and accesses a FTP server to transport data. Because HLA interface is used in the federate migration procedure, simulations are globally synchronized during migration, causing a large latency. In the proposed balancing scheme, only a high-latency federate migration mechanism is presented, and other load balancing aspects are not described.

Cai et. al. [25] developed a Load Management System (LMS) to support the execution of large-scale HLA-based distributed simulations. The proposed balancing system has a main component that organizes all the load management steps and works as an interface, transmitting the federate migration calls to the simulation federates. For monitoring, the load management system accesses the Grid Information Service (GIS) to obtain data regarding the distributed resources. Even thought monitoring data is collected, no re-partitioning mechanism is detailed, limiting the load management scheme only to the transfer of load from an overloaded resource to an underloaded resource. In order to perform federate migration, the system stops the entire simulation, transmits data through GridFTP, and accesses the Grid Resource Allocation Manager (GRAM) service to start the execution of jobs remotely. Therefore, despite the lack of a load re-distribution procedure, their scheme considers the background external load through GIS, but it does not solve heterogeneity issues.

Zajac et. al. [98] proposed a resource management system based on Grid services to support the execution of interactive HLA-based simulations. By accessing the Grid services and designing an structure to interface with HLA simulations' elements, the proposed system aims to provide means to enable dynamic simulation configuration. Even though the system's architecture is delineated by describing elements from a load balancing design, such as a monitoring component and a migration scheduler, only federate migration is detailed in the proposed scheme. The Grid services are used in the system to discovery simulation elements during run-time and to transfer information for federate migration. The federate migration procedure basically consists in calling the HLA interface to stop and start federates, which synchronizes the entire simulation, and accessing GridFTP to transfer data reliably. As a result, besides having a federate migration technique that introduces large delay for the simulation, monitoring and re-distribution mechanisms are not delimited in the scheme.

Tan and Lim [85] presented a load distribution technique for HLA simulations, focusing on optimizing federate migration. In their scheme, an interface component called Federate Wrapper is used to monitor, re-distributed the simulation load, and perform federate migration. Working periodically, the monitoring comprises collecting the amount of calls a federate emits to the Wrapper in order to advance in simulation time. In the scheme, the time advances receive weights to exclude federate's useless processing when

the advances are calculated. Additional information is collected in order to facilitate and improve the re-partitioning process although no re-distribution technique is described. A federate migration mechanism is used to accomplish the load changes, and it is basically composed of three queues to prevent simulation inconsistencies and minimize delays. Thus, the proposed scheme is dependent on the simulation code (time advance) and does not consider resource heterogeneity for load re-distribution.

Bononi et. al. [13, 10, 11] proposed a balancing system that for parallel and distributed simulations aiming at the analysis of communication load. Based on a middleware [12] developed for facilitating the management of HLA-based simulation communication, the balancing design identifies imbalances by detecting the ratio between incoming and outgoing simulation messages. Through out this ratio, the balancing system attempts to reduce the amount of communication between resources, which causes overhead in the simulation execution. The middleware used by the balancing scheme uses a different paradigm for migrating load. In this case, the HLA simulation objects are migrated instead of federates, which imposes a change in the method of implementing HLA-based simulations; simulation tasks need to be performed or are divided according to objects instead of federates. The design totally relies on the communication load to improve simulation performance, but as stated by the authors in [13], the scheme produces computational load imbalances in the environment. As another limitation, the use of a different paradigm for migration imposes different implementation and design methodology for distributed simulations.

Eve though these balancing schemes are able to be integrated into HLA-based simulations, they do not present an explicit balancing scheme. Details about the methods and mechanisms used in their systems are not delineated.

2.4.1.4 Summary

As described in Table 2.1, all the balancing systems for discrete-event simulations present some limitation that impedes them to be applied in large-scale distributed environments or in environments where resources are not dedicated and shared by other applications.

Table 2.1: Comparison of Balancing Schemes for Discrete-Event Simulations

	Sim.	Monitoring	Re-distribution	Migration	Heterogeneity	External Load
Glazer & Tropper	Opt	Time Advance	Comp.	-	-	-
Jiang et al.	Opt	Time Advance	Comp.	-	Weights	-
Burdorf & Marti	Opt	LVT (Vector)	Comp. Speed (StD)	Simplistic (Slow)	Partially (Indirectly)	Partially (Indirectly)
Schlagenhaft et. al.	Opt	VTP	Comp. pVTP + Mig.	Undefined	-	-
Avril & Tropper	Opt	Comm. Throughput	Load (Comm.)	Undefined	-	-
Carothers & Fujimoto	Opt	PAT, TWFrac	Load (Policies)	Clustered	Yes	Partially (Limited)
Jiang et al.	Opt	IPC	Comp. + Comm.	Clustered (Slow)	-	-
Deelman & Szymanski	Opt	Unproc. Events	Comp. (Chains)	Neighbour	-	-
Choe & Tropper	Opt	Space-Time Prod.	Comp.	Undefined	-	-
Low	Opt	* CPU load	Comm. Comp. Lookahead	-	-	-
Peschlow et al.	Opt	Time Advance	Comm. Comp.	-	-	-
Wilson & Shen	Opt	CPU Load	Policies (Comm/Comp)	-	-	-
Boukerche & Das	Con	CPU load	Comm. Comp.	-	-	-
Xiao et al.	Con	Comm. Dep.	Scheduling lvl	-	-	-
Gan et al.	Con	Sim. Time	Central (Priority)	-	-	-
Boukerche	Con	Entropy	Comp. + Comm.	-	-	-
Ajaltouni et al.	Con	CPU Load	Comm. Comp.	Global Sync.	-	-
Luthi & Grossmman	HLA	-	-	Global Sync.	-	-
Zajac et al.	HLA	Grid	-	Global Sync.	-	Only Monitoring
Cai et al.	HLA	Grid	-	Global Sync.	-	Only Monitoring
Tan & Lim	HLA	-	-	Queues	-	-

2.4.2 Federate Migration Techniques

Migration is ken in several areas, such as process migration and mobile agents migration in parallel and distributed computing. Amobile 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 [36]. 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, process migration involves the transfer of an executable code and data. Moreover, as proposed by Artsy and Finkel in [7], 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 processs 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.

Luthi and Grobmann [58] 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 federates start-up at the new location, as well as the saved states transfer. The communication federate transmits orders to the simulations federate in order to save and restore the state, and thus freezes the entire simulation.

Zajac et al. [98] 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 authors system and the federates code. The federate state is saved and restored through the HLA specification methods, and the transfer of the federates state is accomplished using the GridFTP.

Cai et al. [24] 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 federates 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 [85] 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 federates execution through SugarCubes [3], which stops and resumes a federate, and JavaGo2 [4], 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 aforementioned migration approaches is the requirement of third-party mechanisms. The use of an external tool to transfer data increases overhead for the migration process [97]. Peer-to-peer communication thus aims for migration.

Yuan et al. [97] 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 federates 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. [86] 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. [97] and Tam and Lim [85], unnecessary messages must be sent in order to manage each federate. This management involves

[3] http://www-sop.inria.fr/meije/rp/SugarCubes/

[4] www.ohloh.net/p/javago2

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. Part of this unnecessary management and computing is avoided in [56], which presented a two-phase migration mechanism for decoupled HLA federates. The migration of these decoupled federates requires simpler migration management since it does not fully deal with message losses and ordering.

Therefore, minimizing the delay generated by the migration procedure is the principal aim of such techniques. Migration delay dictates balancing responsiveness by restricting the number of load transfers that a balancing system can perform. Also, minimizing such delay provides better performance gains and allows the balancing scheme to redistribute the load more frequently. As summarized in Table 2.2, some migration techniques introduce global synchronization into simulations, propagating the migration delay to all simulation entities and causing a large overhead in the simulation. In this case, simulations that are balanced can be stopped regularly in order to perform load redistribution [91] and migrations. With such approaches, the entire simulation is suspended while a simulation entity is transferred [60], [25], [98], or an optimistic simulation is rolled back to the last GVT for migration [26]. Other migration techniques attempt to decrease the delay caused by migration, employing more complex mechanisms to make the migration process more transparent to simulations [85]. Consequently, transferring a simulation entity has a minimal effect on the rest of a simulation, yet it introduces additional processing overhead into the system.

2.4.3 Challenging Issues and Objectives

The aforementioned dynamic load balancing schemes are focused on sets of specific parameters. They present different approaches to solving the load redistribution problem for distributed simulations. The approaches provide certain performance gains, but they do not fully consider communication balancing, properly detect heterogeneity of resources and external background load, nor minimize migration delay. Computation and communication load balancing is vital for achieving distributed simulation performance improvement, and only the proposed redistribution systems described in [14, 71, 4] consider both computation and communication load in their balancing schemes. Addition-

Table 2.2: Comparison of Federate Migration Protocols for HLA-based Simulations

	Migration Overhead	Synchronization	Third-Party Tools	Transparency	Unnecessary Tasks
Luthi & Grobmann	High	Global Sync.	No Comm. Fed.	High	Yes
Zajac et al.	High	Global Sync.	GridFTP	High	Yes
Cai et al.	High	Global Sync.	GridFTP	High	Yes
Tan & Lim	Average	Freeze free	JavaGo SugarCubes	Low	Yes
Yuan et al.	Average	Freeze free	No Shadow Fed.	Low (SimKernel)	Yes
Tan et al.	Average	Freeze free	No (Peer-to-peer transfer)	Low	Yes
Zeng et al.	Low	Freeze free	GridFTP (No Influence)	Low (Decoupled Design)	No

ally, when communication balancing occurs, migration moves are performed in order to keep interactive simulation entities in the same resource, thus avoiding network overhead. Even though this load redistribution is successful in decreasing communication overhead, it does not deal with all cases of communication configuration, more specifically, with distributed simulations based on a centralized manager, such as HLA/RTI-based simulations. Moreover, some approaches consider the load produced only by the simulation that is balanced, and these approaches are therefore inadequate for non-dedicated, heterogeneous resources. Other approaches use the simulation entity's processing speed to monitor load imbalances, which indirectly and partially detects the existence of imbalances in these distributed resources. In order to fully overcome the problem of heterogeneity, benchmarks and normalization parameters need to be introduced into the balancing system. To detect background load, tools are necessary in order to measure the total load consumption of shared resources, and not just the load consumption from a determined simulation application. Load migration, which is a critical procedure for balancing simulation's load, is absent in several proposed solutions, and if migration is part of a solution, it introduces considerable latencies, decreasing the performance and balancing system responsiveness. Therefore, a new dynamic balancing scheme and a optimized federate migration technique should be proposed. This load balancing scheme's objective comprehends overcoming the previously listed issues and improving the HLA simulations' execution performance in large-scale distributed environments. The feder-

ate migration approach aims at minimizing the modifications on HLA architecture and providing a transparent method of transferring federate among resources.

Chapter 3

Optimized Federate Migration Protocol

Federate migration is an essential mechanism that enables the design and introduction of load balancing schemes for HLA-based simulations. The migration protocol allows the transfer of simulation between resources, and the efficiency in transferring simulation federates dictates the responsiveness of the respective balancing system. Because of the importance of federate migration, the optimization of its procedure and consequently the decrease of its latency substantially increases load balancing performance gain on simulation applications. As a result, in order to avoid the generated migration overhead, an improved, freeze-free federate migration technique is developed.

The designed federate migration technique is based on a simulation agent, which 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.

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 agents interface in order to

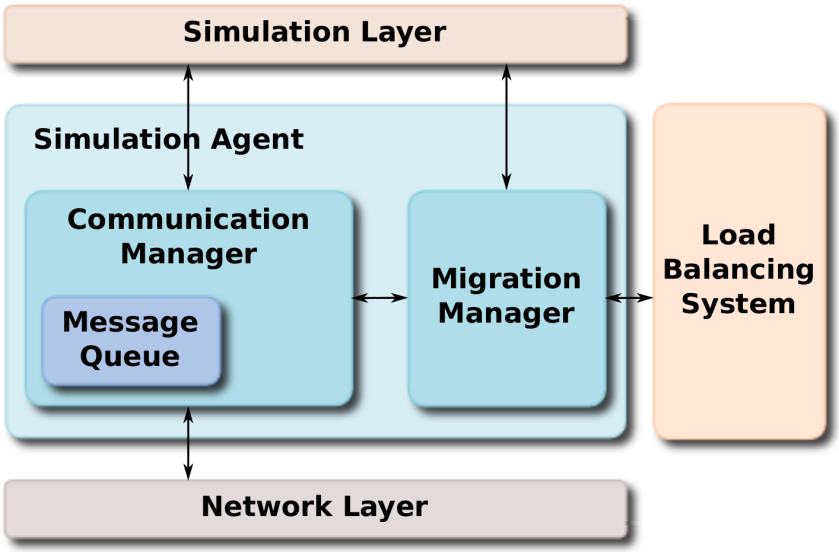


Figure 3.1: Architecture of simulation agent

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.

3.1 Simulation Agent Architecture

As explained in Figure 3.1, 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.

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 on the distributed systems nodes. Based on system analyses, it triggers migration and determines where it must migrate to.

The communication manager is responsible for maintaining the federates 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 federates 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 federates 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 federates state and the received messages. Finally, the migration manager acts at the remote location to restore the federates 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 [81], but this mechanism stops the federates 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 [85] [24] [97] [86], 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 [85] and Yuan et. al. [97], 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 federates 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 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. [97] save all the messages in a queue called inQ, which is sent together with the federates state. Similarly, Tan and Lim [85] use several queues to save and restore incoming messages. Tan et. al. [86] 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. [45]. 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 federates 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 proposed in the literature [97] [85] [86]. 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 federates 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 3.2, 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 federates 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 with the simulation elements about

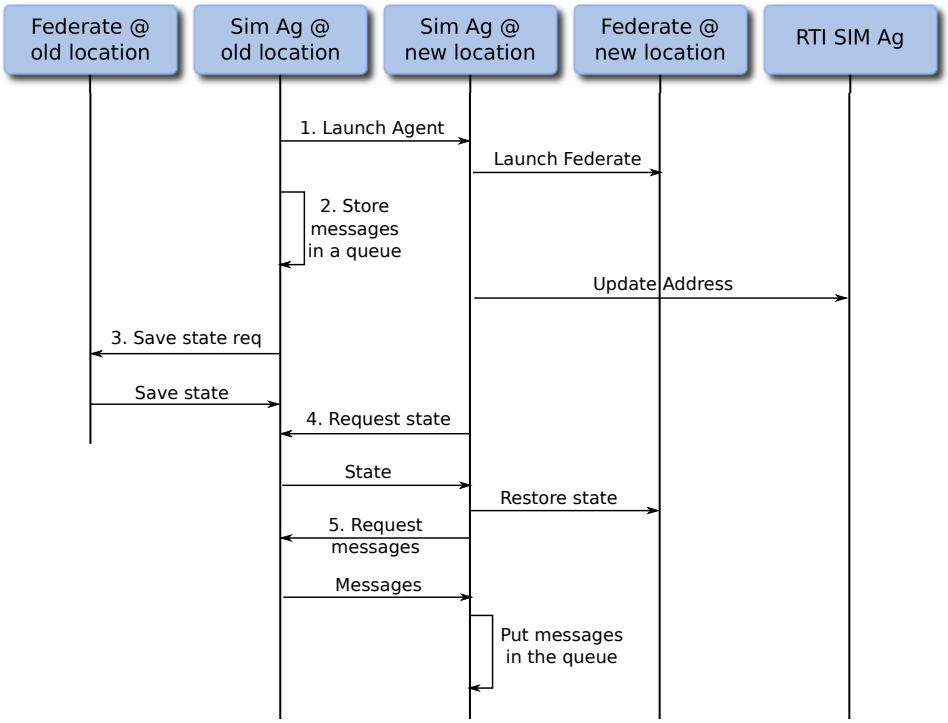


Figure 3.2: Federate migration protocol using the simulation agent

the address change. The agent also advertises the communication manager to store any messages in its queue, so that the messages 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 federates run and to retrieve the federates 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 managers request for the execution state.

In the fourth step, the migration manager at the remote location requests the federates state when the migrating 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.

3.5 Experimental Results and Discussion

Experiments have been conducted in order to evaluate the effectiveness of the proposed federate migration technique. The experiments have shown that the designed migration method produces low latency to simulation execution. This migration technique contributed to and enabled the design of balancing systems that can promote simulation performance improvement. Two major set of experiments have been conducted for the verification of the proposed balancing scheme. The analysis consisted in evaluating the performance of the federate migration proposed approach and proposed dynamic balancing scheme.

3.5.1 Experimental Environment and Scenario

For the experiments, all simulations were run in a cluster composed by 32 nodes. Each node was composed of a Intel Core 2 Duo Xeon processor CPU that runs at 3.4 GHz and 2 gigabytes of DIMM DDR memory RAM, and they were connected to each other through

a gigabit ethernet network. Also, each node was installed 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.

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 tanks 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 simulations 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% [90, 35].

3.5.2 Federate Migration Effectiveness Analysis

For this experimental analysis, just the proposed federate migration mechanism and scheme are assessed, so for that only the computing servers of the IBM cluster were employed since only the effectiveness of federate migrations is evaluated, which are influenced by data transfers between two resources and the scale of the simulations (number of federates and updating objects). 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 results represented a performance gain, decrease in latency, for federate migrations with the proposed migration technique, so when

used by balancing system in large-scale environments, this gain provides a more reactive response to imbalances, enabling more often load redistribution moves. 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 migration latency improvement for the experiments environment, as shown in the later experiments in this section, and they reveal more important implications for the balancing efficiency for larger scale environments.

Moreover, as our benchmark, an HLA simulation coded in C and using RTI 1.3 was used to conduct experiments and analyze the performance of our approach. Similarly to the scenario described in the previous section, the scenario for this analysis was a simulation of training operations of two teams of interactive tanks in a routing space. A tank performs random movements in two-dimensional space that is within the 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 which is related to the tanks new position. Thus, even though the scenario does not require a complex mechanism of data exchange, i.e., only transfers of information between federates for updating objects are used, the communication load is large. The number of update and reflection of objects messages grows proportionally to the simulation scale due to the number of federates and the simulations tanks; larger the number of objects requires more time for saving and restoring a federate's execution status, and more federates participate in the simulation more time is needed to notify the rest of the simulation about a federate migration.

In general, the federate migration procedure occurs when the simulation agent is triggered through a migration call at a given moment during the simulation. This migration call is issued by a balancing system that defined a redistribution of load and selected the federate to migrate to a remote resource. In the case of these experiments, the calls are not issued by the balancing system, but by explicitly sending the calls through a mechanism configured before the execution of the simulation. This mechanism consists in only issuing the migration call through the RTI library properly modified to support the proposed federate migration technique. 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 and deterministically have the same simulation execution state, the call

was triggered only once in the middle of each simulation run.

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 [85], 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.

In the first 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 3.3 shows the federate migration 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 federates 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 federates run is not yet restored increases with the number of federates. This pattern is clearly identified in Figure 3.3, 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 3.4 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 3.4 because the number of federates was the same throughout the entire experiment. Because the amount of information to save the objects 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 curves jitter is caused by the network oscillations, made noticeable by the tiny amount of data transferred. A more complex

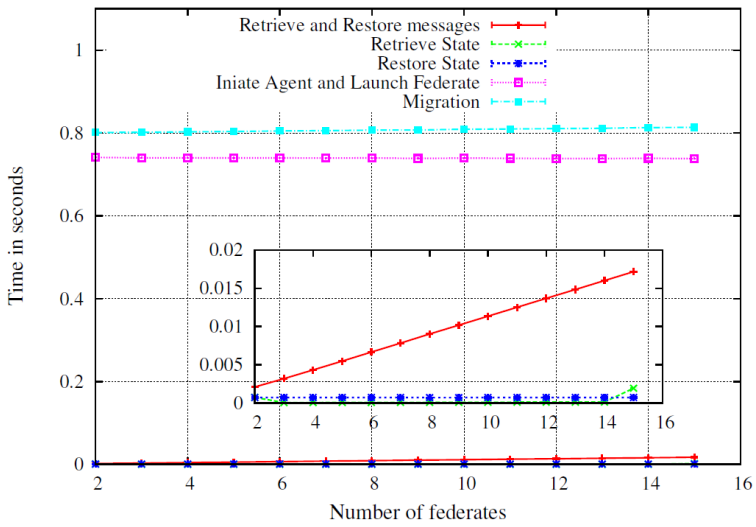


Figure 3.3: Federate Migration Evaluation for an Increasing Number of Federates

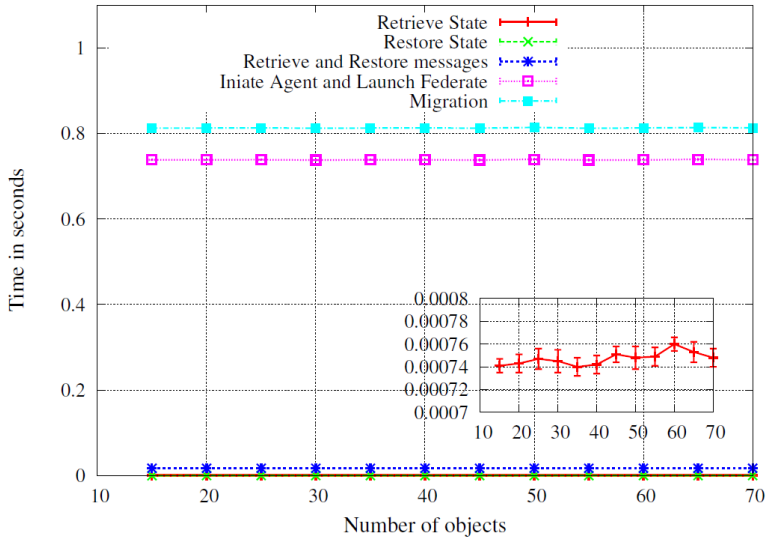


Figure 3.4: Federate Migration Evaluation for an Increasing Number of Objects

simulation scenario and a slower network could considerably increase latency at this step. As a result, all the curves presented in Figure 3.4 show an imperceptible change in the federate migration time generated by data transfer.

For comparison purposes, Figure 3.5 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,

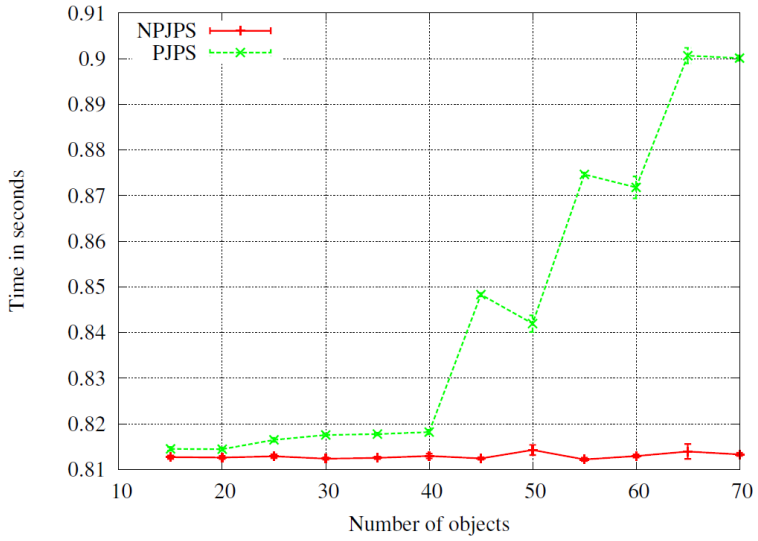


Figure 3.5: Performance comparison of federate migration designs

it is possible to observe how much time our approach saves by not calling PJPS methods.

It is noticeable in Figure 3.5 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 tanks move in our scenario, the tanks 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 precision 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 3.5 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.

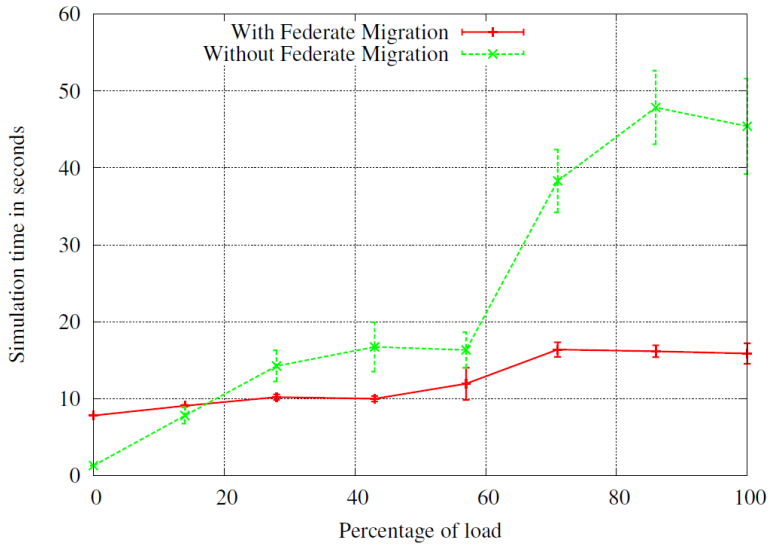


Figure 3.6: Comparison of PSPS and NPJPS federate migrations

This test case 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 clusters node to run together with the federates 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 3.6, 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.

3.6 Summary

An optimized federate migration technique for HLA simulation has been designed and implemented. The federate migration technique is fundamental for balancing the load of distributed systems and enables the development of dynamic balancing systems aiming at large-scale distributed simulations. The optimization of the migration procedure allows the decrease in migration latency and consequently increases simulation performance through reallocation of load. Thus, the main objective has been achieved by minimizing federate migration latency. The designed 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.

The results showed the relevance of federate migration for an HLA simulation. Also, and most importantly, the avoidance of 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. The next step in applying the developed federate migration technique is the design of a balancing system that can properly detect and react to load imbalances, triggering the migration procedure whenever its is needed to move simulation federates between resources.

Chapter 4

Dynamic, Centralized Load Balancing System

A balancing scheme is developed to provide a simulation-independent, dynamic load redistribution system for large-scale distributed HLA-based simulations. This scheme considers the heterogeneity of resources that compose the distributed system and detects the presence of external background load that runs on non-dedicated shared resources. In order to react to dynamic changes of communication and computation load, the proposed system is intended to constantly monitor resources and simulations, using a hierarchical architecture to decrease the monitoring overhead. Moreover, according to the aimed scheme, when imbalances are detected, the system defines a load redistribution according to partitioning policies and then re-configures load through a low-latency federate migration technique.

4.1 General Architecture of the Centralized Balancing System

For the intended load balancing system, an architecture is devised. As depicted in Figure 4.1, the architecture of the balancing system consists of components that periodically gather load information from the distributed resources through the Grid system; detect load discrepancies and trigger the redistribution algorithm; reallocate resources to correct the detected load imbalances; and migrate load to realize established load re-partitioning.

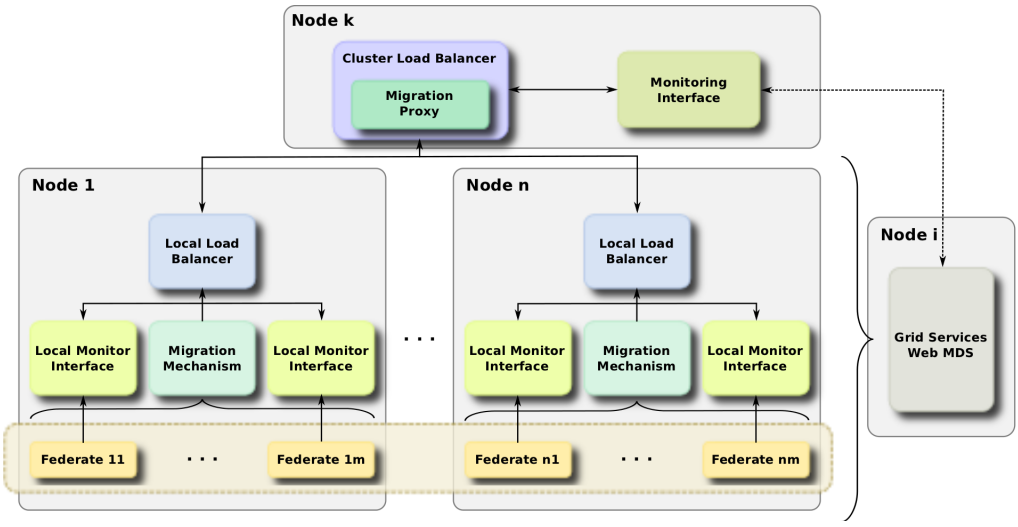


Figure 4.1: The Dynamic Load Balancing's General Architecture

Generally, the balancing system contains Cluster Load Balancers (CLB), Monitoring Interfaces, Local Load Balancers (LLB), Local Monitoring Interfaces, and Migration Mechanisms. Furthermore, in order to provide a cross-platform implementation and ease interaction with the adopted HLA RTI distribution, all the architecture is coded in Java.

4.1.1 Cluster Load Balancer

The Cluster Load Balancer (CLB) performs the main tasks of the dynamic load balancing system, executing the balancing algorithm and managing the load of many clusters of resources. As shown in Figure 4.1, a CLB can be place on a external resource (*Node K*) and communicates with the Grid system, which can be placed in other external resource (*Node i*). Both CLB and Grid system interface can also be placed on the same external resource or on an resource that contains simulation federations and an LLB. In order to organize all the required balancing actions, the CLB is composed of Monitoring, Redistribution, and Migration components. The Monitoring component accesses the Grid Index Service through the Monitoring Interface, which sends queries requesting information about the managed resources. The Monitoring Interface adds modularity and transparency to the balancing system because any monitoring tool can be easily accessed by the balancing system, without requiring knowledge about it. Moreover,

the Monitoring component is connected with LLBs to collect detailed information or with sub-CLBs to collect information from other clusters of resources. Finally, when the monitoring data is gathered, the Monitoring elements perform filtering and selection tasks and trigger the Redistribution component.

The Redistribution component re-configures the load distribution for the detected load imbalances and the required adjustments determined by the balancing policies. The re-partitioning of load involves the computation and communication aspects of a collected data sample. Consequently, resources are classified according to their load and federates are arranged according to their interaction rates. The structure of path distances is used for topology analysis in order to reallocate federates on the shared resources for decreasing communication latencies. After reallocation is performed and migration moves are generated, the Redistribution component emits migration calls to their respective LLB, which forwards the call to the proper Migration Mechanism.

The Path Distances is a data structure that stores information about the communication topology and is employed by the balancing system in order to search for destination resources and perform communication balancing. Based on an analysis of the communication topology, this data structure classifies the resources in distance rings. Resources that have the same distance are grouped in the same ring in the Path Distances. A search for a destination resource becomes possible when such a structure is employed. The distance corresponds to the sum of the number of hops between two resources, including the networking capacity of each hop. Simply put, the ring structure is created just once in the start up of the balancing system since the HLA-based distributed simulations in this scope are based on a centralized RTI.

In order to facilitated the management of large-scale distributed systems, a multi-layered hierarchical design is adopted for organizing the balancing system, as presented in the Figure 4.2. In the hierarchical architecture, a CLB can manage a list of LLBs and a list of sub-CLBs and can be managed by a upper CLB. A CLB controls a set of resources, which are managed directly by a list of LLBs. At the bottom layer of the hierarchy, a CLB controls only LLBs and, if a upper CLB exists, reports the collected data to its upper CLB when performing global balancing. At the intermediate layers, a CLB gathers information from a list of sub-CLBs so as undertake global balancing. At the top of the hierarchy, the root CLB is responsible for gathering all the collected data

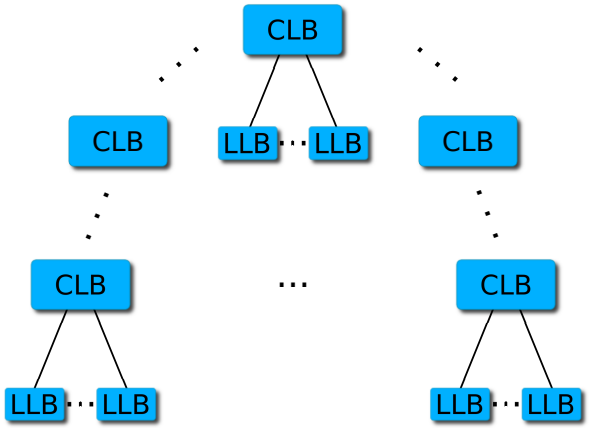


Figure 4.2: The Dynamic Load Balancing's Hierarchical Architecture

from its LLBs and sub-CLBs, performing the redistribution procedure, and reporting to other CLBs the load moves.

Therefore, the hierarchical structure of the balancing system minimizes the system's communication overhead and facilitates management of resources and simulations. Communication overhead is caused by the periodical data gathering needed from a large number of resources and federates. Filtering and selections performed by each CLB in the hierarchical structure decreases the amount of transferred information to be analyzed. This reduction of transferred information becomes vital for large-scale systems. The hierarchical structure also facilitates the issue of redistribution, or migration, calls to each federate, therefore avoiding the need for a centralized element reaching all simulation parts in order to execute the required load modifications.

4.1.2 Local Load Balancer

The Local Load Balancer (LLB) corresponds to an extension of CLB that acts locally in each resource managed by the load balancing system. The LLB is responsible for requesting load information from a federate, providing this information to the Monitoring component in the CLB, and executing the commands sent by a CLB. The load information requested from a federate consists in the CPU time used by a federate in a given interval, and this data is gathered through direct access to the Local Monitor Interface, which monitors each federate individually through a java library specific for monitoring the execution performance of java processes and threads. The library is the

standard Java monitoring and management library interface for the thread system of the Java virtual machine; it is called *ThreadMXBean* and belongs to a Java library. In order to collect a federate's CPU utilization time, the method *getProcessCpuTime()* from the Java library is called. Also, when a LLB receives a migration command, it informs the Migration Mechanism about the federate that is required to migrate and about the destination resource.

4.1.3 Migration Mechanism

The Migration Mechanism is responsible for managing the migration procedure according to the defined migration steps in the Migration Phase. The Simulation Agent is the main element of the Migration Mechanism. The agent supports all the migration tasks in order to keep simulation causality consistent; such migration tasks and the whole migration procedure are detailed in Section 4.4. Thus, the agent manages communication transference in both parts of a migrating simulation entity, so the migration process is imperceptible to the simulation entities. The agent is also responsible for managing all the required transfers of a federate's code, its initialization files, and its running status data. In order to perform such tasks, the agent consists of a Migration Manager and a Communication Manager. The Migration Manager orchestrates all the migration steps so that it does not loose or disrupt simulation events. The manager triggers the Communication Manager and requests a federate's simulation state by calling an interfacing method named *federateSave*. This method retrieves a federate's running status and LRC. The Migration Manager also launches the simulation manager at the remote resource designated by the LLB and manages the information exchange to restore the migrating federate completely. The Communication Manager organizes all the message exchanges of a federate during migration. The manager stores the incoming messages in a queue, which is transferred to the destination resource.

Furthermore, as depicted in Figure 4.1, the Migration Proxy is used as an intermediate element in the migration procedure to help transfer the execution state and the incoming messages of migrating federates. The proxy is employed when the destination resource is located outside the source resource's cluster. This means that the destination resource cannot be reached directly by the source resource. This element establishes connections with the two federate migration parts in order to receive, store, and forward the migration

data. The Proxy temporally stores all the transmitted peer-to-peer migration information locally and sends this information as soon as it is requested the destination migration peer. In order to be reached by balancing system's migration coordinators, the Migration Proxy is added to the CLB, which is placed in a resource that is visible by the rest of the balancing components in a domain (cluster), and runs concurrently as a daemon, waiting for inter-domain migration transfer calls.

4.2 Load Monitoring Phase

The monitoring phase is essential for detecting imbalances that occur unpredictably and dynamically during run time in a distributed system. Monitoring is the initial phase in a reactive or adaptive dynamic load balancing system since the results of environmental analyses indicate the need to reallocate resources and consequently trigger the other balancing phases. After environmental information is gathered, filtering techniques and selection mechanisms are used to retrieve analyze the data in order to identify load discrepancies.

4.2.1 Data Collection for Imbalance Detection Analysis

The monitoring phase involves two-level hierarchical data gathering to minimize the amount of collected information that is transmitted and processed. This two-level design is employed because data collection is limited by the trade-off between precision and overhead. The decrease in the amount of data that is collected allows for an increase in the gathering frequency and the amount of relevant data. The gathered information consists of the communication load of federates and of the computational load of resources and federates. Thus, hierarchical monitoring consists of a cluster data collection that focuses on resources, as well as a local data collection that focuses on the federates, and that is performed if required.

4.2.1.1 Cluster Data Collection

General information about all monitored resources is collected at the cluster monitoring level. The collected information is comprised only of the total resource load that

is consumed during a time interval. With load as the monitoring metric, a simulation-independent balancing scheme becomes possible, and external background load is detected. Such information is collected by accessing the Grid Index Service, which provides monitoring information from the Monitoring and Discovery Service (MDS) [1]. The MDS is a component of Globus Toolkit and collects information by requesting monitoring data from Ganglia [61] through the GlueCE schema [2].

The load information provided by Ganglia is comprised of the average number of ready processes in the queue of certain CPU. The Ganglia system periodically collects this coarse-grained CPU load information. The number of ready processes can be interpreted as a list of waiting processes that will be executed by a CPU; this list therefore represents the accumulated load of a given processor. However, in light of the differences of computing characteristics from one monitored resource to another, a capacity factor is used in order to normalize resource heterogeneity. Thus, this factor is calculated based on the computing capacity of each resource. In order to obtain such capacities, benchmarks determine the processing power of each managed resource. According to Formula 4.1, the capacity factor corresponds to the ratio between the common normalizing capacity ($nCap$) and the computing capacity (Cap_i) of each resource. Thus, the *relative load* ($rload_i$) of a resource is obtained by multiplying its current load ($load_i$) by the normalization factor. The common normalizing capacity is known by all balancing elements, and it includes the benchmark result of a previously determined resource.

$$rload_i = \frac{load_i * nCap}{Cap_i} \tag{4.1}$$

- $load_i$: current computational load value;
- $nCap$: common normalizing capacity;
- Cap_i: computing capacity.

4.2.1.2 Local Data Collection

In the balancing system, a more detailed data collection from resources determines federate migration properly when imbalances are detected in a distributed system. This

data collection is performed in order to retrieve information regarding the communication characteristics of each federate. This specific monitoring information is necessary in order to identify the migration moves that can benefit the system's load balance and prevent the creation of imbalances in the distributed system. The information is also essential for communication balancing because it describes the interaction patterns of a distributed simulation at any given moment. Thus, at the federate monitoring level, each federate is individually monitored, regardless of the external background load.

For load monitoring, information is collected from each resource with regard to the number of federates running on it and the load they produced at any given time. Federates that are running on a resource are ranked according to the load they produce. This resource consumption is detected through the CPU utilization time that a federate receives during a monitoring interval. The ranking of federates helps the load balancing system determine transfer policies and execute them.

For communication monitoring, a communication table is kept for each federate. The balancing system locally logs all federate communication by registering the destination address, the number of messages transmitted and received, and the size of the messages. In this case, because of the HLA RTI centralized simulation architecture, the destination is the same in all the log tables because all federates communicate only directly to the RTI. Thus, at every monitoring interval, the logged communication information is retrieved and erased from the table in order to prepare for the registration of new data in the next monitoring interval.

4.2.2 Filtering and Selection

Filtering and selection processes are applied to the gathered data to search for determinant aspects of decision factors. These decision factors show the need for load redistribution, which triggers proper balancing actions. The data-filtering process consists of coarse-grained and fine-grained procedures. The Coarse-grained filtering validates the gathered information that pertains only to the balanced resources. Monitoring data related to computing parameters not considered by the balancing scheme is discarded, with only CPU consumption retained. Also, data that is related to resources not managed by the balancing system is eliminated because it misleads the detection and the redistribution of load. The fine-grained filtering checks the resources that can perform migrations

for load re-partitioning. Data related to overloaded resources is excluded when such resources do not present any simulation entity running on them. After all irrelevant information is removed from the collected data, selection algorithms are applied so as to properly identify load and communication imbalances.

In the case of CPU load metric selection, the selection algorithm identifies load discrepancies through comparisons between the load of each resource and the overall CPU load average. The discrepancies in the data sample indicate overloaded and underloaded resources, which are potential candidates for load migrations. The average load used to determine such discrepancies is obtained by calculating an arithmetic mean. Moreover, thresholds are incorporated to the mean in order to classify the data sample in load categories and to determine the load imbalances more thoroughly.

The thresholds are required in the selection in order to identify when a resource is overloaded, balanced, or underloaded. Such thresholds depend on a specific policy, which determines the efficiency or the performance of the load balancing system. These thresholds delimit top and bottom boundaries, which establish a tolerance for load variances. Such a tolerance grows proportionally to the mean and is defined according to Function 4.2. The function delimits the tolerance (bds) through three factors: $mean$, β, and α. Because this proportional growth of tolerance can hide considerable load dissimilarities, the factor β is incorporated into the function. Ranging from 0 to 1, this second factor decelerates the tolerance growth, and according to Function 4.3, the minimum ($\max(wl_i)$) and maximum ($\min(wl_i)$) loads in the gathered data sample are used to identify the load oscillation in β. Through this function, the relative load oscillation (β) limits the tolerance variation, so overloaded and underloaded resources are selected more accurately. Moreover, because the load imbalances are defined according to a criteria specified by given policy, the limiting factor α is required to narrow the balance interval. This factor is inserted in Function 4.2 and contains a value that ranges from 0 to 1. The value represents the percentage of the average load that is considered balanced in a given load sample.

$$bds = mean * \alpha * \beta \tag{4.2}$$

- $mean$: overall computational load average.

$$\beta = \sqrt{\frac{\max(wl_i) - \min(wl_i)}{\max(wl_i)}} \tag{4.3}$$

- wl_i: weighed load of a resource.

In the case of the selection for communication load metrics, a similar approach to load selection is employed. All the federates' communication rates are compared with an average rate. These comparisons reveal the federates that interact the most within the balanced system. An arithmetic mean is computed in order to retrieve the average value. To more accurately identify the communicative federate candidates, thresholds are also employed in these comparisons. Because the communication rate is particularly application-dependent and varies according to each simulation implementation, the thresholds are determined through a method that is based only on the analysis of the collected data sample. The standard deviation of the gathered data sample is employed in the calculation of the superior boundary, which is used to identify communication imbalances caused by interactive simulation federates. Such federates present a differential communication rate, which indicates that the simulation performance might be limited by their communication latency. Thus, these federates are selected as candidates for migration in the next balancing phase.

4.3 Load Redistribution Phase

Due to its role in efficiently balancing a distributed load, the redistribution phase is the main part of a dynamic load balancing scheme. Balancing efficiency depends on the relationship between complexity and time; thus, simple redistribution algorithms are employed. These algorithms must generate a reasonable redeployment that reflects the real workload status of the distributed system. Moreover, computation and communication characteristics of the distributed environment and simulation are considered in order to determine the proper load modifications for minimizing imbalances. The redistribution algorithm assesses computation and communication aspects individually because they reflect different load characteristics. These are classified either resource-centred or federate-centred respectively. Thus, based on the classification performed in

the monitoring phase, the redistribution phase determines the migration moves that are needed for better load balance among the available shared resources and for decreased communication latency among the simulation federates.

4.3.1 Redistribution of Load Regarding Computation

As detailed in Algorithms 1 and 2, the redistribution of computational load re-defines the deployment of a distributed simulation by comparing the gathered load resources and by identifying simulation entities to be migrated. The comparison consists of analyzing the computational load of two resources, one of which is underloaded or overloaded. According to this analysis, a migration move is selected by transferring a simulation entity from an overloaded resource to an underloaded resource. A migration move consumes resources and introduces delays in simulations, and precipitated, unnecessary migration moves do not benefit simulation performance. These migrations are generated from the detection of abrupt, short-term load changes, and are avoided in order to minimize balancing system overhead.

Algorithm 1: Pair-Match Load Redistribution Algorithm

```
1  Require: resources_data, specific_resources_data
2  classify(resources_data)
3  order_ascendant(underloaded, balanced, overloaded)
4  foreach src_rsc IN overloaded do
5      dst_rsc ⇐ get_next(underloaded, balanced)
6      migration_moves ⇐ evaluate(src_rsc, dst_rsc)
7  end
8  foreach dst_rsc IN underloaded do
9      src_rsc ⇐ get_next(overloaded, balanced)
10     migration_moves ⇐ evaluate(src_rsc, dst_rsc)
11 end
12 Return: migration_moves
```

Based on the classification of the data sample in the balanced, underloaded, and overloaded categories, the redistribution algorithm reallocates resources. This reallocation occurs according to the following defined premises: to establish migration moves from the most loaded resource to the least loaded resource, to conduct only one migration move for each resource, and to migrate a federate based on its load consumption of the

resource and its communication rates in the simulation. In keeping with the first premise, the data sample is organized in a sorted list in ascendant order. The migration pairs are assembled by matching the extremities of this list. The match is performed while any of the sets of underloaded and overloaded categories contain resources to be analyzed. Thus, the search finishes when it is not possible to identify two resources that are different enough in their computational load to justify an effective load migration according to the adopted redistribution policy.

Algorithm 2: Pair-Match Evaluation Algorithm

```
1  Require: src_rsc, dst_rsc
2  if dst_rsc < min then
3      if number_fed(src_rsc) > 1 then
4          create_migration_move(src_rsc, dst_rsc)
5      end
6      else if number_fed(src_rsc) = 1 & src_rsc > (min * φ) then
7          create_migration_move(src_rsc, dst_rsc)
8      end
9  end
10 else if (dst_rsc − src_rsc) < (min * δ) then
11     if number_fed(src_rsc) > 1 then
12         create_migration_move(src_rsc, dst_rsc)
13     end
14     else if number_fed(src_rsc) = 1 & (dst_rsc − src_rsc) > (min * φ) then
15         create_migration_move(src_rsc, dst_rsc)
16     end
17 end
18 Return: migration_move
```

As described in Algorithm 2, each pair of resources is analyzed by comparing the load of each resource in order to match them for load transfer. The comparisons of the pair-match generates a migration move through the analysis conditions related to the amount of computational load and the number of running federates in a resource. The first analysis condition identifies a potential candidate for receiving a federate; the load of the destination resource is then compared with an established minimum load (min). After this comparison, a migration move is determined if the number of federates in the source resource is larger than 1. Similarly, if the source resource's number of federates is one and its computational load is larger than the threshold $min * \phi$, a migration move is also

determined. The threshold $min * \phi$ corresponds to load for one process in a CPU queue (min), smoothed by the parameter ϕ. The second analysis condition determines the difference of load between the source resource and the destination resource. In this load relation, if the difference is larger than the threshold $min * \delta$ and the source resource has more than one federate running, a migration move is created. The threshold corresponds to a value equal to less than the load produced by a process in a CPU queue. If the source has only one federate and the relation is under the threshold $min * \phi$, a migration move is created.

After the migration pair is identified, a federate in the source resource needs to be selected for the migration move. With regard to the defined premises, the migration of the federate must cause smooth changes in the distributed load configuration and not generate additional communication overhead in the balanced simulation. In order to fulfill such requirements, the balancing algorithm analyzes the migration destination resource and its communication latency in order to choose a federate for migration. If the resource's distance is smaller than the source resource's distance, the algorithm selects the most communicative federate for migration. On the other hand, if the resource's distance is larger than the source resource's distance, the algorithm selects the least communicative federate for migration. If both resources have the same communication latency – for instance, when performing local balancing – or if there is more than one federate with the same interactivity rate, the algorithm selects the federate with the least load consumption.

4.3.2 Redistribution of Load Regarding Communication

After an ordered list of communicative federates is selected, a re-partitioning is performed to search for the most appropriate destination resources for such federates. These federates are re-allocated by evaluating them according to the re-distribution procedure described in Algorithm 3. This re-partitioning of federates is performed with the objective of precisely determining migration moves to destination resources that can benefit simulation performance by decreasing the communication latencies; consequently, a classification of resources is realized according to their network topology to match with the communication rate of each federate candidate. This classification mainly employs the distances between resources and a specific destination, which is the RTI in this particular

application case. As a result, highly communicative federates are elected to be moved to resources that are located closer to the RTI, so the communication latencies are reduced, decreasing delays in simulations.

Following the ordered list of federates and according to the location of each federate, the redistribution algorithm searches for the proper destination resource for each communicative federate. A data structure is used by the algorithm to store all the positions of the shared resources. This structure facilitates the search for a resource that is close to the target destination of a determined federate. By observing this structure and looking just to the communication scope, the most appropriate resource to transfer a federate to is the one that the federate communicates with the most. This resource is the optimal destination for the transfer of this federate because the federate's communication latencies are eliminated or reduced considerably. This approach is extensively employed by the previous communication balancing solutions, which group together the same resource simulation entities that have intense communication among them, avoiding networking delays. However, this reduces the parallelism of distributed simulations. This approach is also considerably limited and disregards the possibility of overloading resources, evidencing the impossibility of applying it to simulations based on centralized HLA RTIs. Therefore, a different technique is used to enable the migration of federates to resources close to a RTI and not to the same resource where the RTI is running. In this case, the path distances structure is essential for determining resources nearby the RTI.

The Path Distances structure organizes all the shared resources in distance rings according the analysis the communication topology. Each ring contains a group of resources that have the same distance to the RTI. The distance is defined as the sum of the number of network hops between two resources, which are weighed by their network capacities. In each distance ring in the structure, the resources are organized according to their computation load, so the balancing scheme is able to identify a resource that does not harm the distribution of computation load, avoiding the generation of load imbalances. Since the network topology is characteristically static and the destination is the same for any simulation federate, the rings of the path distances structure are generated only once: in the initialization of the balancing scheme. However, the ordering of resources in each ring is determined by the computation load of resources in a given moment.

Regarding the communication aspects, only the nominal network capacity in each

Algorithm 3: Communication Redistribution Algorithm

```
1  Require: current_loads, spec_loads, federates_loads, path_distances
2  order(federates_loads)
3  cmean ⇐ calc_mean(federates_loads)
4  cstd ⇐ calc_STD(federates_loads, cmean)
5  comm_federates ⇐ find_comm(federates_loads, mean, std)
6  foreach federate IN comm_federates do
7      while !resource_found and path_distances(next) do
8          ring ⇐ get_closest_ring(path_distances, RTI)
9          resource ⇐ least_load_resource(ring)
10         if !overloaded(resource.load) then
11             migration_move.add(federate, resource)
12             resource_found ⇐ TRUE
13         end
14     end
15 end
16 Return: migration_moves
```

hop and number of hops are used in the Path Distances structure to define distances, considering that the hop with the smallest nominal transmission capacity is used to qualify a distance. Resources are classified according to this static network configuration to define the destination for communicative simulation entities. These metrics represent a near static state of the networking resources, but the addition of dynamic factors would increase the complexity of the balancing system. Although the use of delay variations to fit federates' communication needs to network capacities presents a more realistic solution, its feasibility can lead to considerable balancing delays. By observing the network capacities and attempting to fit federates according to the devices' available bandwidth promotes the problem to a higher level of complexity; an example of this is the solution of the bin packing problem, a NP-hard problem. For instance, many factors, such as communication distance, individual and global communication rate, communication flow, and others, need to be measured because they directly or indirectly influence delays. A deep search and analysis caused by this complexity increase results in higher balancing latencies that can lower the responsiveness for the redistribution mechanism. Therefore, in a matter of simplification that minimizes the balancing overhead, the delays are not considered in the detection and redistribution phases.

The resource with the least load and which is located at the distance ring closest to the RTI is selected as the destination resource candidate to receive the migrating federate. After the selection, the resource candidate needs to be analyzed in order not to become overloaded because of the federate migration. In this case, an analysis evaluates the load of the candidate by considering the average load of the rest of the shared resources. The resource candidate's computation load is compared with the overall computation load of the resources. The difference between the load values needs to obey the policies delimited for computation load distribution in order not to harm the computation load balance, as delimited in Algorithm 4. In the case that a resource candidate is not able to receive a federate for migration, the next candidate is searched in the distance rings; thus, the next ring in the structure is selected, consequently identifying the resource with the least load. The same comparisons are consecutively applied to determine a migration move for the communicative federate. The procedure of searching for a resource candidate in the distance rings continues while the destination resource is not found or the communication latency of a ring is not greater than the latency of the resource where the federate is running. The stop condition of the algorithm establishes that any communication improvement cannot be reached since the situation of the federate cannot be improved in a given load configuration of the distributed system.

4.4 Federate Migration Phase

Migration is the final step of the balancing cycle in the scheme, and after the redistribution defines the federate moves that are necessary to be realized, the migration calls are emitted to their respective Migration Managers. The migration procedure can introduce substantial overhead to simulation execution time; consequently, the responsiveness of the balancing system is directly related to the migration latency. The reduction of the migration latency to a minimal time enables the system to produce a better performance improvement. Basically, the migration procedure consists of suspending the execution of a federate in a resource and restoring it at the chosen destination; however, more complexities exist in this procedure because simulation inconsistencies might be caused by the migration due to the time and data coordination relevance for an HLA simulation. The federate migration procedure is required to transfer the initialization

Algorithm 4: Evaluation Algorithm

```
1  Require: src_rsc, dst_rsc
2  if dst_rsc < min then
3      if number_fed(src_rsc) > 1 then
4          create_migration_move(src_rsc, dst_rsc)
5      end
6      else if number_fed(src_rsc) = 1 & src_rsc > (min * φ) then
7          create_migration_move(src_rsc, dst_rsc)
8      end
9  end
10 else if (dst_rsc − src_rsc) < (min * δ) then
11     if number_fed(src_rsc) > 1 then
12         create_migration_move(src_rsc, dst_rsc)
13     end
14     else if number_fed(src_rsc) = 1 & (dst_rsc − src_rsc) > (min * φ) then
15         create_migration_move(src_rsc, dst_rsc)
16     end
17 end
18 Return: migration_move
```

files of a federate to the destination resource, suspend the federate's execution, retrieve its execution status and the incoming messages, transfer its state and messages to the destination, and restore the federate execution at the destination. The data transfers represent the largest latencies in all the migration procedure, mainly considering it for large-scale environments. Thus, in order to minimize the migration latency, a two-phase migration mechanism is adopted in the balancing system. This technique is similar to the techniques described in Chapter 3 and by Zengxiang et. al. [56]; also, it requires minimal external tools, avoids unnecessary communication and computation overhead, and does not introduce global synchronization in the simulation.

The first phase of the migration procedure is responsible for transmitting the static initialization files, which are required for starting up the migrating federate at the destination resource. After these files are transmitted, the Migration Manager can be configured properly to perform the next steps in the migration process. These files are transferred using a third-party tool, GridFTP [1]. The migrating federate and the respective Migration Manager are launched remotely through the Web Service Grid Resource Allocation and Management [1]. Access to third-party tools introduces substantial overhead in the

migration process, but overhead is not incorporated into the migration latency because the federate is not suspended while the Migration Manager is not completely initialized at the remote destination.

The second phase in the migration procedure corresponds basically to suspending a federate and restoring its execution at the destination resource. Initially, the Migration Manager triggers the exchange of the communication channels in the RTI, so the migrating federate starts receiving messages instead of the federate at the source resource. After the exchange is successfully accomplished, the Migration Manager sends a call for the suspension of the federate's execution. Then, the federate saves its execution state and the messages that it received and did not process. The Migration Manager sends both state data and messages to the remote resource. The execution state is comprised of the dynamic information (variables) that represents the current execution status of a federate and its Local Run-time Controller's state. When the state and the messages are received, the Migration Manager at the remote location passes them to the migrating federate. Finally, the federate restores its state and takes over its execution.

4.5 General Load Balancing Algorithm

The described resource monitoring, load redistribution, and federate migration phases are merged to compose the devised dynamic load balancing algorithm, as detailed in Algorithm 5. In every balancing cycle, the algorithm triggers the monitoring of resources in order to detect load imbalances. If any load re-partitioning is required, load redistribution is invoked and migration moves are performed. The balancing is initiated periodically, and a balancing cycle occurs in periodic Δ intervals. The interval is limited to the frequency in which the MDS service updates its monitoring database, every 20 seconds, adding minimal overhead to the distributed system and dictating the reactivity of the balancing scheme.

As delineated in Algorithm 5, the distributed load is balanced in local or global cycles. The balancing system performs computation and communication load redistribution in the global scope, considering that the communication balancing is accomplished first. Communication balancing is only realized globally because a global view of the entire system is required for determining communication rate differences. On the other hand,

Algorithm 5: Global Dynamic Load Balancing Algorithm

```
1  path_dist ⇐ analysis(resources_topology)
2  while TRUE do
3      loads ⇐ query_MDS() current_loads ⇐ filter_MDS_data(loads)
4      if balancing_globally then
5          spec_loads ⇐ request_LLBs(current_loads)
6          if root then
7              foreach sub_CLB IN list_CLB do
8                  ext_loads ⇐ request_loads()
9              end
10             merge(ext_loads, current_loads)
11             foreach sub_CLB IN list_CLB do
12                 ext_spec_loads ⇐ request_spec_loads()
13             end
14             spec_loads ⇐ merge(ext_spec_loads, spec_loads)
15             feds ⇐ communicative(spec_loads, path_dist)
16             if feds ≠ ∅ then
17                 mig_moves ⇐ redistribute_comm()
18             end
19             mean, bds ⇐ calculate_mean_bds(mng_loads)
20             over, under ⇐ select(mng_loads, mean, bds)
21             if over ≠ ∅ ∧ under ≠ ∅ then
22                 mig_moves ⇐ redistribute_comp()
23             end
24         end
25         else
26             foreach sub_CLB IN list_CLB do
27                 ext_loads ⇐ request_loads()
28             end
29             merge(ext_loads, current_loads)
30             send_upper(current_loads)
31             foreach sub_CLB IN list_CLB do
32                 ext_spec_loads ⇐ request_spec_loads()
33             end
34             spec_loads ⇐ merge(ext_spec_loads, spec_loads)
35             send_upper(spec_loads)
36             mig_moves ⇐ retrieve_mig_moves()
37         end
38     end
39     else
40         overload_cand ⇐ select_overloaded(current_loads)
41         spec_loads ⇐ request_LLBs(overload_cand)
42         mng_loads ⇐ filter(current_loads, spec_loads)
43         mean, bds ⇐ calculate_mean_bds(mng_loads)
44         over, under ⇐ select(mng_loads, mean, bds)
45         if over ≠ ∅ ∧ under ≠ ∅ then
46             mig_moves ⇐ redistribute_comp()
47         end
48     end
49     foreach move IN mig_moves do
50         if internal(move) then
51             send_migration_move(move)
52         end
53         else
54             forward_migration_move(move)
55         end
56     end
57     balancing_globally ⇐ choose_scope() wait( Δ )
58 end
```

computation balancing is performed both locally and globally. In both communication and computation balancing, local and cluster data collections are necessary for the monitoring phase. The cluster data gathering is triggered as soon as the load balancing cycle starts in order to obtain an overall overview of resources. After collection, the cluster data is filtered, and a gathering of specific data is performed with all resources or with a defined group of resources. This group is based on the median calculated with the cluster data during local balancing. After data is collected, selection is initialized. In the selection step of the monitoring phase, communication balancing identifies the communicative federates while computation balancing determines load imbalances within the shared resources.

The respective redistribution algorithm is triggered to determine the possible migration moves, which depend on the type of balancing that is performed. This occurs if any computation or communication imbalance is detected. When global balancing is realized, the root CLB gathers all migration moves and sends them to their respective destinations: to its LLBs or sub-CLBs. This forwarding occurs with each CLB until all migration moves reach the bottom of the hierarchy: an LLB. Moreover, only one move involving a resource is allowed per balancing cycle; thus, the balancing system reacts gradually to load changes by correcting load distribution and evaluating the effects of the redistribution. After the migration moves are produced, they are forwarded to their respective migration mechanisms, which are responsible for triggering and managing the migration procedure. A migration move is then triggered locally in a source resource. The migration process occurs independently from the load balancing cycle. In order to exclude the load and latencies caused by migrations, the balancing system ignores the data collected from the resources selected for migration in the previous balancing cycle. Consequently, enough time is provided to these resources to stabilize their processing of a new load reconfiguration.

4.6 Extension of the Communication Balancing Scheme

The proposed dynamic balancing scheme employs a non-complex analysis technique, acting like a greedy algorithm. Even though it is a simple approach, the used analysis

provides a fast solution that is close to an optimal simulation redistribution. With this approach, the scheme obtains a rearrangement of load in a short time. The quick balancing answer minimizes the consumption of resources, without introducing balancing overhead to the system that is managed; this technique also increases the responsiveness of the balancing system, enabling a more frequent detection and redistribution of simulation load. Furthermore, regarding only the observed communication aspects, the redistribution algorithm achieves an optimal solution, in which the network distances are minimized according to interaction rates of communicative federates. Nonetheless, considering computation factors in the redistribution algorithm, the balancing problem is reduced to a sub-optimal solution. Because these factors are considered to avoid computation overload of the shared resources, they interfere with the communication objectives and can impede a communicative federate's movement to a closer location.

Moreover, the communication rate is a factor that is attached to each specific simulation implementation and consequently is obtained through calculations regarding only the gathered data sample. Consequently, the analysis of this metric evidences certain communication particularities, which represent the considerable communication overhead. However, the analysis may not identify some particularities. In such an analysis, the calculation of a common overall high average of federate communication rates can hide the overhead caused by some communicative federates. As a result, the balancing scheme might determine that the system is balanced, but the distributed system does not reach reasonable performance improvement even if migrations are realized to reduce delays.

Therefore, the detection of imbalances and the redistribution of simulation load in the balancing scheme are based on a mechanism that does not fully identify communicative federates. The mechanism employs standard deviation for calculating the threshold used to detect federates with high communication rates. The standard deviation tends to follow the load behaviour of an analyzed data sample, so the use of standard deviation to identify communicative federates impedes the detection of some communication imbalances or hides them. Such behaviour is observed when the collected data sample is composed of extreme values. The extreme communication rates lead to a steep increase or decrease in the calculation of the mean, augmenting the standard deviation value. The increase of such a value, caused by the high variation of communication rate, makes

the detection and redistribution algorithms focus on the extreme cases and not identify other non-extreme high communication rates. Consequently, the use of just standard deviation to select and determine redistribution can hamper the dynamic balancing for such simulations.

Even though extreme values of communication rates are rare for some data samples, they can be produced frequently by distributed elements in certain applications, evidencing a common data pattern. For instance, as described by the experiments in the next section, the simulation scenario was composed of several federates that intensively published objects' updates while a few federates were subscribed to these updates and received an overwhelming number of messages. In this scenario case, message recipients presented the largest communication rates and became the bottleneck in the simulation. These communication rates led the standard deviation to assume a large interval, which excluded other communicative federates from the redistribution analysis, as shown in Figures 4.5 and 4.6. The figure describes the results obtained using the communication load redistribution approach described in the previous section. In such results, the number of migrations is constantly limited to three in every simulation case. These migrations explicitly depict how the detection of imbalances was misled by a few communicative federates, which received all the object updates during the simulations and kept an extremely high communication rate. Thus, the standard deviation cannot be applied to every case; other approaches are required to determine a general, better suited reallocation of distributed elements to improve the performance of a distributed simulation.

As mentioned previously, two parts of the balancing algorithm are influenced by these extreme communication rate values and need to be modified/extended: detection of imbalances and redistribution of federates. The detection of imbalances shows which parts of an application/simulation are causing imbalances and the indication of load changes that are essential to achieve a better execution performance. As a consequence, the redistribution determines the required modifications with the information that is provided by the detection phase. If the detection is not performed properly, the rearrangement of distributed elements is limited. A better approach to detect the imbalances is needed, so extreme cases are considered but do not influence the analysis of the rest of the data sample. For the extension, two approaches are proposed: one applies iterative filtering in

the detection/redistribution process and the other employs limiting conditions different from the standard deviation.

4.6.1 Communicative Federate Filtering

In this extension, a filtering of federates is performed to specify the analysis of gathered data samples. The communicative federates that were moved to a more favourable location continue to be monitored by the balancing system. Since such federates might present a dynamic communication behaviour, they need to be observed constantly; thus, the balancing scheme needs to analyze all the federates and all the resources in order to improve the distribution of simulation parts. However, the consideration of this global view in the analysis can mislead the identification of communication imbalances in certain particular cases, as stated previously. In order to avoid this situation, recursive filtering is applied on the data sample. This filtering increases the threshold's precision that determines the communicative federates.

The detection threshold's precision is influenced by the distances of nodes that are considered in the calculations. A further analysis that performs filtering and/or selection is employed to allow the detection of imbalances belonging to non-extreme samples. Currently in the balancing scheme, the distances of resources to certain specific locations are employed to determine redistribution of load. In the filtering, the distances are used to incrementally search the federates that cannot benefit from the balancing algorithm. Therefore, the recursive filtering determines which federates are relevant for the detection of imbalances and the reallocation of resources. This cumulative filtering is added to Algorithm 3 and is performed before the search for closer resource destinations starts.

In the detection algorithm (together with the load re-distribution), the distance of each federate's resource is used to remove a federate from the data sample. The federates that are considered overloaded through an initial communication load analysis are selected for a distance analysis. This distance analysis observes the proximity of such federates to their communication destination. Because these federates are already in the closest available topological position, no migration move can be performed to decrease their communication overhead. The presence of such federates can hamper the detection of other communicative federates, which can have their communication latencies decreased through federate migrations. Consequently, as an extension, if a communicative

federate is running on a resource that is located in the closest position to the destination, the federate is discarded from the analysis. With this procedure, other federates which have a moderate but relevant communication rate are considered in this extended detection process. Thus, more interactive federates can be identified in the detection phase.

The process of identifying the communicative federates that are closer to their communication destination is performed repetitively until the system reaches a defined condition. This process influences the analysis of the communication balance of the system, improving the reduction of communication overhead produced by the distributed elements. Generally, whenever the search algorithm identifies a communicative federate to be excluded from the list of federates, a recalculation of mean and its standard deviation is performed on the data sample. With this recalculation, new overloaded federates are identified and analyzed in order to refine the search or to produce migration moves to improve the simulation performance. However, this process needs to have a well defined condition to reach a final deployment of federates. Such a condition is determined by a threshold that is obtained by summing the average and the standard deviation calculated after the first federate filtering. Therefore, through this approach, the issues caused by the extreme communication loads are avoided; moreover, this extension of the algorithm produces a more extensive analysis than when employing just the standard deviation to classify federates as overloaded.

4.6.2 Extensive Federate Analysis

A second extension for the detection/re-distribution approach is designed to improve the re-distribution of federates by increasing the balancing convergence speed. In this extension, the algorithm checks all the federates by maintaining a condition which enforces certain limitations and which is thereby more comprehensive than the one stated previously. This condition is composed of a federate analysis and a resource analysis. The first one provides the number of federates that require migration to produce a better simulation performance. The second analysis determines the availability of resources to receive such federates. In this analysis, computation load analysis is employed as a determinant factor to conduct this extension. As a result, the balancing system redistributes federates while imbalances exist or migrations are still possible.

Observing the inability to detect communicative federates caused by the use of standard deviation in the analysis, the second extension excludes the utilization of such a value. The limiting condition is determined by the relation between the initial average and the median of the data sample. The incorporation of the median into the analysis provides a general description of the communication rate distribution in the collected data sample. This new analysis shows a general view of the amount of communicative federates present in a simulation. The analysis disregards the federate's communication rate, preventing miscalculations. Therefore, the limiting condition for the communication balancing analysis is denoted by the smallest amount between the two statistical values, $min(average, median)$. Through this approach, the issues caused by the extreme communication loads are totally avoided.

With this technique, communication balancing evaluates a more ample set of federates, which also includes those with a relatively moderate communication rate. Consequently, this extended algorithm produces a more extensive analysis than when employing just the standard deviation to classify federates as overloaded. The use of a low-value threshold induces a selection of a larger number of federate candidates to be checked as communicative. Generally, the threshold determines the analysis of at least half of the data sample in order to determine migration moves properly. Through this extensive evaluation, the detection step certifies that communicative federates in certain cases are not excluded from the redistribution procedure. As a consequence of a more extensive analysis, the approach introduces additional overhead to the balance processing. However, this additional overhead, which is larger than the previous extension approach and the original detection algorithm, is still noticeably less than the overhead caused by the evaluation of an entire distributed simulation. In this extension, the number of evaluations is limited to half of the simulation federates if resources are available. In a large-scale overloaded system, this processing reduction becomes crucial to achieve reasonable measuring and analysis efficiency.

Allying both federate and resources analysis parameters in the balancing scheme, this extension generalizes the procedure of detection and redistribution by re-allocating federates to available non-overloaded resources. Basically, the extension of the approach attempts to assign all the available resources to simulation federates in order to decrease the distance between them and their communication destination. These two parameters,

the number of communicative federates and the number of available resources, condition the balancing. The detection/redistribution algorithm finishes when one of these parameters reaches zero. In this case, either all the resources in a better position are used for redistributing the load, or all active federates are re-allocated to a closer position. Therefore, this extension makes the procedure of detection and redistribution more general through a cyclic reallocation of shared resources to simulation federates.

Like the previous communication redistribution algorithms, the list of federate candidates is analyzed sequentially, generating migration moves for the most communicative federates first. Communicative federates are assigned to a closer resource and eliminated from the list. This process is realized while closer resources are available. The algorithm accesses the Path Distance structure to retrieve information from the available resources and updates this structure when a migration assignment is determined. For every federate, the algorithm needs to perform a search in a sequential pattern in the structure's rings, always starting the search from the beginning of the Path Distance that contains the list of resources. In this particular case, the beginning of this structure is the ring closest to the RTI.

The resource search in this extension is formed in this manner because federates are organized by their communication rate and are definitely assigned to a destination resource after analyses of computation load are performed on resources in the rings. Since the computation load of resources is used in the decision-making part of the redistribution and is not employed to organize the list of federates, resources can be assigned to a lager number of communicative federates. Thus, with this approach, even if there is no considerable communicative federates in the data sample, federates are still assigned to resources. The technique attempts to move any federate that presents a higher communication rate in order to decrease the communication delays and to balance distributed simulation.

4.7 Experimental Results and Discussion

Initial experiments have been accomplished in order to evaluate the effectiveness of the proposed federate migration technique and the effectiveness of the preliminary design of the proposed dynamic load balancing scheme. The experiments have shown that

Table 4.1: Physical Environment's Configuration

	IBM Cluster	**Dell Cluster**
Nodes	32	24
CPU	C2D 3.4 GHz	Quad 2.40 GHz
Memory	2 gigabytes	8 gigabytes
Network	gigabyte Ethernet	2-gigabit Myrinet

the designed migration method produces low latency to simulation execution and the balancing system's ability to detect and react to load imbalances. Both works contributed to a simulation performance improvement, which directly exerted a decrease in simulation time. Four major set of experiments have been conducted for the verification of the proposed balancing scheme: only computational load balancing, communication load balancing, both computational and communication load balancing, and the extension for the communication balancing.

4.7.1 Experimental Environment and Scenario

A large-scale, heterogeneous, non-dedicated distributed environment has been used to support the experiments. As summarized in Table 4.1, this environment has been composed of a Dell cluster consisting of 24 nodes, an IBM cluster consisting of 32 nodes, and a fast-Ethernet link connecting them. In the Dell cluster, each node was comprised of a Quad-core 2.40 GHz Intel(R) Xeon(R) CPU and 8 gigabytes of DIMM DDR RAM memory. All nodes were interconnected through a Myrinet optical network that allowed data transmission up to 2 gigabits per second. In the IBM cluster, each node consisted of a Core 2 Duo 3.4 GHz Intel(R) Xeon(R) CPU and 2 gigabytes of DIMM DDR RAM, and a gigabyte Ethernet network connecting the cluster's nodes. Furthermore, the experiments have been supported by the HLA platform with an RTI version 1.3 and the Globus Toolkit 4.2.1, having Linux as the operating system in both clusters.

The experimental scenario comprised federates evenly allocated to shared computing servers and an HLA RTI executive placed on a dedicated server. The simulation scenario consisted in coordinating the movement of teams of tanks for training operations in a two-dimension routing space. Such teams were composed of interactive tanks that moved

and interacted according to a time-stepped simulation model. Each simulation consisted of 1 to 1000 federates; even though the simulations in the heaviest load scenario did not impose an apparent overload for each CPU (an average of around 6.33 federates per CPU), the synthetic load produced by each federate was able to use the CPU completely, raising up its consumption to 100%.

In the simulation, each federate calculated the movement of a tank (object), published the resulting tank's position to appropriated federates, and subscribed for a interest region related to the position of its tanks inside the routing space. The calculation of movement consisted in determining the next position for its tanks (objects). Because this calculation did not impose considerable load to the CPUs involved with the simulation, synthetic load [63, 62] was added into each federate, producing intensive computing processing, which generated enough load to totally consume a processor's computational resource for a whole simulation time step. The synthetic load was intended to create a static, intensive load for all federates that could identically replicate load scenarios for comparison purposes. Thus, load added in each federate comprised a simple recursive calculation of a given number expressed by Function 4.4. Due to the recursion, the function presents $O(2^n)$ complexity, causing high CPU consumption for the calculation of relatively large numbers. The number used in the experiments was 30, making the each time step consume a CPU's time for around 4.25 seconds on average.

$$f(x) = \begin{cases} 1 & : x = 1 \\ x \times (f(x-1) + f(x-1)) & : x > 1 \end{cases} \tag{4.4}$$

- x: sole input value in the synthetic load function: amount of load imposed to the processor.

For generating the communication load controlling the communication imbalances, selected federates published certain number of large special objects and other federates subscribed to these special objects' updates. The publication caused a high network consumption that slowed down the simulation with considerable communication overhead and consequently delays in simulations. While normal simulation objects required the transfer of 700 bytes for an update, such special objects applied a 54000-byte load data transfer between each pair of federates. In order to create highly intensive data transfers, one federate in the simulation was selected to subscribe to all special objects.

To introduce external background load in the distributes system, a process that completely consumed the processing time of three CPU cores was placed on one of the resources. This external process generated load through launching three threads that ran in parallel and executed the Function 4.4 with 30 as the input parameter.

For dynamic scenarios, every simulation federate individually and randomly changed its communication and computational load intensity each $2 \times \Delta t$ seconds, in which Δt is the balancing cycle period. With 25% of probability, a federate modified its computational load intensity by decreasing the parameter of the load function to 15 or by increasing it to 35; the federate also turned on or turned off the rate of updates of special objects with the same probability. Moreover, the external load was randomly moved among the distributed resources each $2 \times \Delta t$ seconds.

Moreover, in order to provide trustworthy results, each plotted point in the graphs represents the average of 33 runs. Thus, the confidence intervals were calculated using the z-distribution at a confidence level of 95% [90, 35].

4.7.2 Load Balancing Performance Analysis

As a common simulation scenario in these experiments, simulation federates were deployed on the 56 nodes. It must be taken into consideration that one node was totally dedicated to run the HLA RTI executive. The values presented in Table 4.2 correspond to the load balancing system's configuration parameters. The LLBs of the balancing system were placed in every cluster node except the node where the RTI was running; moreover, a CLB was placed in one node of each cluster. Additionally, in the experimental scenario, federates coordinated the training operations of two teams of interactive tanks in a two-dimensional routing space in time-stepped simulations. Such federates performed computing-intensive calculations to produce the tanks' movements, published the tanks' positions, and subscribed to interest regions. The HLA simulations were composed of a range of 1 to 1000 federates, running for 100 time steps. Each federate managed 1 to 500 tanks. The parameters of these simulations varied according to three test case groups and are summarized in Table 4.3.

In the experimental analyses, the proposed dynamic balancing scheme was compared with the base line. The base line is obtained by statically and evenly distributing the simulation federates on the available resources and by running the simulations without

Table 4.2: Dynamic Balancing System's Configuration

α	min	δ	ϕ
0.15	150	8	6

Table 4.3: General Parameters for the Experimental Simulations

	Feds	**Objs/Fed**	**TS**
Computation	1 – 1000	1	100
Communication	500	1 – 1000	100
Comp./Comm.	1 – 600	100	100

any balancing. The resources' heterogeneity generates computation imbalances and the network distances produces communication delays. In the case of the dynamic balancing scheme, the same deployment of federates is used, but the balancing system modifies the load distribution as needed. The dynamic modifications of distributed load can be observed in the improvement in performance, but they are better understood through the number of migrations that were performed by the balancing system. Thus, in the graphs, the right Y axis represents the execution time of the simulations while the left X axis shows the number for migrations. The combination of the information contained in the left Y axis, the Migration curves, and the Dynamic Balancing curves exposes the amount of redistribution effort for achieving a proper balancing.

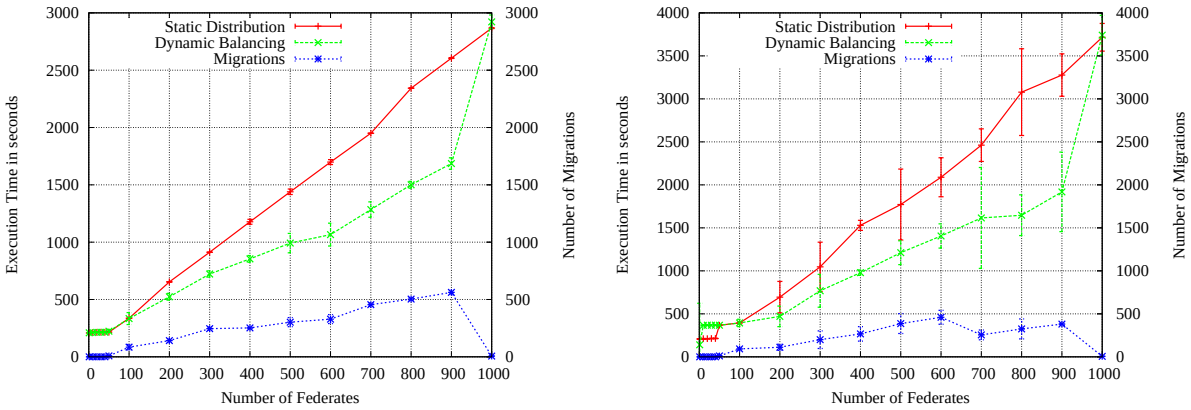
4.7.2.1 Evaluation of the Computation Load Balancing

In the first test case group, the effectiveness of the dynamic load balancing system in detecting and reacting to computational load was observed as distributed load imbalances occurred. Thus, the first analysis reveals the reactivity of the balancing system to the irregular deployment of computational load and dynamic load variations. The second test case group indicates external background load inserted into the system randomly and dynamically.

In this analysis, all the experimental simulations were distributed based on an even initial static partitioning. However, imbalances decreased the simulation's performance due to the heterogeneity of the shared resources and the dynamic computational load

changes. The simulations were composed of 1 to 1000 federates, which produced a continuous load or a load that changed periodically, resulting in negligible to intensive computing.

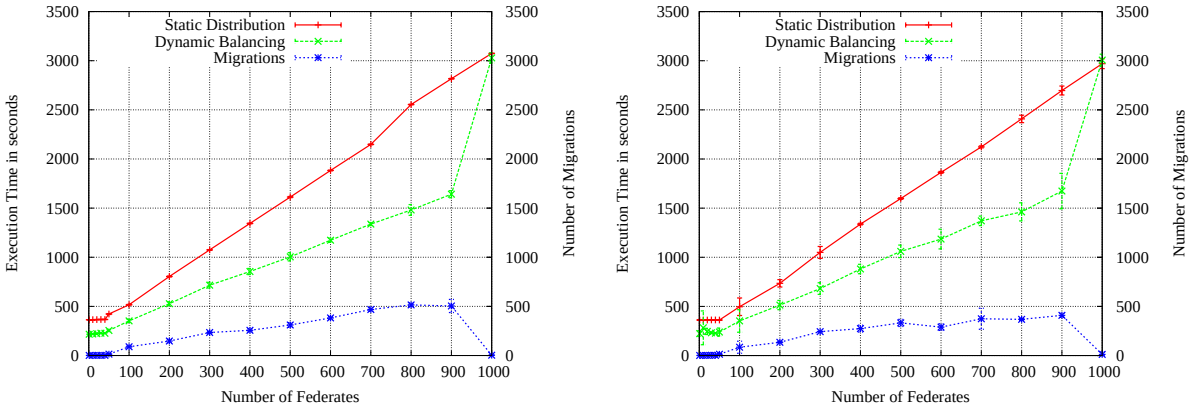
According to the graph in Figure 4.3a, the balancing scheme reacted to an uneven partitioning of load. The reaction to imbalances normalized the computational load over heterogeneous resources and reduced the simulation time. As shown by the non-existence of migration, the balancing scheme did not redistribute the load for experiments of less than 100 federates because the simulations did not require any load balancing. A noticeable improvement was produced for the experiments with more than 100 federates since the load generated was large enough to produce imbalances. When the experiments were composed of more than 900 federates, the distributed system reached a load saturation point. In this case, all the shared resources were overloaded, and no redistribution of load could improve the simulation's performance. This behaviour is also present in the experiments depicted in figures 4.3b, 4.4a, and 4.4b.



(a) Constant Load (b) Dynamic Load Changes

Figure 4.3: Performance Evaluation for an Increasing Number of Federates

When observing reactivity to dynamic load changes, as presented in Figure 4.3b, the redistribution scheme detected the load modifications during run time and performed the proper load redeployment. This caused an improvement in the simulation's performance similar to the improvement in the previous analysis. In this experiment, there is a larger variance of simulation times for both unbalanced and balanced curves. This variance is caused by the random, dynamic computational load changes produced by each federate



(a) Static Constant External Load

(b) Random and Dynamic Move of External Load

Figure 4.4: Detection of External Load for an Increasing Number of Federates

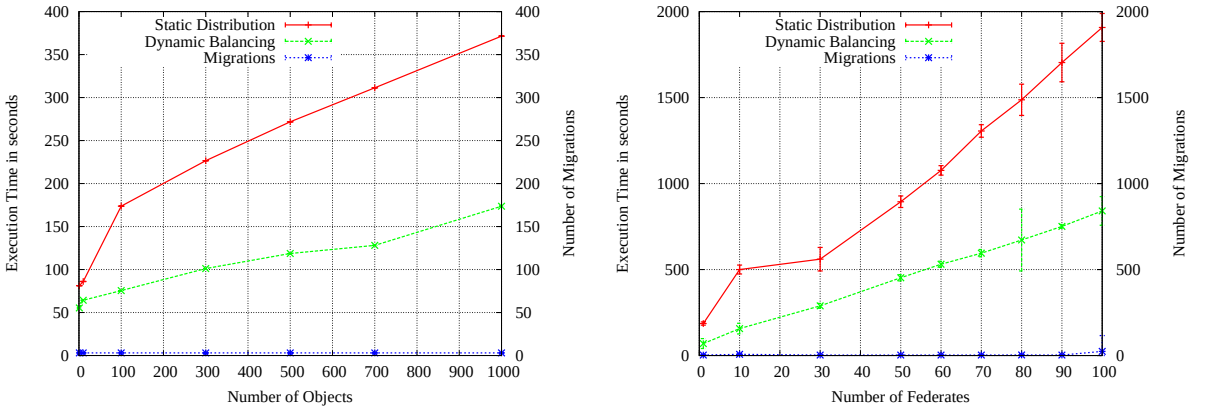
in the simulations.

In order to measure the capability of the load balancing system in detecting and reacting to an external background load, the previous experiments' scenario was extended by adding an external application in one resource. The application introduced a controlled amount of load to overload a resource. The external application was also dynamically and randomly placed on the shared resources.

As shown in Figure 4.4a, the introduction of an external load led to increased addition of execution time. This is noticeable in the experiments with 1 to 100 federates. When there were over 100 federates, the load produced by the federates in the system surpassed the external application's overhead, but it continued to produce an offset in the simulation times when compared with Figure 4.3a. The graph in Figure 4.4b describes the capacity of the balancing system to detect external load that appeared dynamically and periodically on the shared resources. The results indicate a variance in the simulation times due to the load oscillation caused by the dynamic movement of external load.

4.7.3 Evaluation of the Communication Load Balancing

The ability of the balancing scheme to react to communication load imbalances is evaluated in the second test case group. The analysis examines the reaction of the dynamic balancing system when redistributing the load of simulations containing specific federates with more communication or with an increasing amount of communication. The exper-



(a) Increasing Number of Objects (b) Increasing Number of Communicative Federates

Figure 4.5: Communication Balancing of Increasing Communication Load

iments observed the execution time of simulations composed by 500 federates. Among these federates, some published and subscribed to special objects with large amounts of data to produce the required intensive communication.

According to the graph in Figure 4.5a, the proposed balancing system identified communication load imbalances in HLA simulations. These simulations contained one federate with additional communication, an increasing the number of object updates. Such an experiment demonstrated the capacity to detect communication overhead caused by a highly interactive federate. This detection resulted in the constant number of federate migrations per simulation. Moreover, in a scenario with an increasing number of federates with 100 objects, as shown in Figure 4.5b, the balancing system successfully detected the communicative federates in the simulation. The balancing reduced the network latency of simulations with intensive communication and also decreased the simulation times. As shown by the curve, the improvement presents a linear trend resulting from the communication overhead of the RTI. Finally, the graph in Figure 4.6 shows successful re-partitioning according to dynamic communication load changes in the experimental simulations. Such simulations were composed of 100 federates that produced a random amount of interactions periodically.

In the three graphs, the dynamic load redistribution for minimizing of communication delay effectively reduced the simulation time of the experiments. However, as observed in the graphs, the balancing system employed a constant, small number of migrations to

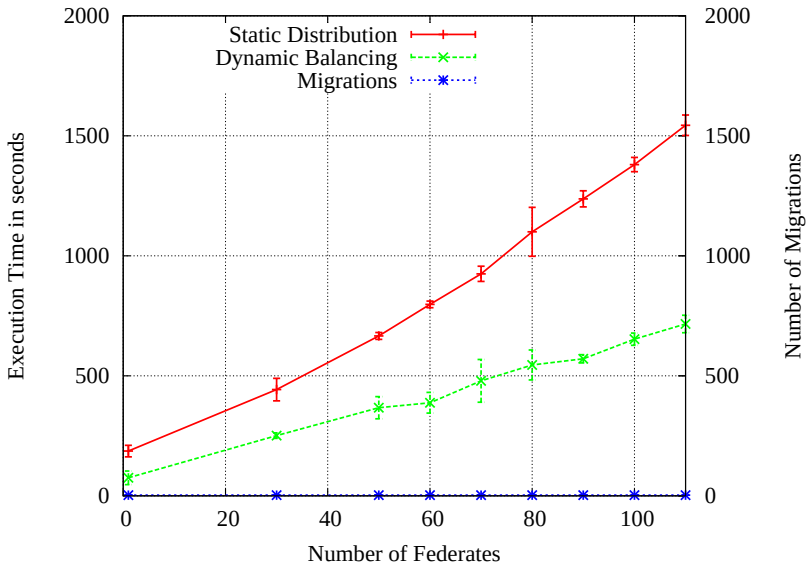


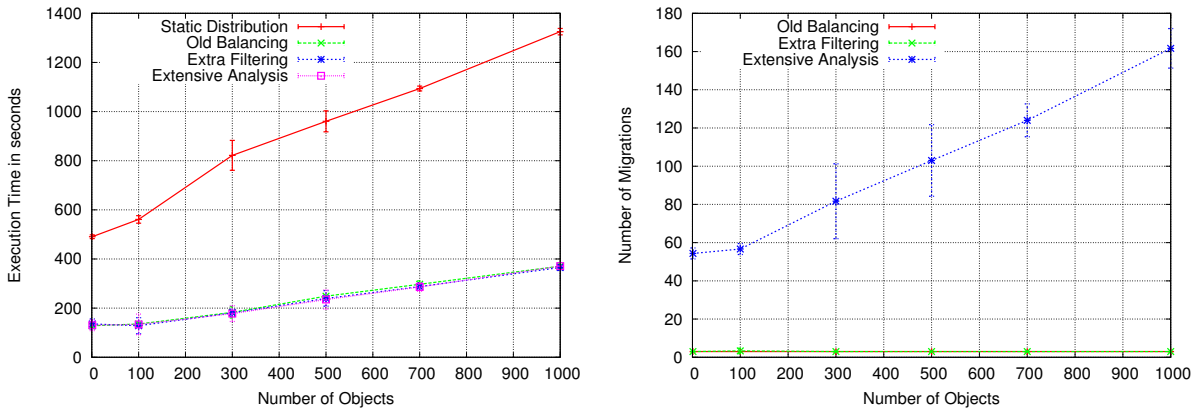
Figure 4.6: Communication Balancing with Dynamic Communication Load

perform such a redistribution of load, even for large-scale experiments. This characteristic results from the use of standard deviation to detect communicative federates. The detection of communication balancing does not identify all the communicative federates when some federates with extremely high communication rates are present in the simulations. This technique reacts properly to imbalances but is limited in specific simulation cases, such as the described communication experiments.

4.7.4 Evaluation of the Extension for the Communication Balancing

The first experiment measured the performance gain that the balancing system provided to a HLA virtual simulation. In this experiment, the dynamic balancing scheme, described previously in this Chapter and called *Old Balancing* in the graphs, was compared with the two aforementioned extensions. As depicted in the graph in Figure 4.7, an increasing communication overhead was applied in a small set of simulation federates: a federate that publishes special object updates, and three other federates that subscribe to them. The increase of communication load reflected in a proportional delay the simulations' execution time. A noticeable performance gain can be observed in the graph when comparing the base-line, which is not balanced, with the other balanced

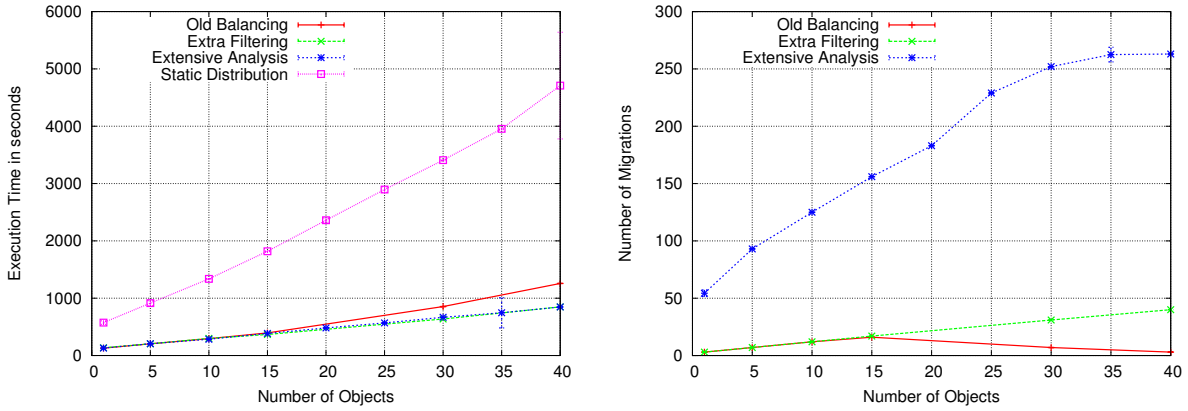
cases. When observing just the curves of the three balancing schemes, the old balancing system shows a slightly smaller performance gain in Figure 4.7a, but when comparing the number of migrations among the schemes, the extension that performs extensive federate analysis presents a high number of migrations. By crossing the information between the graphs in the figure, the results evidence that the extensions are not more efficient than the old balancing scheme for simulations with a few communicative federates, though performing more modifications in the simulation distribution.



(a) Performance Comparison (b) Comparison of Number of Migrations

Figure 4.7: Communication Balancing Gain for an Increasing Communication Load

In the second experiment, the responsiveness of all the balancing schemes is observed and compared when an increasing communication overhead is introduced to the overall distributed simulation. In the experiment, 100 special objects were assigned among 40 federates. The curves in the graph of Figure 4.8a show that all the schemes reduced the communication overhead considerably by detecting latencies and reacting properly to them. Nevertheless, the curve that corresponds to old balancing presented a performance decrease when compared with the curves of the extensions. When observing the number of migrations in Figure 4.8b, the performance difference originates from the decrease in the number of migrations that occurred simultaneously with the increase of execution time in Figure 4.8a. Such behaviour occurred due to the inability of the old balancing to react properly when extreme communication rates are present in the collected data sample. This limitation does not exist in both extensions because of their more detailed analysis and extra filtering.



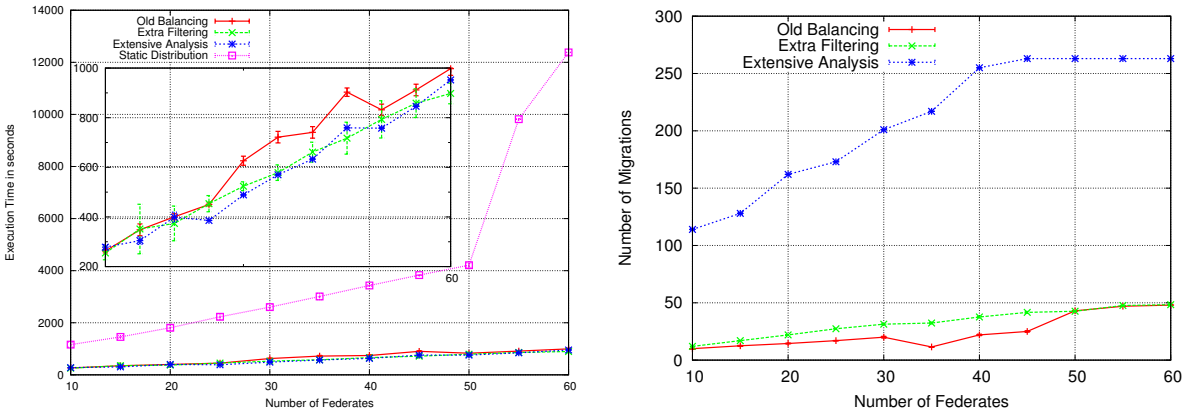
(a) Performance Comparison

(b) Comparison of Number of Migrations

Figure 4.8: Communication Balancing with an Increasing Number of Communicative Federates

Finally, the third experiment measured the capacity of the balancing systems to detect and react to dynamic changes of communication load. This experiment is similar to the previous one, but its communication load changed dynamically instead. The federates that were selected to present communication overhead changed their load randomly during the execution time of a simulation, producing a unpredictable behaviour. In this experiment, 1 to 60 federates in the simulations produced a random amount of object updates that ranged from 1 to 100. As shown in Figure 4.9a, the balancing schemes presented a large performance gain when compared to the base-line, and all their curves showed similar performance improvement; however, by observing the curves in more detail, the old balancing scheme shows a performance decrease between 30 and 50 federates when compared with only the extensions. Such behaviour is caused by the overall communication overhead in the system that was not detected by the old balancing scheme, which employs standard deviation to detect imbalances. The same behaviour is observed in Figure 4.9b when the curve of the old balancing scheme is compared with the curve of the balancing with extra resource filtering. This particular behaviour is caused by the communication load produced in the experiments. With load less than 30 federates, the communication rates were not as high as needed to mislead the old balancing. Between 30 and 50 federates, the communication load was enough to produce a few extreme communication rates, which influenced the old balancing but not the extensions. With a

load of more than 50 federates, the larger number of communicative federates (less load variations in the data sample) caused the old balancing to have a narrower standard deviation, inducing a better detection of communication imbalances. Thus, through this experiment, it was noticed that the extensions proved to be more effective than the old balancing scheme in some cases.



(a) Performance Comparison

(b) Comparison of Number of Migrations

Figure 4.9: Communication Balancing with Dynamic Communication Load

4.7.5 Evaluation of the Computation and Communication Balancing

In this test case group, the dynamic balancing of both computational and communication loads is observed in two scenarios. The test case meant to analyze the balancing system's efficiency when computation and communication imbalances were present in distributed environments and simulations. For both scenarios, federates that perform computation- and communication-intensive tasks are evenly placed on the system. Moreover, an external background process is inserted in determined resources. In the dynamic scenario, the external application was moved periodically and randomly on the shared resources, producing a considerable load, and the federates' computational and communication load changed periodically and independently.

In the first scenario, as described in Figure 4.10, the experiment shows that the proposed dynamic load balancing system successfully reorganizes an imbalanced load.

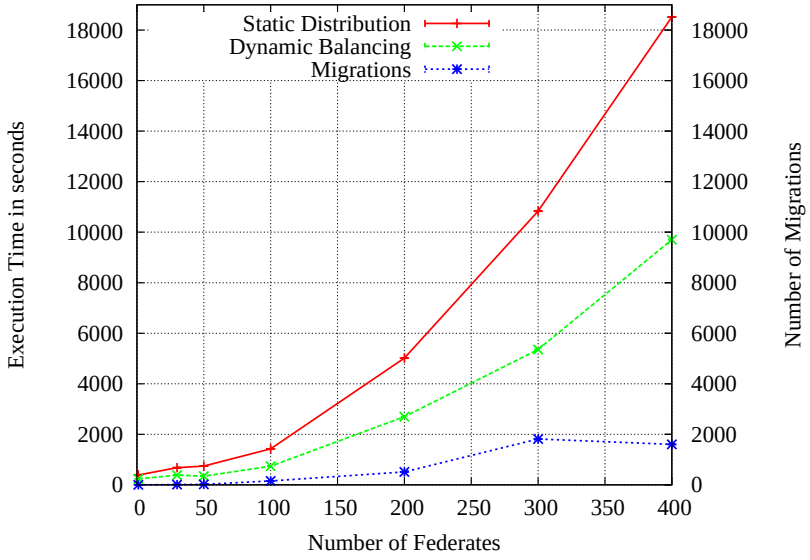


Figure 4.10: Balancing of Constant Communication and Computational Load

The decrease of execution time indicates that the balancing system detects the load differences produced by external applications, computational load, and communication latencies. However, experiments with over 400 federates show a decrease in performance gain due to the saturation of load in the distributed system, which is confirmed by a reduction in the number of migrations. In the second scenario, as depicted in Figure 4.11, the observed distributed system produces dynamic load changes, which are successfully detected by the balancing system. The dynamic variations can be identified in the graph by the increase in confidence intervals and by the decrease in execution times. As a result, the dynamic balancing system proved able to determine load imbalances and to redistribute the load accordingly, so better utilization of resources was reached.

4.8 Summary

In order to evenly distribute the load of large-scale HLA-based simulations on non-dedicated, heterogeneous environments when computational and communication imbalances are present, a hierarchical dynamic load balancing scheme is devised. To support such simulations, the balancing approach is composed of three sequential phases with a tree structure that detects imbalances, re-partitions the distributed load, and migrates

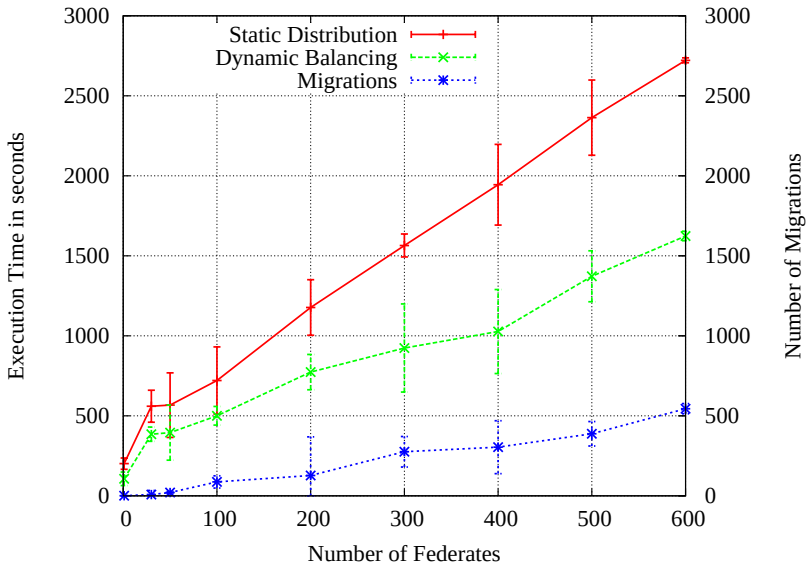


Figure 4.11: Balancing of Dynamic Communication and Computational Load

federates. The system also accesses Grid services to perform reliable load transfers, as well as monitoring that detects external background in shared resources. The hierarchical architecture is used to minimize the overhead produced by the balancing system, and benchmarks are employed in order to overcome the heterogeneity issues present in shared resources. As a result, this centralized design of the devised balancing scheme was able to decrease the execution time in the preliminary experimental simulations by improving the utilization of available resources.

The experiments proved the efficiency of the balancing scheme in detecting and reacting properly to dynamic computational load and communication rate changes. The devised balancing scheme could to detect and react to both balancing aspects. The extensions developed for the balancing of communication load also improved the balancing scheme functioning, enabling it to more effectively detect communication load changes and to produce a re-arrangement of federates that was able to decrease the simulation execution time. Even though the gains that the extensions provided were not substantially large when compared to the old balancing scheme, the extensions were more efficient in balancing the simulations in specific situations. These specific situations caused the detection step of the old balancing to fail. Moreover, a particular case was observed in the experiments when comparing results of both extensions; the extensive analysis showed

the same/similar performance gain in all experiments, but the number of migrations that it produced to balance simulations was considerably higher than other schemes. This particular situation happened due to the trade-off between migration latency and responsiveness.

Even though this dynamic redistribution scheme effectively balances the load, the devised scheme, as well as its extensions, is subject to drawbacks inherent to the centralization of its algorithms. The drawbacks consist in single point of failure, bottle necks, and delays arising from synchronization. Such negative aspects can impede the execution of the balancing system or decrease the balancing responsiveness through processing and communication overhead. This overhead is partially or completely avoided by introducing a distributed redistribution algorithm in the balancing scheme.

Chapter 5

Distributed Load Balancing System

The distributed balancing scheme essentially aims to decentralize the redistribution process of the scheme and provide balancing performance similar to the centralized approach. In the centralized scheme, a global view of distributed simulations and environments is employed to achieve the most suitable load changes. Instead of having a central component that realizes load redistribution from a global scope, each balancing component participates in the balancing scheme to rearrange the simulation load in a determined region or in a neighbourhood of regions. These components perform their part of load balancing independently from the others, so the synchronization existent among load management elements is avoided, minimizing delays and preventing failures. In the balancing scheme, the recent past of load status is employed as the predictive metric for detecting imbalances and determining re-distributions of the load. However, the collected data in the balancing decision-making mechanisms is not the most recent status information of a distributed system since the status load information of an entire distributed system information is not retrieved at the same time and from the same single location. In order to maintain this asynchronous redistribution processes, the schemes employs an architecture that is similar to the one described in. The architecture receives modifications for the inter-relations between the management components to rearrange the load independently. In the approach, with a decentralized redistribution procedure, data transfers exist only when rearrangement of the load is necessary, decreasing the flow of collected load data. Thus, the regional balancing is initiated in a certain region of resources and is propagated to the entire distributed system through inter-domain,

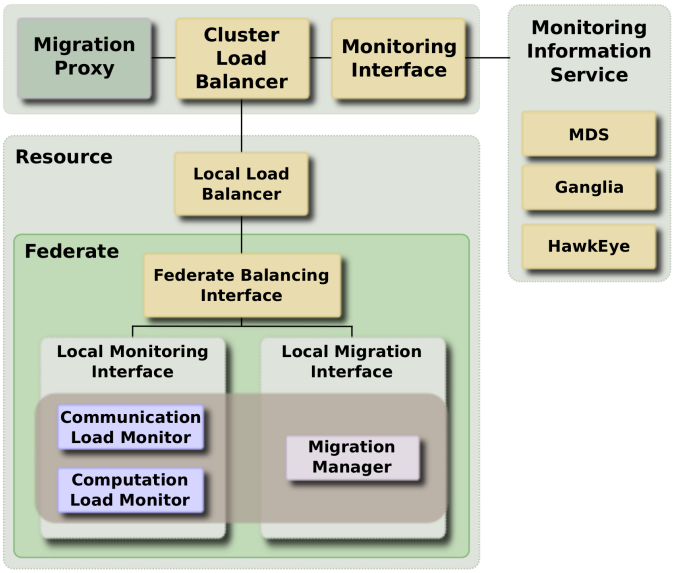


Figure 5.1: The Distributed Load Balancing's General Architecture

peer-to-peer load moves.

5.1 Architectural Components of the System

As depicted in Figure 5.1, the Cluster Load Balancer (CLB) is the main component in the load balancing system. The component manages federates and resources that are delimited in a domain in the distributed environment; the domain consists of a set of resources that are in a network region and presents common network capacities. In order to manage such elements, a CLB controls all the balancing steps: monitoring, redistribution, and migration. Since dynamic responsiveness is vital in the balancing system, the CLB constantly requests load information from the resources and federates, which triggers further data analysis and rearrangement of federates that generate migration calls to realize the modifications on the distributed load. The monitoring basically comprises resource-centred data and federate-centred data.

The resource data, which involves CPU consumption in each shared resource, is obtained through a Monitoring Interface. This interface communicates with a Monitoring Information Service that provides raw computational load data from certain resources. The service can be a Grids MDS, Ganglia, Hawkeye, and other similar interfaces, and

the interface is responsible for filtering and transforming the collected data to a standard input data format for the balancing system.

The federate load data is retrieved through a Local Load Balancer (LLB), which is a component that acts as an access interface to federates. A LLB also allows the balancing system to perform changes on the simulation distribution by forwarding migration calls. The federate load data comprises detailed monitoring information, such as CPU utilization of each federate and its communication behaviour. In each LLB, there is a Local Monitoring Interface (LMoI) and a Local Migration Interface (LMiI), which are responsible for accessing their respective local components that are placed together with a federate. The LMoI contains a Communication Load Monitor and a Computation Load Monitor while the LMiI presents a Migration Manager. The Computation Load Monitor obtains the CPU consumption of a federate by accessing a Java monitoring library interface, *ThreadMXBean*, every time it is triggered. The Communication Load Monitor, on the other hand, constantly logs the communication of a federate in a table, recording the amount of data transmitted and destination address. When triggered, the Communication Load Monitor aggregates this information in a sum and passes it to the LLB. The Migration Manager (MM) is the component that forwards and manages migration calls by accessing Grid services in order to emit federate migrations on remote resources.

To constantly obtain current data about resources load status, a resource management system becomes vital for providing load data to dynamically re-organize the load of distributed simulations. Foster [40] originally proposed Grid computing as a resource sharing system involving the coordination of resources, individuals, and institutions. Because of its services and characteristics that enable reliability, flexibility, and security, Grid is amply employed to manage distributed resources. Globus Toolkit [1] is the *de facto* implementation for a Grid computing middle-ware, and it is based on Open Grid Services Architecture (OGSA) [39], which uses stateful Web Services for providing and maintaining services of the Grid architecture. Grid services include the monitoring of resources and applications, the scheduling and allocation of resources, and the identification of application requirements [68]. Such Grid services are accessed by the balancing system in order to retrieve load information and perform reliable submission and transfer of federates to remote resources.

In a general view, the balancing algorithm is organized in three sequential phases:

monitoring, redistribution, and migration. In the monitoring phase, an intra-domain data collection is performed by a CLB and each LLB. After data is collected, a CLB filters it to keep only the relevant information for further monitoring steps, which consists in detecting computation and communication imbalances and selecting federate candidates for migration accordingly. The filtering comprehends removing non-manageable resources from the list and overloaded resources that do not present simulation federates. In the redistribution phase, analyses are performed to identify discrepant communicative federates that need to have their communication delay decreased and to determine overloaded resources that can have their load alleviated. These analyses are realized locally or remotely. The local redistribution transfers the simulation load between resources of a domain while the remote redistribution requires the interaction between CLBs to identify imbalance and generate a migration call.

The migration calls are all forwarded to their respective Migration Mechanism (MM). The MM controls all the required steps to perform a transparent, two-phase federate migration successfully and efficiently, and aims to minimize migration latency, as well as improve balancing responsiveness and simulation performance. The two-phase migration protocol used in the balancing system is described in Section 4.4.

The structure of the balancing system is organized hierarchically and according to the distribution of resources and network regions. A LLB works in each managed resource and is represented as a leaf in the hierarchy since it is the end-point in the balancing system. A CLB is responsible for a set of LLBs and delimits a domain in the distribute balancing structure. The CLB interacts with neighbour CLBs in order to redistribute load according to computation and communication aspects. Different from the balancing scheme described in Chapter 4, the hierarchical structure does not necessarily need to have a root CLB to centralize data gathering and the re-partitioning of load. According to the proposed scheme, the CLBs only transfer to each other relevant information for determining imbalances and generating load moves. Due to the scale of the balanced system and the possible existence of heterogeneous resources, benchmarks are required to properly identify imbalances of resources with different computing capacities through a normalization of the gathered data sample for the local or the inter-domain re-partitioning.

5.2 Distributed Load Balancing Scheme

The distributed balancing scheme is the merger of computation and communication balancing schemes. The joined redistribution algorithm attempts to associate the positive characteristics of both schemes without generating imbalances as collateral effects. The balancing of communication is performed by a CLB. In the communication redistribution algorithm, the analysis always involves two remotely connected resources (federates) and a third resource that receives a federate. The computation balancing is restricted to one resource (source) that might trigger load rearrangement to a second one (destination). Since the conditions for balancing communication load are more restrictive and include a wider scope that computation balancing, it is performed first in the redistribution algorithm.

As described in Algorithm 6, the balancing scheme is executed in cycles, which are determined by periodical Δ intervals. The interval introduce a trade-off between responsiveness and amount of balancing overhead introduced in the system, and between responsiveness and detection of simulation load oscillations. A balancing system with small cycle intervals may cause overhead from constant data collection and analysis, or it might not identify cyclic load oscillations that can generate unnecessary precipitated load rearrangements. In the scheme, Δ is delimited to 20 seconds, producing minimum overhead and introducing awareness of substantial load oscillations. The time value is the frequency in which the Monitoring Information Service adopted in the system (MDS service) updates its monitoring database.

Before the algorithm starts its execution, the balancing system is deployed and configured. The deployment is performed according to the distribution of shared resources and the network topology. Each resource designated to run federates receives a LLB, but resources that present the entire view of a region of resources receive a CLB. The connections between these CLBs are delimited according to their resources' proximity or hierarchical positioning. In this initialization, a CLB is configured with predefined parameters that determine the nominal communication capacity of the domain managed by the CLB.

The hierarchical structure allows a decrease in the amount of information transmitted to CLBs and permits the introduction of aggregation techniques. As stated in [73], these techniques minimize the quantity of information that is exchanged during the monitor-

Algorithm 6: Distributed Dynamic Load Balancing Algorithm

```
1  while TRUE do
                                                  /* Data is gathered, filtered, and selected */
2      loads ⇐ query_MDS()
3      current_loads ⇐ filter_MDS_data(loads)
4      current_loads ⇐ normalize_loads(current_loads, benchmark)
5      spec_loads ⇐ request_LLBs()
6      mng_loads ⇐ filter(current_loads, spec_loads)
7      spec_loads ⇐ request_LLBs()
8      bottom, top ⇐ calculate_thresholds(spec_loads)
                                              /* Communication re-distribution is performed */
9      high, low ⇐ select_communicative(spec_loads, bottom, top)
10     send_federate_candidates(high, low)
                                            /* Local computation re-distribution is realized */
11     mng_loads ⇐ filter(high, low)
12     mean, bds ⇐ calculate_mean_bds(mng_loads)
13     over, under ⇐ select(mng_loads, mean, bds)
14     mig_moves ⇐ redistribute_local(mng_loads)
15     send_migration_moves(mig_moves)
              /* Decision about inter-domain re-distribution and data request from neighbour CLBs */
16     if mig_moves = ∅ then
17         data_neighbours ⇐ request_Neighbour_Load_Data()
18     end
19     else
20         if relFactor ≥ random_number(1, 100) then
21             send_migration_moves_to_caller(migration_moves)
22         end
23         else
24             data_neighbours ⇐ ∅
25         end
26     end
                                  /* Selection of overloaded resources based on neighbour CLBs' loads */
27     neighbours ⇐ identify_Neighbour_Less_Load()
28     if neigbours! = ∅ then
29         overloaded_resources ⇐ select(firstNeighbour)
30         federates ⇐ select(spec_loads, overloaded)
31     end
32     else
33         overloaded_resources ⇐ filter_resources(extStD, localStD)
34         overloaded_resources ⇐ order_resources(selectionParameter)
35         federates ⇐ select(spec_loads, overloaded)
36     end
        /* Resources' Information is sent to neighbour CLBs for further analysis, and it waits for
       migration moves to be triggered */
37     send_to_neighbour(overloaded_resources, federates)
38     comm_migration_moves ⇐ wait_for_migration_moves()
39     send_migration_moves_LLBs(comm_migration_moves)
40     comp_migration_moves ⇐ wait_for_migration_moves()
41     send_migration_moves_LLBs(comp_migration_moves)
                               /* Re-distribution parameters are adjusted for the next balancing cycle */
42     comm_adjust_factor(bottom, top)
43     comp_adjust_factor(relFactor, overloaded_resources, migration_moves)
44     wait( Δ )
45 end
```

ing process to the extent that just the relevant or essential information that is needed to perform balancing analysis is acquired. Thus, distributed simulations can present large-scale dimensions with negligible or minimal balancing overhead interference. The

aggregation is mostly applied when collecting load data from LLBs, which are deployed in each resource. A LLB evaluates and processes the data collected locally from each federate before it is forwarded to a CLB. During the redistribution procedure, aggregation is employed to exchange load information of each domain, decreasing the amount of federate and resource data communicated between CLBs.

In Algorithm 6, data gathering is the first step in the balancing algorithm. The data is collected from the Monitoring Interface Service to retrieve load information from each resource. Then, detailed load data is gathered from each federate through LLBs. After all monitoring information that belongs to a domain of resources is collected, filtering is applied to the data. The reason filtering is applied between communication and computation load balancing procedures, besides removing resources that are not managed, is to prepare the gathered data for the next balancing steps of detection and redistribution. A detection mechanism is used to determine if the system is imbalanced and needs to have its load redistributed. This detection step in the balancing scheme is related to the type of analysis that is performed: communication or computation.

5.3 Communication Balancing Algorithm

In a centralized redistribution of load, synchronization is employed among all involved balancing components in order to provide consistency of actions and accurate information. In addition, time coordinated interactions among load management components introduce global synchronization in the whole balancing system. The synchronization forces all the distributed components to operate at the same pace, induces the balancing system to be susceptible to failures, and produces overheads and delays that proportionally increase with the scale of the balancing system. The part of the balancing system that depends upon a global view of a distributed system to detect and react to imbalances is impaired by this synchronization approach, but the local balancing of the scheme is not fully exposed to fault because the CLBs that are not affected by the failure continue to function accordingly. In the case of centralized communication balancing, local balancing is never realized because the redistribution scheme needs to be aware of whole distributed system to identify corrections and perform inter-domain federate migrations. If the load management element of this centralized scheme that is the root in the hier-

archical architecture fails, the sub-load management elements are incapable of providing sub-balancing, which results in the failure of the entire communication balancing scheme.

To prevent the issues originated from the centralized redistribution, asynchronous relations between hierarchical, independent CLBs are introduced in the balancing scheme. In this case, aggregated monitoring information and redistribution calls are reported only to neighbour CLBs, and cumulative delays and waiting are not propagated inside the balancing system. Upon receipt, the calls are analyzed to accept federates in a domain of resources. The analysis can result in redistribution moves and then in migration calls. To perform this analysis, a CLB needs to compare the communication rates of federates contained in the call with the communication rate of its local federates, observing the availability of resources and their computational load status. The comparison of communication rates can be realized synchronously or asynchronously. For the synchronous comparison, a CLB uses the most accurate load status information in a domain; consequently, it needs to wait for the collection of current data. This technique produces delays and rarely uses the most recent status load information in comparisons due to waiting for local recent status data. The technique also propagates synchronization between CLBs, bounding their processing parallelism and independence. For the asynchronous comparison, the most recently collected local data is used to detect imbalances, and delays and propagated synchronization are avoided. The accuracy of local status load is related to the time in which the redistribution call is received, and this precision aspect is also present in the synchronous comparison. Since this local data is used as the prediction metric in the balancing scheme and is employed to determine the range of acceptance of a CLB, the data is required to represent an average or a tendency of communication load that belongs to certain domain in the recent past.

The proposed redistribution phase attempts to mimic the behaviour of a centralized redistribution scheme, in which a central element has a global view of the entire distributed environment. Generally, all federates in HLA-based simulations generate different communication rates, and these federates are statically deployed on computing resources that are inter-connected through network links with different nominal transmission capacities and different communication distances (hops). To introduce awareness about the network topology into the balancing system, CLBs are organized in layers according to the topology of the distributed system on which simulations are running.

Each CLB is configured to be connected with its subsequent and antecedent CLBs in the hierarchical structure, which are aligned with balancing needs in a given moment. The alignment is defined in each CLB by two thresholds which dictates the orientation and intensity of redistribution of federates on the shared resources. The two thresholds represent the maximum and minimum communication rates that determine a domain and are based on the sample data gathered during a balancing cycle. Using the amount of communication between federates in the last balancing cycle (recent past), the thresholds are used to select federates that do not reflect the communication rate of a domain and need to be moved to a proper domain. The thresholds (bounds) are constantly calibrated to adapt to dynamic communication changes and produce the impression of a centralize, global-view analysis. The calibration also directly influences the responsiveness of the communication balancing scheme by inducing federates to be migrated to other domains.

An α and a β are established for each CLB to determine the communication imbalances in a cluster of resources, as depicted in Figure 4.2 in Chapter 4. In order to establish the hierarchical organization of the balancing system, it is assumed that $\alpha_{l1} > ... > \alpha_{ln}$ and $\beta_{l1} > ... > \beta_{ln}$ are sequences organized in descending order, observing that all $\alpha_{ln} > \beta_{ln}$ and all $\beta_{ln} \geq \alpha_{ln-1}$. To maintain the coherence of the balancing organization, the bounds αs and βs are induced by the communication capacity of resources and are modified by inter-relations between CLBs, which result from the redistribution calls originating from neighbour CLBs. The adjustments of these bounds occur periodically during each balancing cycle: when recent sample data is gathered and fed to the balancing system with the communication rate of each federate. Additionally, explicit adjustment regarding the previous migration move calls is employed to re-calculate the bounds, so they are calibrated according to the success in achieving federate migrations, reflecting the need of the system to have the load moved. Thus, each CLB that receives a call containing federate candidates for federate migrations analyzes them and answers back with the proper migration calls and a direct suggestion for adjustment.

5.3.1 Definition of Migration Moves

Based on the hierarchical structure definitions, as well as the bounds used to organize the load redistribution, migration moves always occur between neighbour CLBs, so federate candidates for migration are not sent directly to their most appropriate destination.

A federate would be transferred to the correct destination only if a CLB was directly connected with the CLB responsible for the destination. Consequently, based on the matching check with the top and bottom bounds of a CLB and the communication rate of a federate, the federate is transferred gradually to positions that are closer to its final destination until it reaches the correct position in the hierarchical structure. Depending on the architectural hierarchy of the balancing system, the process of moving a federate to its final destination may take several balancing cycles. This process introduces inefficiency of the load redistribution and generates large delays and overhead through the consumption of a considerable amount of resources to perform all the required migrations. As a result, a jump condition is required to avoid these unnecessary migrations, saving time and communication and computation expenses.

5.3.2 Cluster Load Balancer Inter-Relations

Even though the balancing system is composed of distributed load management elements that work independently, it needs to create the sense that all the hierarchical, distributed system acts as centralized. To produce a pseudo-global view of the entire distributed system, rules are defined to regulate the bounds of each CLB, and migration move policies are introduced. The rules reflect the topological knowledge of the distributed system and coordinate the migration move policies. Such rules are configured according to the organization of the hierarchical structure, which obeys the topology of resources. The rules also depend on the arrangement between CLBs: the definitions of subsequent and antecedent neighbour CLBs in the hierarchical structure.

Based on the topology of the environment, each CLB manages its inner bounds, α and β, to move federates while focusing on minimizing the communication delays according to the hierarchy of balancing elements. In such a hierarchical organization (root, intermediate, and leaf CLBs), the root's α contains value infinite and the leafs' β contains value -1 or minus infinite, delimiting the boundaries of the balancing system, which are the closest and the furthest position in each branch of the structure. The organization and the relations between CLBs produce the following situations for calibrating the balancing bounds:

$$\beta_1 = \alpha_2, \beta_2 = \alpha_3, ..., \beta_{n-1} = \alpha_n. \tag{5.1}$$

$$\beta_{n-1} < \alpha_n or \beta_{n-1} \ll \alpha_n. \tag{5.2}$$

$$\beta_{n-1} > \alpha_n or \beta_{n-1} \gg \alpha_n. \tag{5.3}$$

- α: top communication load range boundary;
- β: bottom communication load range boundary.

Expression 5.1 denotes the ideal communication distribution state of all bounds which the balancing system aims to reach as it represents the situation when the system is completely balanced. However, this situation is not achieved because of the need to measure and create interactions between CLBs in order to constantly check a system's communication balance and keep the balancing system dynamically reactive to communication load changes.

In Expression 5.2, the overlap of two sequential domains is produced. In this situation, every federate candidate that is sent to a CLB is successfully redistributed and generates a federate migration call. In this case, this approach minimizes the number of calls due to the increase in acceptance range of a CLB, and imbalances are barely detectable because of the overlapping condition between two CLBs' domains. Therefore, the relation between CLBs represented by this expression is avoided because it impairs the responsiveness of the balancing system and fails to promptly detect communication load changes.

Expression 5.3 generates a discontinuous breach between the domain rage of two CLBs. The breach causes the constant generation of redistribution calls even if the distributed system achieves a balanced state. These redistribution calls can contain some unnecessary federate candidates for migration, which introduces a slight balancing communication overhead in the system. However, such calls are needed in the structure in order to regulate CLBs' thresholds periodically, so the CLBs can adapt to the communication changes accordingly and promptly. Through the inter-relation policies, the calls force each CLB to update its internal variables and make each CLB to directly or indirectly adjust the variables of its interacting CLBs. In a direct regulation, a call, which contains a calculated value based on the generated migration calls, is sent to a sub-CLB to be considered in the re-calculation of its α. In a indirect regulation, the

non acceptance of redistribution calls incites the sender CLBs to adapt their α values accordingly, so in the next balancing cycle, they perform a more coherent detection of imbalances.

Regarding the redistribution of federates, the balancing follows the condition defined by Expression 5.3 as a rule to adjust bounds of acceptance ranges. According to the expression, CLBs send federate candidates that might match with the acceptance range of their neighbours. Following this rule, three situations determine the redistribution behaviour of a CLB: the presence of computationally underloaded resources; the existence of computationally overloaded resources; and the arrival of a federate candidate with an extreme or negligible communication rate.

The first situation expresses the availability of resources for receiving load, so federate migration can benefit the simulation performance through the resources' position, i.e., the proximity of the resources. This situation occurs when the number of federates in a domain is smaller than the number of federates that can be supported by the available resources in the domain. According to the order imposed by the topological distances, a CLB in such a situation needs to enforce the move of federates from its sub-CLBs to its resources, so the communication of such federates presents a delay decrease, leading to an improvement in simulation performance. As a result, a CLB lowers its β (bottom bound) and incites its sub-CLBs to decrease their α (upper bound). With this tactic, the CLB forces federates to be migrated to its resources.

The second situation represents the lack of availability of resources, so all the resources managed by a CLB are completely loaded with federates and cannot receive any federate without performing further computation load analyses. If the resources are not able to admit any more federates, federate moves and exchanges are realized to re-organize the load and decrease the communication overload, keeping the distributed system's computation load balanced or benefiting the distribution of the computation load. In this case, a CLB narrows its acceptance range, inducing federates to be migrated to resources managed by another CLB and relieving the load of local resources. With the relief of the load, the CLB is able to admit federates in future balancing cycles. Moreover, federate exchanges enable the reallocation of federates to benefit the communication load. For the exchange, a CLB elects a federate to be moved in exchange of a federate candidate. The election of this federate depends on its communication rate and the direction in

which the communication balancing is being performed.

The third situation describes the case in which the communication load of federates in redistribution calls exceeds even the opposite threshold of a CLB. Because of the dynamic characteristics of distributed simulations, federates might suddenly change their communication behaviour. For example, a federate that was positioned far from the RTI due to its past communication behaviour may suddenly become communicative and need to be re-positioned to decrease its communication delays. The abrupt, extreme changes in communication rates of these federates cause them to be transferred to other resources; however, several balancing cycles are required to completely transfer such a communicative federate to its final correct position. Therefore, to speed up the transferring process, a CLB allows a federate to jump several distances or domains (CLBs) until the federate reaches the proper resource for its communication rate. When the communication rate of a federate in a redistribution call exceeds the opposite bound of a CLB, the CLB simply forwards it to the next known CLB in the topology sequence. If the communication rate is high enough, this procedure is repeated many times, following the hierarchical layers of the balancing structure. If a federate matches this condition and is forwarded to a subsequent CLB, the forwarding CLB along with the sub-CLBs involved with the federate candidate awaits an answer from the destination CLB. Upon a rejection of a federate candidate, a forwarding CLB, which becomes the most suitable position for the federate candidate, keeps the federate and answers back to the sub-CLB that sent the redistribution call.

5.3.3 Adjustment of the Communication Balancing Thresholds

The adjustments of bounds occur asynchronously and periodically according the load rearrangement needs. Focusing on an inter-relation between a CLB and its upper CLB, it is observed that for the CLB's α and its upper CLB's β, the balancing system attempts to reach the relation $\alpha \simeq \beta$ and $\alpha \leq \beta$. Because each CLB works independently, it realizes the balancing analysis of its federates and resources based on only the locally collected data sample. Since the scope of the balancing system only regards the communication delays, the communication balancing scheme is federate-centred. The scheme is mainly focused on communication rates to detect and react to imbalances. Standard deviation is used to partially calculate the top and bottom bounds of a CLB, leading to flexibility

by adapting when the communication rate increases or decreases. Due to the inter-relations between CLBs, each CLB needs to adjust its bounds to its upper and lower CLBs through a calibration parameter (θ). Thus, the bounds are a result of a merging between asynchronous adjustments and self-redistribution, as shown in Expression 5.4.

$$\alpha_{ln-1} = \alpha_{ln-1} * (1 - \theta) + \beta_{ln} * (\theta), \theta \geq 0. \tag{5.4}$$

- α_{ln-1}: a CLB's top communication load boundary;
- β_{ln}: a CLB's upper CLB's bottom communication load boundary;
- θ: calibration parameter that defines the CLB's inter-relations.

The parameter θ is used as a control to assure the dependency relations between CLBs. In the scheme, θ is related to the amount of success in generating migration moves from the federate candidates that are received, i.e., the fraction between the two values, $moves/candidates$.

5.4 Computation Balancing Algorithm

The proposed computation balancing algorithm is divided in two scopes: local and inter-domain. The redistribution in the local scope is responsible for rearranging 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 use 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 domains.

5.4.1 Load Redistribution in Local Scope

In local balancing, the redistribution phase is simply realized as an exchange of load between local resources. The balancing is divided into monitoring, redistribution, and migration steps, such as in the computation balancing technique introduced in 4. According to the local balancing algorithm, a CLB initially collects load information from

Algorithm 7: Pair-Match Evaluation Algorithm

```
1  Require: src_rsc, dst_rsc
2  if dst_rsc < min then
3      if number_fed(src_rsc) > 1 then
4          create_migration_move(src_rsc, dst_rsc)
5      end
6      else if number_fed(src_rsc) = 1 & src_rsc > (min * φ) then
7          create_migration_move(src_rsc, dst_rsc)
8      end
9  end
10 else if (dst_rsc − src_rsc) > (min * δ) then
11     if number_fed(src_rsc) > 1 then
12         create_migration_move(src_rsc, dst_rsc)
13     end
14     else if number_fed(src_rsc) = 1 & (dst_rsc − src_rsc) > (min * φ) then
15         create_migration_move(src_rsc, dst_rsc)
16     end
17 end
```

managed resources, filters it, and classifies the resources as overloaded or underloaded as described in Algorithm 6. After this classification, the CLB performs pair analysis of resources where it identifies migration moves based on the load differences between an overloaded resource and an underloaded resource. As delineated in Algorithm 7, this pair-match analysis uses load redistribution rules to determine the need of federate migration. The rules are used to check the availability of computing power of the destination resource based on overload of the source resource. In such rules, awareness of external background load, heterogeneity of resources, and partitioning of load are included in the pair comparisons. After the moves are generated, they are sent to their respective federate to perform the needed load changes.

5.4.2 Load Redistribution in Inter-Domain Scope

In the inter-domain balancing, the redistribution algorithm coordinates the arrangement of the simulation load in pairs of domains. This pair reallocation of load allows the balancing system to gradually achieve a system wide balance, avoiding synchronization, overhead, and a single point of failure. Different from global, centralized balancing, the redistribution does not require global awareness for analyzing the computational load status in a system. Based on a push method, the inter-domain is triggered by an overloaded CLB to facilitate the balancing when the system presents a overall increase in load. According to the balancing scheme, an overloaded CLB initiates the redistribution process by selecting a neighbour CLB that is the most appropriate location to send

the local load based on the current knowledge of the distributed neighbourhood. As a result, for each inter-domain load reallocation, the migrating federates are recognized by a receiver CLB as a new load that is submitted to its domain.

As described in Algorithm 6, the inter-domain balancing is necessarily performed after the local balancing. This policy, adopted in the balancing scheme, assigns more priority to the local redistribution analysis. The priority allows the system to enforce the achievement of even distribution of intra-domain load before external measures are used to decrease the amount of load in a domain. With this approach, inter-domain redistribution is employed when the local distributed load achieves some stability after load changes are generated due to simulations or previous load rearrangements. In this case, a broader balancing algorithm is applied to identify available resources remotely. The inter-domain redistribution is triggered if the load imbalances detected by the local scheme are not large enough to produce federate migrations.

To control the frequency in which the inter-domain balancing is activated, a distribution factor is introduced in the general balancing algorithm. This factor influences the relations between CLBs, reflecting the need to realize distributed balancing through an indicator that evidences the degree of imbalance between different domains. The value of this factor increases proportionally to the number of migration moves. The migration moves reflect the difference of relative load between two CLBs and consequently the need for more load exchange. In order to check imbalances between CLBs, the factor is initialized with a maximum value of 100%. At the end of each balancing cycle, this relation factor is adjusted to conform with the real load status between a CLB and its neighbour CLBs. As detailed in Expression 5.5, the adjustment is calculated based on the migration success rate of inter-domain redistribution between two CLBs in a balancing cycle. The success rate consists of the ratio between the number of migration move produced in the end and the number of resource candidates selected for migration ($extmig/extcadrsc$).

$$fac_n = -(\alpha_{n-1}^2) + \alpha_{n-1} + \alpha_{n-1} * fac_{n-1}, n > 0 \tag{5.5}$$

- α_{n-1}: migration success rate for the previous balancing cycle;
- fac_{n-1}: inter-domain adjustment factor defined in the previous balancing cycle.

Inter-relations between CLBs are vital for maintaining the inter-domain properly. These relations are introduced as an adaptation of the distributed balancing scheme, which is is triggered when necessary, as detailed in Algorithm 6. The adaptation of the scheme reflects the balancing needs, and the inter-relations induce the redistribution to proactively react to intra-domain load oscillations and initiate the detection of imbalances between domains. Based on the load analysis, which considers the average domain load and its respective standard deviation, the relations between a local CLB and an neighbour CLB is delimited by three situations, which are basically the differences in the relative load of CLBs.

The first CLB inter-relation situation is represented through an inequality, which is expressed by $CLB_{external}.load < CLB_{local}.load + \alpha$. The inequality employs load averages to identify that the resources managed by the remote CLB present less load than the local resources. Based on this situation, the balancing system determines a selection of redistribution moves as a initialization of the inter-domain balancing, which requires further analysis performed by a remote CLB.

The second situation is described using two inequalities that are represented by $CLB_{external}.load \geq CLB_{local}.load + \alpha$ and $CLB_{external}.StD \geq CLB_{local}.StD + \beta$. These comparisons evidence the larger amount of relative load that the remote resources present when compared with the local resources, but they also show a larger standard deviation of the average load of remote resources. Even though the remote resources contain more load on average, the average represents the result of a calculation with larger variation than the local resources' average load, as shown through the standard deviation analysis. Redistribution moves are generated since this variation might expose the remote underloaded resources that are hidden by the average load.

The third situation is represented by the following two comparisons: $CLB_{external}.load \geq CLB_{local}.load + \alpha$ and $CLB_{external}.StD \leq CLB_{local}.StD + \beta$. In this situation, the local resources contain less relative load than the remote resources and exhibit larger variations on their average load. Through the standard deviation analysis, the remote resources present a lower probability of existing underloaded resources than the local resources. As a result, no redistribution move is selected for triggering further analysis. This difference of load can be an evidence of imbalance, but the local CLB does not request any load from remote CLBs since the remote CLBs detect this imbalance in

their next balancing cycle.

As described in the above three situations and in Algorithm 6, the first step of the inter-domain redistribution algorithm is to detect load imbalances and select overloaded resources as candidates for further remote analysis. Upon receiving the average load from neighbour CLBs, the local CLB uses the inequalities described in each situation to determine a CLB to receive a load if any imbalance is detected. Initially, the remote CLBs are classified according to their load, and an ordered list of underloaded CLBs is created. The list contains all CLBs that comply with the first inter-relation condition. If this list is not empty, the first underloaded CLB is chosen to receive federates from the local CLB. The balancing scheme can distribute the excess local load to the selected underloaded CLBs instead of selecting only the first underloaded CLB from the list. According to the load of a chosen CLB, overloaded resource candidates and their respective federates are identified. Another selection analysis is employed to generate a list of CLBs if the list of underloaded CLBs is empty. This list contains CLBs that present a standard deviation value higher than the local CLB. Like the second inter-relation situation, this selection identifies a CLB that might contain resources which can receive load and contribute to the even redistribution of the load that is causing the imbalance. After these lists are created, the parameter $(CLB_{external}.load + CLB_{local}.load)/(CLB_{external}.load + CLB_{local}.load)$ is used as a factor to order the selected lists. The first CLB in this list is selected, and based on its average load, the overloaded resource candidates are identified for rearrangement.

After the selected resource candidates are received in a remote neighbour CLB, the CLB initiates the proper analysis to accept the receiving load and produce migration moves. Similarly to the local redistribution algorithm, pair-match comparisons are performed by the remote CLB to identify destination resources for the selected load transfers. First, the CLB orders its local resources ascendingly and the receiving resources descendingly. This ordering facilitates the selection and matching of overloaded resources with the underloaded resource. Each external resource is matched with a local underloaded resource through load comparisons, as described in Algorithm 7. If the load difference between the resources represents a condition of load imbalance, a federate migration move is generated with the federate previously selected by the CLB which issued the redistribution call. At the end of the inter-domain redistribution, a set of migration moves is produced, and then the set of migrations is returned to the calling CLB to be

processed, generating migration calls.

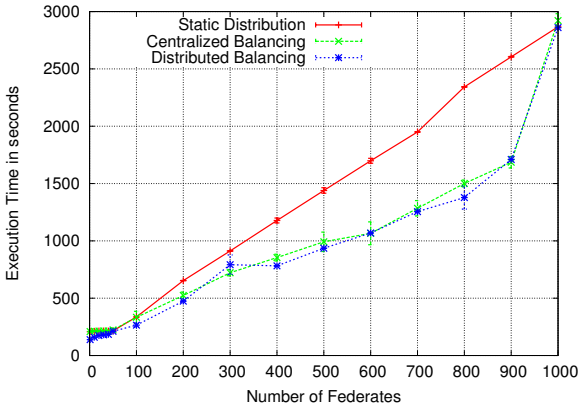
5.5 Experiments and Results

The devised distributed balancing scheme aims to improve fault tolerance, decrease delays originating from communication and computation overhead, and avoid the existence of bottlenecks for the balancing of simulations. Although the distributed scheme improves the qualities of the balancing system, it needs to reach its main objective of reacting to communication imbalances when necessary. Three experimental analyses have been conducted to determine the efficiency of the proposed balancing scheme in detecting and reacting to communication imbalances. The experimental evaluation consisted in comparing the balancing performance of the distribution balancing approach with the centralized approach described in Chapter 4. The experiments comprised the execution of large-scale HLA-based simulations, and they were realized on a distributed environment composed two computing clusters connected by a fast-Ethernet network connection. For comparison purposes, the same scenario described in 4.7.1, which comprised the distributed environment, the simulation scenario, and the synthetic load, has been used to conduct the experiments.

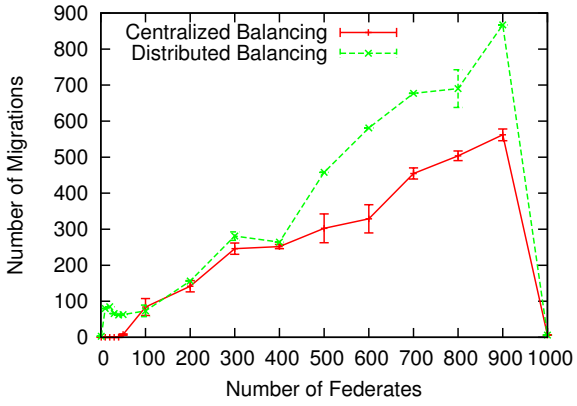
The experimental results were generated from the calculation of an average of 10 runs for each plotted point in the graphs. All averages in the results thus showed a confidence interval with confidence level of 95% based on a Student's distribution [51, 46].

5.5.1 Computation Balancing Analysis

In the computation balancing analysis, the simulation scenario was specified to produce intensive computational load and minimum communication overhead. The negligible communication load allowed a focused analysis of the proposed distributed balancing for computation imbalances. In this case, the single object update of a federate in a simulation step produced just 100-byte load for the network resources. The computational load is highly intensive and totally consumes the CPU in a simulation interval. Due to the heterogeneity of experimental resources, the even deployment of simulation federates generate an imbalanced distribution of load in the system. The analysis was divided into four experiments to evaluate the proposed balancing scheme performance: a static



(a) Performance Comparison



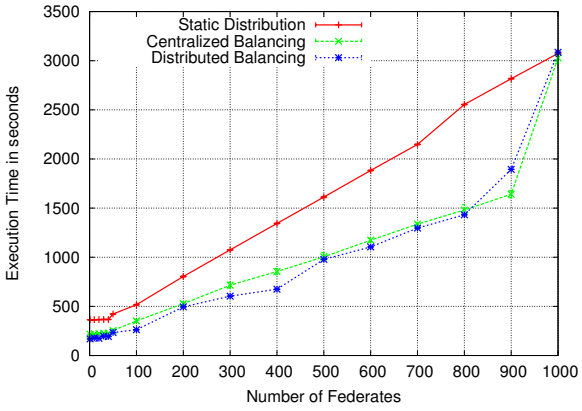
(b) Number of Migrations

Figure 5.2: Evaluation of the Computation Balancing for an Increasing Number of Federates with Static Load

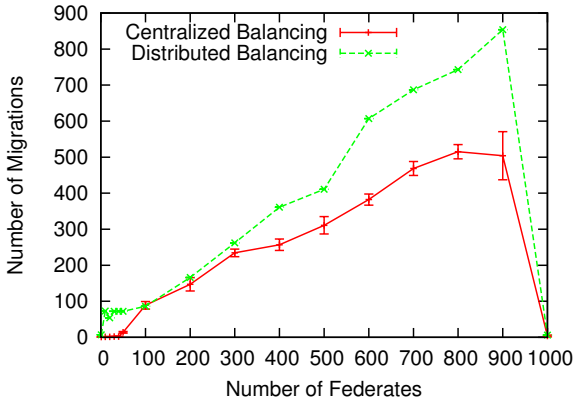
imbalanced load distribution; a scenario containing a static, external background load; dynamic simulation imbalances, and a dynamic, external background load.

In the experiments with static loads, the performance gain provided by the proposed redistribution algorithm was evaluated. As shown in Figures 5.2a and 5.3a, the distributed balancing was compared with a centralized balancing technique and with a baseline. In both graphs, the distributed balancing approach presented a performance improvement for simulations similar to the centralized approach, even with an external background load, which totally consumed all processor cores. However, the distributed re-partitioning presented a larger number of migrations, shown in Figure 5.2b and 5.3b 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.

In the dynamic scenarios, the reaction analysis observed the capacity of the proposed balancing scheme to detect dynamic imbalances, react, and generate the proper load modifications. As shown in Figures 5.4a and 5.5a, the distributed approach reduced the simulation execution time equivalently to the centralized approach. In Figure 5.4a, 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 a considerably larger

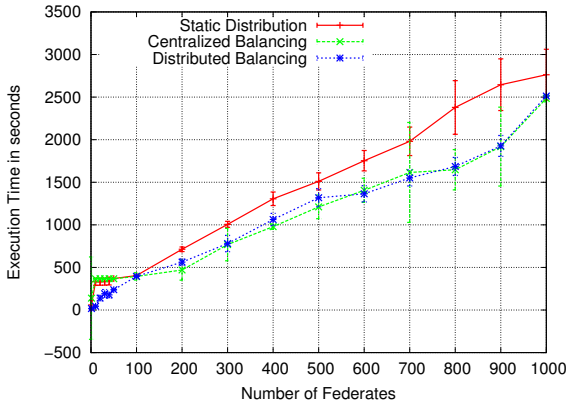


(a) Performance Comparison

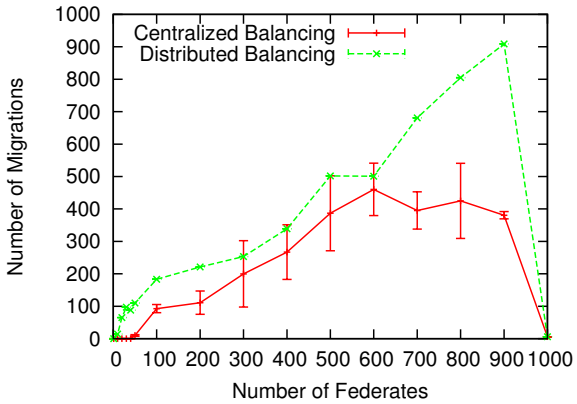


(b) Number of Migrations

Figure 5.3: Evaluation of the Computation Balancing for an Increasing Number of Federates in Presence of Static External Background Load



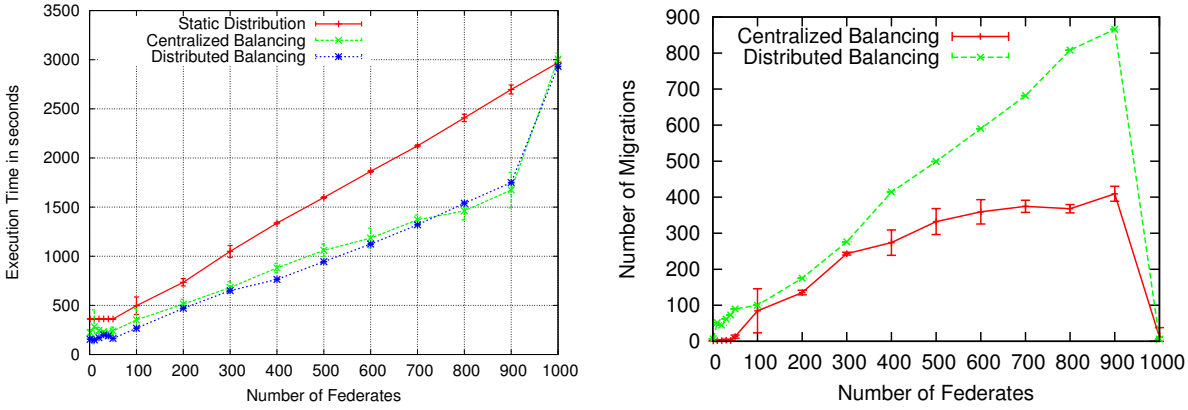
(a) Performance Comparison



(b) Number of Migrations

Figure 5.4: Evaluation of the Computation Balancing for an Increasing Number of Federates with Dynamic Load

number of migrations than the centralized balancing, as depicted in Figures 5.4b and 5.5b respectively. However, in all the experiments, the curves converged to the same point when the simulation contained 1000 federates. This convergence occurred due to the load saturation of the distributed system, which reflected in a few migrations.



(a) Performance Comparison

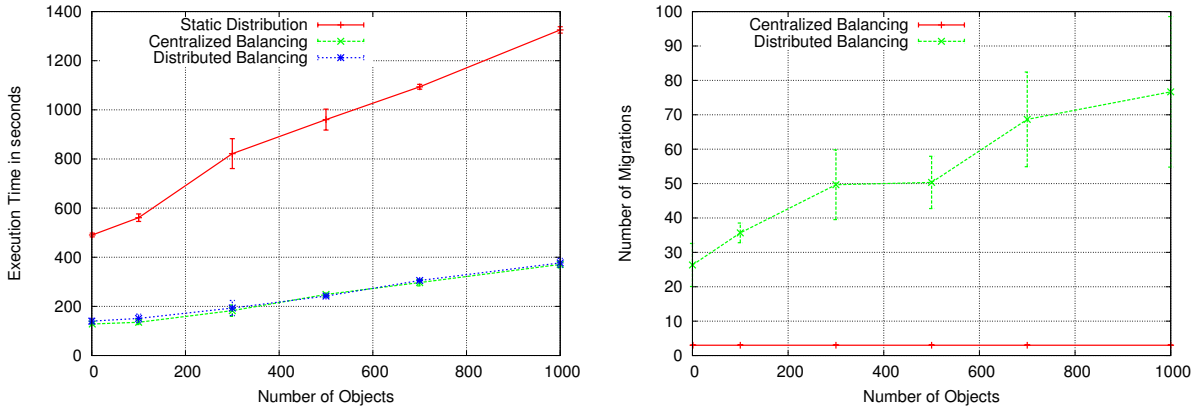
(b) Number of Migrations

Figure 5.5: Evaluation of the Computation Balancing for an Increasing Number of Federates in Presence of Dynamic External Background Load

5.5.2 Communication Balancing Analysis

For the communication balancing analysis, the simulation scenario was modified to produce massively imbalanced communication rates without any interference of computation load. The simulations were composed of 500 federates that managed 1 to 1000 simulation objects and produced light, simple processing. To insert controlled communication latency imbalances in the experimental simulations, some federates published data regarding special objects that were designed to generate a large communication overhead. This special object required an update that produced a 54000-byte communication load per simulation step while a common object produced 100 bytes per update. The analysis of communication balancing was divided into three experiments to evaluate the performance gain for simulations with static and dynamic communication imbalances.

The performance gain introduced in the HLA simulations and the reaction to a communication imbalance were observed in the first experiment. Such observation was enabled by introducing an increasing communication overhead caused by a small number of federates in a simulation. The overhead comprised the publication of a special object by a federate and the subscription to the same objects by other three federates. As shown in Figure 5.6, the execution time of the simulations was directly influenced by the increase in communication load, mainly for the baseline, which was not balanced. The centralized and distributed balancing systems presented a substantial performance



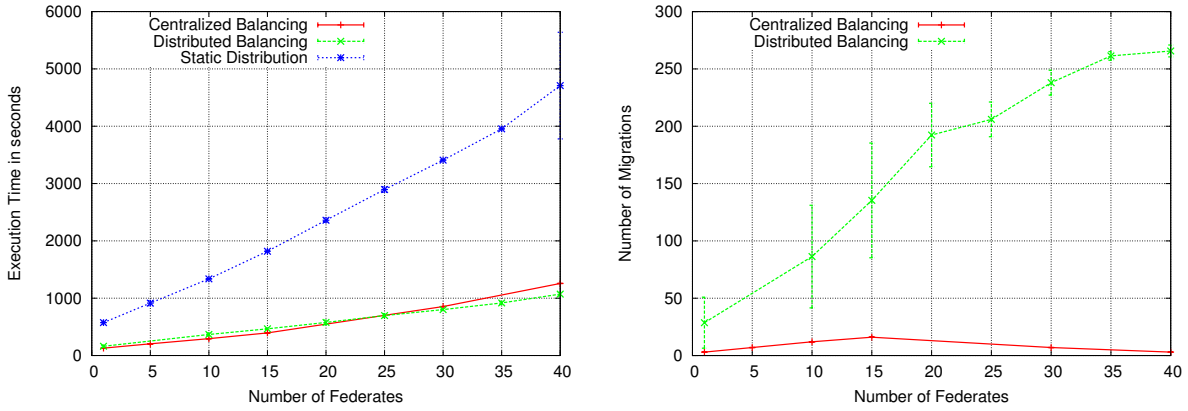
(a) Performance Comparison (b) Number of Migrations

Figure 5.6: Evaluation of the Distributed Communication Balancing for a Static Increasing Communication Load

gain when compared with the baseline curve in Figure 5.6a. However, in Figure 5.6b, there is a noticeable difference in the number of migrations. The centralized approach only performed the migration moves for improving the communication latencies for the communicative federates, but the distributed approach continuously realized migrations while there were available resources with a smaller communication distance.

The reactivity of the proposed balancing system to increasing communication overhead in the HLA simulations were analyzed in the second experiment. The analysis evaluated the balancing response to a higher communication overload, which consisted in inducing an overhead produced by 100 special objects in 1 to 40 federates. In this case, the balancing system needs to adapt accordingly, providing several migration moves with the objective of minimizing the communication latencies. As depicted in Figure 5.7, the baseline presented a high increase in simulation execution time as the number of communicative federates increased. As shown in Figure 5.7a, the centralized and distributed approaches presented a similar performance gain to that shown in Figure 5.6a. Figure 5.7b evidences that the distributed approach realized a larger number of migrations to achieve the same decrease of simulation time achieved by the centralized approach. As observed in Figure 5.7b, the centralized approach did not move all the communicative federates in the simulation, denoting its limitation in detecting overloads.

The efficiency in detecting and reacting to dynamic changes of communication load

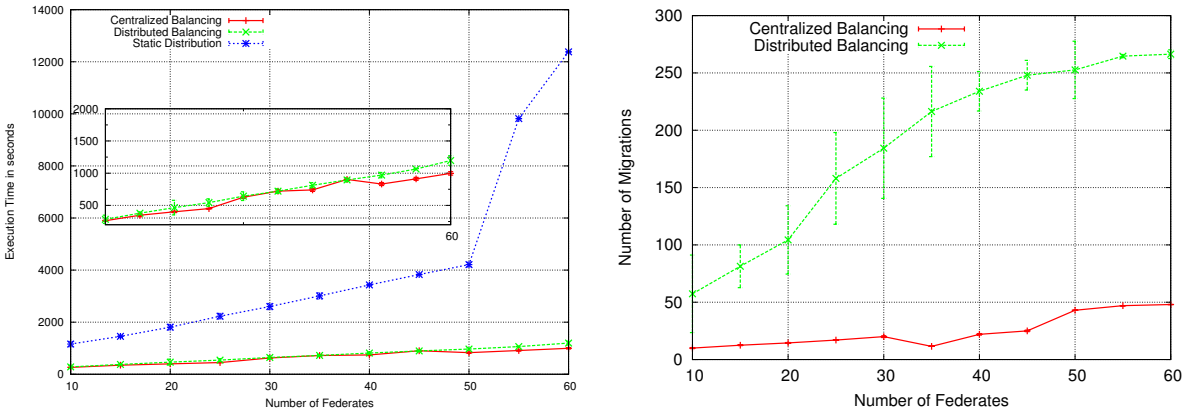


(a) Performance Comparison (b) Number of Migrations

Figure 5.7: Evaluation of the Distributed Communication Balancing for an Increasing Number of Communicative Federates with Static Communication Rate

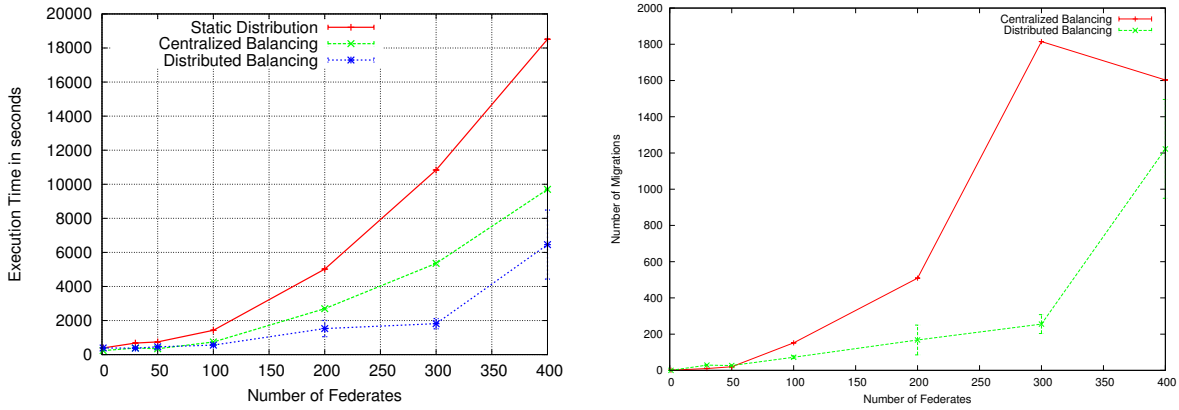
was evaluated in the third experiment. Similar to the previous experiments, the simulations were composed of 1 to 60 communicative federates that performed data updates each time step. The updates of these communicative federates contained a random number of special objects ranging between 1 to 100 to induce a dynamic and unpredictable behaviour during the execution of the experimental simulations. As depicted in Figure 5.8a, the balanced simulations outperformed the baseline simulation. More specifically, after 50 federates, the baseline presented a significant increase in the simulation caused by the cumulative saturation of communication, which did not influence the balanced simulations. When comparing only the balancing systems, they both showed similar performance gain, but with differences in the number of migrations, as shown in Figure 5.8b. The centralized balancing realized less migrations than the distributed balancing, evidencing that the distributed approach has a strong tendency to react to slight changes the communication load.

Through the experiments, the distributed balancing of communication showed similar performance to a centralized balancing scheme for large-scale HLA-based simulations. The amount of computation and communication required for managing the communication load grows in proportion to the magnitude of the simulation that is balanced. For large-scale HLA-based simulations, the central element that redistributes the load becomes a bottleneck in regards to computation and communication aspects. The exis-



(a) Performance Comparison (b) Number of Migrations

Figure 5.8: Evaluation of the Distributed Communication Balancing for an Increasing Number of Communicative Federates with Dynamic Communication Rate



(a) Performance Comparison (b) Number of Migrations

Figure 5.9: Evaluation of the Distributed Computation and Communication Balancing for an Increasing Number of Communicative Federates with Static Load

tence of only one element that performs the main tasks of a system generates risks that open the system to generalized failure. Thus, centralized balancing for such simulations can compromise the redistribution of load.

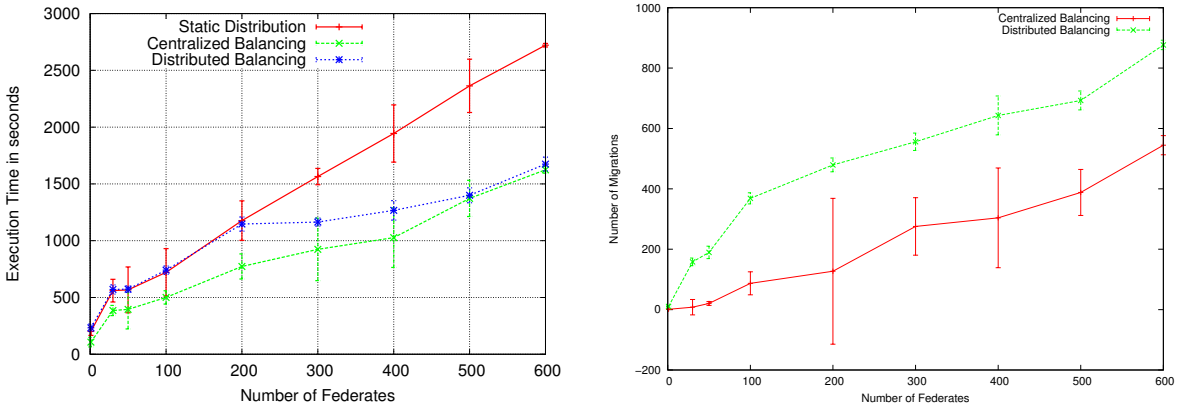
5.5.3 Computation and Communication Balancing Analysis

Computational and communication loads were introduced in the simulation scenario to observe the detection and reaction of the proposed balancing system. In this case, the

simulations present a better resemblance to normal simulations, which contain computing intensive tasks and constant interactions between federates. These simulations were composed of 1 to 600 federates that intensively processed computational tasks and contained 1 to 100 objects. An external background process was inserted in the system to produce computational load in a resource, completely and incessantly consuming a CPU.

In the first experiment, a static scenario was introduced. In the scenario, an external process was added to a resource, all federates produced intensive computational load, with 10 of federates updating 100 special objects. As depicted in Figure 5.9a, even though the proposed balancing scheme is supposed to have similar performance as the centralized redistribution scheme, it presented better performance. This was a result of the fewer number of migrations, shown in Figure 5.9b. The proposed approach was expected to perform considerably more migrations due to its inter-relation rules, which would incite more redistribution analyses, generating better responsiveness. However, the approach reduced the migrations and improved the performance gain because bounding policies were inserted in the balancing scheme to allow the coexistence of both distributed balancing algorithms, such as verification of computational load balancing when performing communication load redistribution.

In the second experiment, all the computational and communication load parameters changed dynamically during the experimental simulations. In the simulation scenario, all federates randomly oscillated their computational load intensity from light to high. Ten percent of the federates were selected to update special objects in each simulation step, and the number of these objects randomly changed from 1 to 100 every 40 seconds. Also, the external load consumed the CPU of a resource that was randomly selected each 40 seconds. In this dynamic experimental scenario, the distributed balancing approach performed worse than the centralized scheme, as depicted in Figure 5.10a. In contrast to the unusual performance results in the static scenario, the performance loss in this experiment was caused by the higher number of migrations, as shown in Figure 5.10b. This high number of migrations evidences the high responsiveness to load changes, but this responsiveness was hampered by the time interval of 40 seconds chosen to apply random changes in the experimental scenario. This interval was considerably short when compared with the balancing interval used to configure the balancing system, which is 20 seconds. With such fast load changes, the load redistribution significantly lost its



(a) Performance Comparison (b) Number of Migrations

Figure 5.10: Evaluation of the Distributed Computation and Communication Balancing for an Increasing Number of Communicative Federates with Dynamic Load

effectiveness since the time for the produced load reconfiguration to generate performance gain was coincidentally overlapped by the time interval of the experimental scenario.

5.6 Summary

The main objective of the proposed balancing scheme is to overcome the drawbacks of a centralized scheme: a single point of failure, communication and computation overhead, and system bottlenecks. In order to detect and react to dynamic communication load changes, the proposed scheme is composed of three phases: monitoring, redistribution, and migration. Grid services are employed in these phases to collect information about computation load status of resources and to perform reliable submission of federates and data transfer of their static files. Moreover, the structure of the balancing system is organized hierarchically in several layers, facilitating data gathering, decreasing overhead, and enabling distributed rearrangement of load. In the structure, the balancing components act independently and asynchronously, and it uses interactions in order to move federates and reorganize simulation federates to maintain an evenly distributed load while minimizing communication delays.

The experiments proved that the distributed balancing system is able to provide similar performance gain as the performance increase provided by the centralized scheme

used in the experiments. However, the distributed approach presented a considerably larger number of federate migrations than the centralized approach in order to achieve the same simulation performance gain. This difference in the number of migrations reflects the responsiveness of the distributed scheme, which reacts more easily to load changes, producing precipitated modifications in some very specific cases. Because of these additional precipitated load migrations, modifications to the distributed balancing scheme's design are introduced. As a result, based on this balancing scheme, migration history and migration latency are needed in the redistribution algorithm, so they are required to be analyzed and incorporated into the balancing design. Moreover, the awareness of these balancing aspects helps to prevent the creation of nomadic load (back and forth balancing issues) and to accurately produce federate migrations by avoiding costly data transfers.

Chapter 6

Extensions for the Distributed Load Balancing System

As a measure to further extend the distribute balancing scheme and improve responsiveness to load imbalances, extensions are devised and incorporated into its balancing design. The balancing extensions observe some specifics aspects from the balancing system or the distributed simulations. The aspects evidence situations and behaviours that if not considered by the balancing system might lead to loss of balancing efficiency and simulation performance. Four behaviours and situations were studied to produce the extensions: communication delay, migration history, migration latency, and projection on load samples. These metrics enabled the development of a delay-based balancing system that reacted to current load conditions of communication resources, a self-adaptive load redistribution scheme that observed the effectiveness of its own responses to imbalances to modify its balancing behaviour, a migration-aware balancing scheme that considered migration latency in its balancing decision-making, and a predictive redistribution algorithm that used smoothing and load forecasting to prevent imbalances and avoid precipitated balancing load moves.

6.1 Delay-Based Communication Balancing System

To react to dynamic communication load changes, the proposed balancing scheme incorporates periodic measurements of network conditions and communication behaviour,

load analysis, and rearrangement of HLA-based simulation federates. In the balancing scheme, the most recent status past is adopted as the main prediction metric. A decentralized load redistribution algorithm is employed to prevent issues present in centralized schemes, such as single point of failure, synchronization, and overhead. The algorithm acts according to the hierarchical structure that organizes the balancing components. This structure basically represents the network topology that determines the positioning of the computing resources and classifies simulations in different domains (clusters). A domain consists in a region of the environment that is monitored by a set of balancing components.

6.1.1 General Architecture of the Delay-based Balancing System

Similarly to the architecture of the distributed balancing scheme, presented in Chapter 5, some additional components are placed to enable awareness to communication delays in this extension of the distributed balancing system, as depicted in Figure 6.1. The load balancing system relies on Cluster Load Balancers (CLB) to coordinate all the balancing steps. This component manages the resources and simulation federates inside a domain, and it accesses information from other domains to detect imbalances. The balancing steps that are organized by a CLB are comprised of monitoring, redistribution, and migration. Monitoring is the initial part in each balancing cycle and requires the measurement of simulation and resource aspects. The balancing metrics are obtained through a Monitoring Interface component and Local Load Balancer (LLB). Based on such metrics, a CLB initiates the detection of imbalances and redistribution of load.

A LLB acts as an interface between a CLB and a set of federates in a resource. In order to retrieve monitoring information from federates and forward migration calls, the CLB is composed of a Communication Monitoring Interface (CMI) and a Migration Manager (MM). The CMI accesses each federates load status through the Federate Management Interface (FMI). The FMI observes the communication rate of a federate, calculates the measurements, and keeps the information in a table and in a circular queue. The table stores the communication behaviour of a federate, containing the communication destination and amount of data transmitted. The circular queue includes the

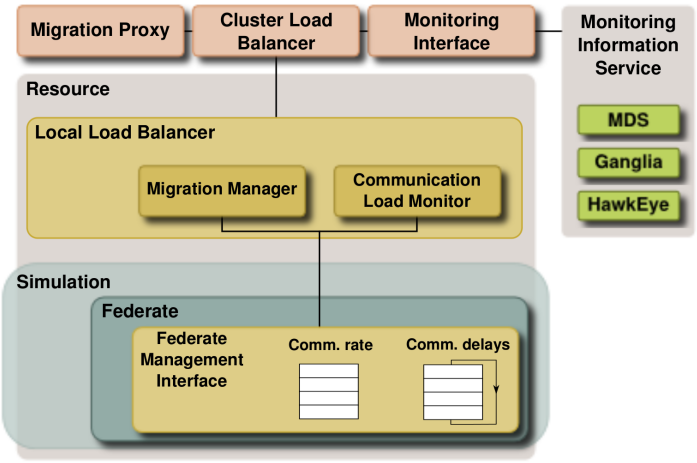


Figure 6.1: The Delay-Based Communication Balancing's General Architecture

communication delay for each simulation interaction to a remote federate. The (MM) component forwards migration calls to the respective federate and coordinates the process of transferring a federate to a remote location.

6.1.2 Load Redistribution Scheme of the Delay-based Balancing System

Monitoring, redistribution, and migration are the three major steps that compose the proposed balancing scheme. Monitoring requires the measurement of communication load status and communication resources conditions. The data collection is initiated in each domain by a CLB, which forwards the request to its respective LLBs that subsequently triggers a CMI. Upon the request of the CMI, a FMI collects the contents of the communication table and circular queue, processes them, and answers back. The processing of communication rates consists of summing all the data transmission entries in the table since all federates communicate with the HLA RTI. Regarding the communication delay, an average is calculated from all entries in the circular queue. The CMI merges the measured metric in a message through aggregation methods and sends the message to its CLB, which initiates the monitoring analysis. The CLB also retrieves and filters measurements related to the computational load of resources contained in its domain through a monitoring information system.

6.1.3 Detection and Redistribution Algorithm

The proposed detection and redistribution phase is the essential part of the balancing scheme since this phase dictates the responsiveness and efficiency of the scheme by delimiting the elements that are reallocated and their destination. Based on the placement of distributed balancing components, the characteristics and conditions of communication resources determine the relations between these components.

Delay is the metric employed in the balancing scheme to represent the dynamic conditions of communication resources. Together with the nominal network capacities, delay provides awareness of communication capabilities in a given moment. This metric indirectly supports the detection of network resource overload. By providing flexible means to perceive the dynamic communication load, delay is added to the analysis of the nominal network to extend the coordination of the inter-relations between CLBs. Therefore, since communication load changes dynamically, the direction in which simulation federates are moved is modified to reflect the available resources accordingly.

In the proposed balancing scheme, communication delay is assumed as the time in milliseconds spent to transmit one byte of data. This metric is obtained by locally measuring the amount of time taken to send each message of a simulation federate. A circular queue of size N is employed as the data structure to save every delay measurement. Since communication delay presents a volatile characteristic that reflects on frequent oscillations, an average is calculated from the saved measurements to generate an estimation that reflects a more stable delay value. Size N is used to restrict the amount of information that is stored locally and to emphasize the most recent communication status. In the case that every measurement of this metric is logged during a Δ time interval, recent communication changes occurring at the end of the balancing cycle are ignored because of the large quantity of old measurements in the beginning of the interval in a long list of delay values. On the other hand, a small parameter N might not be able to represent the real communication conditions since it can produce imprecise estimations that vary excessively. Thus, as a matter of simplification, the circular queue is restricted to 200 elements; this size accommodated both low and high communication load experiments.

Not only is communication delay measured, but the total amount of data transmitted and received by a federate is collected as well. The total communication load of a federate is collected, aggregated, and sent to its CLB. The aggregation is employed to

decrease the amount of data transmitted between balancing components and to minimize the interference with the network resources [73]. After monitoring data is retrieved, an average delay is calculated for every resource according to the average delay of its federates. A CLB also calculates the total amount of data exchanged by federates in its entire domain and the average domain delay. At the end when this information is produced, the CLB requests the average communication delay and total amount of transmitted data from neighbour CLBs.

As described in Algorithm 8, communication delays, amount of data transmitted, and goodput of a domain are used to identify imbalances. The goodput is a metric obtained at the application layer and represents the amount of data delivered per time. The goodput reflects the network distance of each domain from the RTI, and considers all the factors that influence the transmission time until data is delivered to the destination in its application level. Through these metrics, a greedy technique is used in the redistribution algorithm to identify a simulation distribution. The greedy analysis allows for a fast redistribution solution that offers a good load rearrangement, improving responsiveness. Therefore, based on delays, CLBs are ranked, and pair analyses are performed between the CLB and each neighbour CLB, starting with the CLB with the lowest delay.

Two actions are used in the redistribution algorithm to decrease the communication overhead. In the first action, the redistribution procedure identifies the neighbour CLBs that present communication delay plus an error lower than the CLBs. This approach transfers communicative federates to locations with better network conditions by iteratively analyzing delays and goodputs. In the goodput analysis, resources with higher goodputs are assigned to receive the most communicative federates, which are ranked in a list according to their communication rate. The list is classified in high, medium, and low communication intensity. The resources not close to the RTI but with lower delay averages receive federates classified as medium communication intensity. In the second action, neighbour CLBs with similar average delay ($delay_{ext} - error \leq delay \leq delay_{ext} + error$) and goodputs lower than the CLBs are selected to received federates with the lowest communication intensity. In this approach, the redistribution attempts to reduce concurrency of the communication link for highly communicative federates. The value of the *error* that determines the equivalence of delays is obtained based on percentage of the local delay, and it is defined as 3%.

Algorithm 8: Detection-Redistribution Algorithm

```
1  Require: delay_CLB, delay_list, abs_comm_CLB, abs_comm_list, goodput_CLB, goodput_list, federate_list
2  while delay_CLB > max(delay_list) do
3      order_ascendingly(delay_list)
4      temp_CLB ⇐ select_first(delay_list)
5      if delay_CLB > (temp_CLB.delay + error) then
6          if goodput_CLB ≤ temp_CLB.goodput then
7              data_comm ⇐ calculate_transfer()
8              repeat fed_eval ⇐ select_most_comm(federate_list)
9              if fed_eval.ab_comm ≤ data_comm then
10                 fed ⇐ fed_eval
11             end
12             until fed ≠ ∅
13         end
14         else
15             data_comm ⇐ calculate_transfer()
16             repeat fed_eval ⇐ select_med_low(federate_list)
17             if fed_eval.ab_comm ≤ data_comm then
18                 fed ⇐ fed_eval
19             end
20             until fed ≠ ∅
21         end
22         federate_mig.add(fed)
23         adjust_delays(delay_CLB, temp_CLB.delay)
24     end
25     else
26         endLoop
27     end
28 end
29 CLB_list ⇐ select_similar_delay(delay_list)
30 order_ascendingly(CLB_list)
31 for i = 1 to CLB_list.size() do
32 if goodput_CLB > CLB_list_i.goodput then
33     data_comm ⇐ calculate_transfer()
34     fed_eval ⇐ select_lowest_feds_fit(data_comm)
35     if fed_eval.ab_comm ≤ data_comm then
36         federate_mig ⇐ select_lowest_feds_fit(data_comm)
37     end
38     federate_mig ⇐ fed_eval
39     adjust_delays(delay_CLB, temp_CLB.delay)
40 end
41 Return: federate_mig
```

The next step in any of the communication delay conditions is identifying the correct federate to be migrated. This task occurs in iterations and requires an analysis of the communication load of a list of federates. The federate with the largest communication rate and less than the value obtained in Formula 6.1 is selected. The communication rate provides the means to assign a priority to federates. With this procedure, the balancing system prevents the production of imbalances by containing the load changes.

$$comm\_transfer = \frac{|d_2 - d_1| \times (a_1 + a_2)}{2 \times (d_1 + d_2)} \tag{6.1}$$

- d_1 and d_2: communication delay of domains (clusters);
- a_1 and a_2: communication load of domains (clusters).

The difference of communication load between two domains used to select federates is estimated in Formula 6.1. The formula employs the absolute amount of data transmitted in each domain (a) and the most recent average communication delay of their CLB to estimate the communication load difference. In this case, the simple subtraction is not aware of the difference between domains goodput and the external background processes. The recent communication and network pseudo-status are incorporated in the calculation through the introduction of delays. As a result, a more flexible value is achieved, which reflects the sudden network changes.

In each redistribution iteration, communication loads and delays of each CLB involved with a migration move are adjusted for the next analysis iteration. The adjustment is introduced to restrict the amount of federates that are selected for migration. This restriction also prepares the analysis metrics for the next iteration. The absolute communication load value is adjusted through a subtraction ($a - a_f$) or addition ($a + a_f$), in which a is a CLBs value and a_f is the communication load of a federate. For the communication delays, additional calculations are needed, which are performed through Formula 6.2. The formula uses the adjusted communication load value ($a - a_f$) to determine the value for the delay. Similarly, the delay of the neighbour CLB is obtained through Formula 6.3, which proportionally considers the federate load in the domain. Then, the ranking process of neighbour CLBs and the redistribution analysis are restarted.

$$adj_d = \frac{a \times d}{(a - a_f)} \tag{6.2}$$

- a: communication load of a domain (cluster);
- a_f: communication rate of a federate;
- a: communication delay of a domain (cluster).

$$adj_i = \frac{a \times d}{(a + a_f)} \tag{6.3}$$

- a: communication load of a domain (cluster);
- a_f: communication rate of a federate;
- a: communication delay of a domain (cluster).

When the load modifications are identified, federate migrations need to be issued, requiring federate identifications, source resources, and destination resources. All federate candidates belonging to the same destination domain are grouped in a list, and these federate lists are sent to their respective CLBs. Upon receiving this migration request, a neighbour CLB analyzes the computational load of its local resources in order to receive the federates. If computational overloads are not generated, the local resources with the least load receive the federates. Through this technique, computational imbalances are avoided while redistributing the simulation load for communication purposes. After resources are identified, the neighbour CLBs send their information to the CLB, which defines migration calls accordingly and forwards them to their respective federates. The resources involved with the triggered federate migrations are inserted in a list in order to be excluded from the next balancing cycle. Through this approach, enough time is provided to resources to establish their normal execution. Load irregularities (oscillations) that are caused by migrations are avoided.

6.1.4 Extension to the Delay-Based Balancing System

As exemplified in the experiments, the proposed delay-based balancing scheme presents efficiency issues related to the detection precision and the background communication load. The average communication delay in a great extent of the data sample mostly represents the current network load status. However, variations can be found in the details of this average and the majority of the variations misleads the decision-making of the balancing scheme. Imprecision originating from the communication delay oscillations in the data sample misguides the balancing scheme, which cannot identify imbalances properly or might produce unnecessary distribution modifications. The delay oscillations are caused by data transmissions with latencies that do not reflect the real conditions of the network resources, and these oscillations might exert significant influence on the monitoring metrics. Since the average communication delay is employed as the method to

determine the characteristic aspect of the network resources load situation, the amount of irregular variations in the data sample proportionally misleads the final value of communication delay. Consequently, for each delay calculation, the monitoring components observe the time spent to transmit a certain amount of data, and such time can be influenced by the simulation application or by other means from the under layers in the communication system. The influences need to be considered in order to provide a more accurate measure of communication delay. Furthermore, the other issue of the balancing scheme is the limitation of responsiveness to external background communication load. The dependency of the redistribution algorithm on the network topology hampers the redistribution actions, which have to avoid network links with large nominal bandwidth but presenting high consumption of communication resources. Therefore, a set of modifications on the redistribution algorithm is presented as an extension to improve the balancing effectiveness.

6.1.4.1 Extension of Communication Data Analysis

The extension of the data analysis consists of inserting techniques and mechanisms just after data is collected or just before comparisons and interpretations are realized to determine imbalances. Essentially, these techniques comprise the exclusion of extreme values of communication delay from the circular queue and the smoothing of the CLB communication average delay.

For the exclusion of non-representative values of the circular queue, the extension attempts to improve the measurement of communication delay by avoiding specific transmission situations that might influence the production of extreme values. The values are generated by a lack of precision of monitoring components in interpreting and recording the collected data. The problems with the calculated delay involves the existence of communication delays with zero value, too large transmission times for a small amount of data caused by either too little information sent or the simulation application processing during the transmission. Basically, the actions for minimizing the oscillations of delay are related to dealing with such monitoring problems.

In some cases, if the amount of data transmitted is small enough to consume less time than the precision offered by the balancing system, the monitoring mechanism records the time spent for sending data as zero. In the case of the implemented balancing

system, one millisecond denotes the minimum time value detected by the local monitoring component. Consequently, the delay calculated from this transmission time is zero even though the real delay might be high. This misrepresentation leads the balancing system to incorrectly interpret the current status of the distributed system and resources. Since this technique is employed in every step of monitoring, the calculated communication delay of federates, resources, and clusters are influenced.

In order to avoid this misrepresentation of delays, an initial, simple technique is adopted. The transmission times with zero value are discarded from the recorded list since they reflect an unreal value. This value, which is generated from a lack of precision, totally differs from a real delay in any optimistic circumstance. With this action, only realistic delay values are used, providing a more representative average at the end of the calculations. Consequently, since the oscillations of the calculated average are partially caused by these discrepant values, their exclusion decreases the variations and produces a more precise representation of the recent delay.

Even with the exclusion of zero values of delay, the final average delay oscillates due to the transmission of a small amount of data. This data is large enough to produce non-zero communication delay, but its size is still small enough to make its transmission susceptible to small processing and networking time variations. In the presence of a large amount of transmitted data, these time variations are negligible since they produce insignificant changes or oscillations on the calculated communication delay. Thus, because the time variations heavily affect the transmission of a small amount of data and cannot be avoided, communication delays for these transmissions are not considered in the calculation of the federate average delay. In the proposed extended scheme, 100 bytes is the minimum amount of transmitted data to be considered in the calculation of delays, and every delay based on a quantity of data inferior to this amount is discarded from the circular queue.

The detection and exclusion of high, discrepant communication delay values is another measure adopted in the extension to decrease the communication delay oscillations. Even avoiding messages with zero transmission time and with too small of a payload, object updates that fit in a category which is considered normal might produce delays that differ totally from the rest of the delays in the data sample. These discrepant values are originated from ongoing processing that occurs simultaneously with the data

transmission and hampers the measurement of the sending time. Also, sudden spikes of communication resources consumption by other processes considerably influences the simulation communication delays. These spikes reflect on a few data transmissions, but they do not represent the general communication conditions in certain locations of the distributed environment. Because these sudden changes might happen unpredictably, the discrepant values need to be detected and eliminated from the data sample to avoid erroneous calculation of the final average communication delay. The detection is performed after the delays are calculated and when they are requested by the balancing system for redistribution analysis through the LLB. The detection and exclusion is postponed to the last moment before all elements in the circular queue are retrieved, so the analysis of the data sample for detection can identify the divergent values based on the data sample pattern. The analysis consists of using the interquartile range to determine and eliminate the outliers in the data sample. Normally, the collected communication delays tend to present a gaussian distribution for a large data sample, and the existence of outliers may influence the average delay to differ from its real normal distribution. The interquartile range presents a robust statistical method with a breakdown point of 25%, providing an efficient technique for identifying outliers for exclusion and improving the representation accuracy of the average. After the discrepant values are identified and eliminated, the average communication delay is calculated based on the middle fifty values of the sample data.

Even though the filtering of gathered delay metrics considerably decreases the sharp oscillations of average communication delay, smoothing techniques are applied on the CLB averages to diminish the inappropriate variations between them and to determine a trend. The sequence of the CLB communication delay averages is composed of a list of successive data points naturally ordered according to their time and spaced at uniform time intervals. Due to this natural temporal ordering characteristic, this sequence of delays is observed and interpreted as a time series that oscillates (decreases and increases) along with the time. Techniques for processing time series can be applied to these averages in order to produce a prediction or identify an smoothed value. One of these techniques that can be used for determining a more steady delay value based on the past oscillations is exponential smoothing. In this smoothing method, weights are assigned to each data point in the time sequence and according to time distance from the current

time. Exponential smoothing acts as an iterative simple weighed average calculation based on the previous observation and the previous smoothed average. Even though single exponential smoothing offers a fair smoothing for a time series, it does not consider trends in the data sequence. Such trends present high relevance for this smoothing analysis since they can be identified in the load oscillations of the data sample. In order to incorporate the trend of the communication delays, double exponential smoothing is employed.

The double exponential smoothing technique employed in the CLBs communication delay analysis is represented by Formulas 6.4 and 6.5. These associated formulas are used when the average is requested for the redistribution of a neighbour CLB, as presented in Algorithm 8. The local CLB, upon the request of its neighbour CLB, generates the smoothed delay average instead of the pure delay average and returns it back to the requester. Smoothing is applied among the current and the past communication delay averages calculated for the CLBs. Observing that the current value of the sequence of communication delays is used to calculate its smoothed value replacement in double exponential smoothing, a CLB does not need to save all the past data (average delays) since these are reflected in the parameters for the calculation of the last exponential smoothing.

$$sum_i = \alpha \times elem_i + (1 - \alpha) \times (sum_{i-1} + t_{i-1}), 0 \leq \alpha \leq 1 \tag{6.4}$$

- $elem_i$: current communication delay value;
- sum_{i-1}: communication delay smoothed in the previous analysis;
- t_{i-1}: trend estimation.

$$t_i = \gamma \times (sum_i - sum_{i-1}) + (1 - \gamma) \times t_{i-1}, 0 \leq \gamma \leq 1 \tag{6.5}$$

- t_{i-1}: trend estimation in the previous analysis;
- sum_i: communication delay smoothed in the current analysis;
- sum_i: communication delay smoothed in the previous analysis.

The smoothing technique is only employed at a data sample for i larger than 1 since it needs to have at least one previous element in the data sequence to support the current element. One simple method of initialization of the smoothing function is to assign the first average obtained to the first smoothed valued, so sum_0 just receives $elem_0$. Both functions are configured through two constants: the smoothing function is defined by the smoothing constant α, which works in conjunction with γ that delimits the trend estimation function. These values are chosen according to application-specific needs, and for this extension, α is set to 0.9 and γ is set to 0.5. Since α is set to a high value in the proposed extension of the balancing scheme, the initialization elements do not present much influence on the final smoothed average but they exercise certain weight to generate smoothing. For each calculation, a CLB saves the sum_i and the t_i to use for the next average request of its average.

6.1.4.2 Extension of Load Redistribution Algorithm

The redistribution algorithm also requires modifications in order to consider the existence of external background communication load in the system. In the previous algorithm, the presence of an external process might substantially influence the communication conditions for a distributed simulation, so the balancing scheme needs to consider communication delay more heavily in the redistribution. The former redistribution algorithm observes delay and uses goodput to orientate its load reallocation. In the presence of external background load, the delay based balancing scheme slowly redistributes the load by transferring the most communicative federates at the end due to the goodput constraints. However, the goodput cannot be discarded since it is the only parameter that provides a further view of the environment resources. In this case, the redistribution algorithm is extended by introducing an additional condition that attempts to improve the responsiveness of the balancing scheme to external background communication load, as described in Algorithm 9. The added condition is based on a threshold that identifies a large difference of communication delay between domains. The threshold ext_resp is a percentage of a neighbours delay and is incorporated to the neighbour CLBs communication delay for the analytical comparisons. The threshold is defined proportionally to the difference of a CLBs goodput ($ext\_resp = temp\_CLB.delay \times (goodput_{CLB} - temp\_CLB.goodput)/temp\_CLB.goodput$

Algorithm 9: Extended Redistribution Algorithm

```
1  Require: delay_CLB, delay_list, abs_comm_CLB, abs_comm_list, goodput_CLB, goodput_list, federate_list
2  while delay_CLB > max(delay_list) do
3      order_ascendingly(delay_list)
4      temp_CLB ⇐ select_first(delay_list)
5      if delay_CLB > (temp_CLB.delay+) then
6          data_comm ⇐ calculate_transfer()
7          repeat fed_eval ⇐ select_most_comm(federate_list)
8          if fed_eval.ab_comm ≤ data_comm then
9              fed ⇐ fed_eval
10         end
11         until fed ≠ ∅ federate_mig.add(fed)
12         adjust_delays(delay_CLB, temp_CLB.delay)
13     end
14     else if delay_CLB > (temp_CLB.delay + error) then
15         if goodput_CLB ≤ temp_CLB.goodput then
16             data_comm ⇐ calculate_transfer()
17             repeat fed_eval ⇐ select_most_comm(federate_list)
18             if fed_eval.ab_comm ≤ data_comm then
19                 fed ⇐ fed_eval
20             end
21             until fed ≠ ∅
22         end
23         else
24             data_comm ⇐ calculate_transfer()
25             repeat fed_eval ⇐ select_med_low(federate_list)
26             if fed_eval.ab_comm ≤ data_comm then
27                 fed ⇐ fed_eval
28             end
29             until fed ≠ ∅
30         end
31         federate_mig.add(fed)
32         adjust_delays(delay_CLB, temp_CLB.delay)
33     end
34     else
35         endLoop
36     end
37 end
38 CLB_list ⇐ select_similar_delay(delay_list)
39 order_ascendingly(CLB_list)
40 for i = 1 to CLB_list.size() do
41 if goodput_CLB > CLB_list_i.goodput then
42     data_comm ⇐ calculate_transfer()
43     fed_eval ⇐ select_lowest_feds_fit(data_comm)
44     if fed_eval.ab_comm ≤ data_comm then
45         federate_mig ⇐ select_lowest_feds_fit(data_comm)
46     end
47     federate_mig ⇐ fed_eval
48     adjust_delays(delay_CLB, temp_CLB.delay)
49 end
50 Return: federate_mig
```

when the $goodput_{CLB} > temp\_CLB.goodput$ and $ext\_resp = error$ otherwise).

6.2 Migration-Aware Load Balancing System

Migration latency directly influences the load redistribution performance since the latency dictates the frequency in which HLA federates can be moved. Load redistribution schemes mostly consider the relation between load imbalances (computation and communication) and available resources to identify the need to reallocate load. The schemes partially consider the migration latency, but they do not directly evaluate the migration delay interference into the simulation performance. Their balancing thresholds are statically set as an attempt to consider performance loss in the redistribution move when comparing the difference of load between resources. This static configuration is roughly determined through extensive experiments, which provide an imprecise value that guides the responsiveness of the balancing system.

The main objective of the proposed balancing scheme is to introduce the measurement and analysis of migration latency in the load redistribution algorithm of a distributed balancing system. Through such measurements and analysis, costs of migrating load among resources are incorporated into the decision-making of the balancing algorithms to introduce awareness of migration latency by calculating estimations of migration delays. Since real migration latency values can only be obtained after migrations are performed, estimations are needed and are based on the past simulation load migrations to enable cost analysis before migration calls are issued to modify load distribution. In order to make possible awareness of migration latencies, migration metrics are measured and monitored so they may be evaluated together with load measurements of resources and simulations. As a result, the balancing algorithms' phases are modified to accommodate migration-aware analysis in the scheme.

6.2.1 General Architecture of the Migration-Aware Balancing System

Since migration awareness is introduced on an already defined balancing algorithm, the proposed scheme is defined similarly to the architectures described in Chapter 5. As depicted in Figure 6.2, the Cluster Load Balancer (CLB) is the main element in the balancing system whereas it coordinates all the other balancing elements in a domain and conducts the main tasks of the balancing procedure. A CLB collects measured load

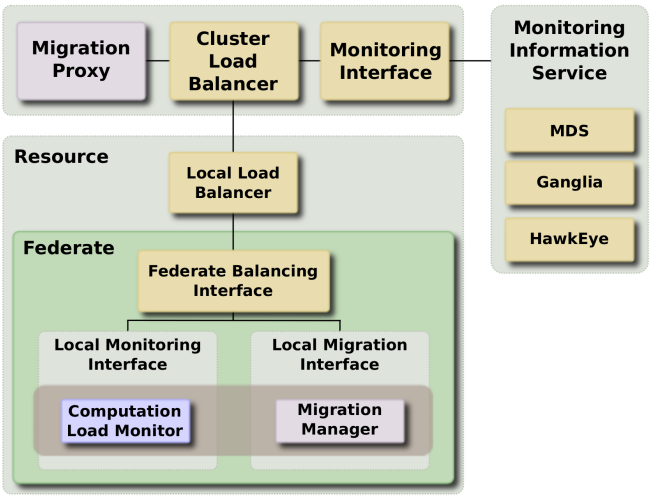


Figure 6.2: General Architecture of the Migration-Aware Load Balancing System

status data in two scopes: simulation and resource. For the gathering of information concerning resources' load status, a Monitoring Interface is accessed. The Monitoring Interface obtains the data by requesting Monitoring Information Services, which can be represented by a third-party monitoring tool that retrieves load status data from a set of resources; in this implementation of the balancing system, Grid Services [39] are used to provide such data, which consists in the processor queue length of each resource. For retrieving information related to federates' load of virtual simulations, Local Monitoring Interfaces are accessed through Local Load Balancers (LLB). Such interfaces gather and aggregate load metrics that represent the CPU consumption of federates for detecting imbalances and migration-related metrics for the awareness of migration latencies.

A LLB acts as an interface between the balancing system and virtual simulations; the interface enables the management of federates in each resource. Placed on each resource, a LLB obtains the information about each federate's load consumption, aggregates this information, and sends it to the respective CLB, which has initially placed a load status request. The data gathering performed by a LLB is achieved by accessing each Computation Load Monitor through a Local Monitoring Interface. Upon request, this Computation Load Monitor provides the amount of time a federate has used the CPU of a resource. The Local Monitoring Interface gathers this federate's load data during a balancing time interval and sends it upon request. Together with this load data, mi-

gration measurements are also collected and sent to a CLB for analysis. The LLB also forwards migration calls from the CLB to the respective federate. Such migration calls are transmitted to a Migration Manager (MM), which is instantiated for each federate.

A MM coordinates all the steps for the federate migration procedure, so the transfer of federates between resources can be conducted consistently and without loosing data. In order to perform this migration procedure efficiently and effectively, a two-phase federate migration protocol is employed in the scheme, , as described in Section 3. This federate migration technique basically divides the reallocation of a federate between resources through transfers of static information and dynamic data. For the transfer of static initialization information, a third-party tool, Grid Services, is employed in the system for transferring data reliably. For the transfer of dynamic execution state information, a peer-to-peer technique is used between MMs.

6.2.2 Balancing Algorithm of the Migration-Aware System

Essentially, the proposed migration-aware balancing scheme is divided into monitoring, redistribution, and migration phases. Such phases impose a sequential processing and determine the balancing algorithm. The division of the algorithm into the aforementioned phases simplifies the balancing problems by handling the issues from each phase separately. According to the description in Algorithm 10, the balancing scheme is triggered in cycles to enable responsiveness to load imbalances based on the current distribution of load status. In a balancing cycle, monitoring is the first phase to be conducted because detecting imbalances directly regulates the redistribution algorithm. As a scope of the proposed balancing scheme, only metrics of computational load are observed in redistribution analysis. In order to enable fast responses to imbalances, a greedy technique is employed in the redistribution phase; this problem solving heuristic-based technique provides load reallocation for a delimited set of resources, which enables a global suboptimal balancing solution. Based on the identified imbalances, the rearrangement of load is defined, and migration calls are issued to simulation federates to conduct migration procedures.

Due to its importance in enabling responsiveness to dynamic, unpredictable simulation load changes, monitoring is crucial and necessary for the redistribution algorithm. The monitoring starts with data gathering by accessing a Monitoring Infor-

Algorithm 10: Main Load Balancing Algorithm

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

mation Service through a call ($query\_MDS()$) from a CLB to its respective Monitoring Interface. Upon receiving information about the resources' and simulation's load status requested from the monitoring services, data filtering is employed to eliminate non-managed resources or overloaded resources that do not contain any simulation element: i.e., a resource that cannot have its load lowered. Also, normalization of load values ($normalize\_loads(current\_loads, benchmark)$) is applied to solve the inherent issues of resources' heterogeneity by using benchmarks that define computational capacities. Based on the selection of overloaded resources ($select\_overload(current\_loads)$) in the gathered data sample, the balancing system requests more detailed load data ($request\_LLBs(overload\_cand)$) through each LLB: the CPU consumption of each federate, which is used to locally rank federates according to their load. According to the responses from the LLBs, overloaded resources without simulation entities, federates, are then identified and disregarded ($filter(current\_loads, spec\_loads)$). All the collected information is relevant to load status measurements; migration-related metrics also need to be provided by each LLB, which provides these metrics together with the simulation load data. The migration-related metrics are gathered to enable the analysis of migration latency when redistributing simulation load. The migration metrics measured and gathered consist of migration delay, migration distance, and federate state size. Mi-

Algorithm 11: Inter-Domain Redistribution Algorithm

```
1  Require: neighbours_data
2  neighbours ⇐ identify_Neighbour_Less_Load()
3  order_neighbours_by_load(selectionParameter)
4  if neigbours! = ∅ then
5      neighbours ⇐ select_Neighbours(extStD, localStD)
6      order_neighbours_by_load(selectionParameter)
7  end
8  foreach neighbour IN neighbours do
9      overloaded_RSCs ⇐ select(neighbour)
10     federates ⇐ select(spec_loads, overloaded)
11 end
12 if overloaded_RCSs ≠ ∅ then
13     sort_list_load(overloaded_RSCs)
14     sort_list_load(neighbour_RSCs)
15     moves ⇐ redistribute(overloaded_RSCs, neighbour_RSCs)
16     moves ⇐ analyze_migration_latency(moves)
17 end
18 send_migration_moves(moves)
19 adjust_factor(relFactor, overloaded_RSCs, moves)
```

gration delay mostly represents the time in milliseconds spent to transmit a federate's execution-state data and incoming messages. Migration distance contains the topological distance, as well as the communication latency, between the two resources involved with the migration of a federate. Federate state size provides the amount of data transmitted through peer-to-peer data transfer. A migration latency estimation is then calculated based on these three metrics.

The detection/redistribution phase of the balancing scheme is triggered just after all the monitoring data is retrieved and gathered in a CLB, as detailed in Algorithms 10 and 11. The balancing algorithm is composed of local and inter-domain scopes, and the inter-domain load reallocation is triggered based on migrations generated in the local scope of the redistribution algorithm. As a premise for the balancing algorithm, the local load rearrangement is applied as a means of diminishing the load discrepancies of resources delimited in a domain (cluster). In the load redistribution for local scope, the gathered information is analyzed by identifying differences on load distribution based on a calculated average load ($calculate\_mean\_bds(mng\_loads)$). The analysis comprises ordering the resources according to their load and matching pairs of them (overloaded and underloaded) in order to achieve a close-to-even distribution of load in a cluster of resources ($select(mng\_loads, mean, bds)$), as described in Algorithm 12. Based on this selection of resources, federate migrations are determined and analyzed according to their estimated delays.

Algorithm 12: Pair-Match Evaluation Algorithm

```
1  Require: src_rsc, dst_rsc
2  selected_federate ⇐ select_federate_smallestLatency(src_rsc)
3  if dst_rsc < min then
4      if number_fed(src_rsc) ≥ 1 & src_rsc > (min * φ) then
5          Δt' ⇐ Δt × α
6          create_migration_move(src_rsc, dst_rsc, selected_federate)
7      else if number_fed(src_rsc) > 1 then
8          Δt' ⇐ Δt
9          create_migration_move(src_rsc, dst_rsc, selected_federate)
10     end
11 else if (dst_rsc − src_rsc) > (min * δ) then
12     if number_fed(src_rsc) ≥ 1 AND (dst_rsc − src_rsc) > (min * φ) then
13         Δt' ⇐ Δt × α
14         create_migration_move(src_rsc, dst_rsc, selected_federate)
15     else if number_fed(src_rsc) > 1 then
16         Δt' ⇐ Δt
17         create_migration_move(src_rsc, dst_rsc, selected_federate)
18     end
19 end
20 if migrationMove then
21     estimatedGain ⇐ estimateMigGain(dst_rsc, src_rsc, Δt')
22     estMigTime ⇐ estMigTime(dst_rsc, src_rsc, selected_federate)
23     Return: migrationMove, estimatedGain, estimatedMigTime
24 end
```

The inter-domain load redistribution is conducted when modifications of local distribution are not produced ($mig\_moves = \emptyset$) or a set of domains (clusters) show substantial load imbalances, evidencing the continuous need of simulation load's transfers ($relFactor \geq random\_number(1, 100)$). The inter-domain imbalances are defined by the distribution factor ($relFactor$), which represents the ratio between the total number of inter-domain migrations and the total number of selected overload resources in the previous balancing cycle for a local CLB; this evidences the degree of imbalance between a local domain and its neighbours. The $relFactor$ contains a value that ranges between 0 and 1, which then is multiplied by 100 in order to be compared with the randomly generated number. Constant triggering of inter-domain redistribution is prevented in the balancing scheme since the inter-domain redistribution requires costly data gathering, and a low distribution factor in the past balancing cycle might incapacitate the detection of current substantial imbalances through a biased threshold. Therefore, a random, uniformly distributed number is used to add uncertainty to the analysis for the need of inter-domain balancing since load can oscillate dynamically.

In summary, the redistribution algorithm in both scopes act similarly; the difference concerns the inter-domain scope, which requires load status data from the re-

sources in the neighbour domains for the detection of imbalances and load redistribution ($request\_Neighbour\_Load\_Data()$). The retrieved inter-domain load information is used to detect large load differences between a local domain and its neighbours, evidencing the largest imbalance and consequently producing the largest number of migrations. As delineated in Algorithm 11, the underloaded domains are first selected ($identify\_Neighbour\_Less\_Load()$) and then ordered ($order\_neighbours\_by\_load$ ($selectionParameter$)) for the next step in the load analysis if any underloaded domain is detected. In iterations, a set of underloaded neighbour CLBs is selected, and local overloaded resources are determined ($overloaded\_RSCs \Leftarrow select(neighbour)$) according to the redistribution algorithm described in 11. The redistribution algorithms in both scopes present lists of resources organized in descending order based on the load, and the most overloaded resources are selected first to perform pair-match evaluations with the most underloaded resources ($redistribute(overloaded\_RSCs, neighbour\_RSCs)$). This matching continues subsequently in the list until all overloaded resources are assigned to a migration move or if the difference of load between overloaded and underloaded resources does not reach the threshold for creating a migration move. After federate migrations are determined, they are analyzed according to the migration latency they can generate in the simulation execution ($analyze\_migration\_latency(moves)$).

As detailed in Algorithm 12, the pair-match algorithm compares the load of an overloaded resource and an underloaded resource. Such a comparison consists in detecting the difference of load between the two resources and analyzing it based on thresholds ($min * \phi$, $min * \delta$, and $number\_fed(src\_rsc)$): if the difference exceeds a certain value, a migration move is generated. In order to enable migration-aware analysis, estimations are defined for the migration moves that are created. For each pair of resources and a selected federate, the migration delay ($estMigTime()$) and the performance gain ($estimateMigGain()$) are determined. Based on its load and migration time, a federate from the overloaded resource is selected for the migration move. The measurements of its last migration move are used to produce an estimation, which is evaluated later in the migration filtering. For creating the migration move, the federate with the lowest load consumption is selected for migration if it is eligible. A federate is considered eligible for migration only if it does not have any restriction for running on a specific resource, such as interfacing the rest of a distributed simulation to a user. The estimations use

the past migration distance, migration delay, and federate state size to define the current migration delay. As a result, the migration moves are determined, and migration delays are analyzed and compared to obtain the candidates that most benefit simulation performance.

After all possible pair-matches are analyzed, a list of migration moves is obtained. Migration latency and performance gain estimations are defined for each migration move in such a list based on simple rules of proportionality. The migration delay estimation (t_e) of a federate is calculated with the migration latency and the distance obtained from the last migration move conducted for a federate, as described in Formula 6.6. Based on the distance between the source and destination resources in the migration move, latency is proportionally estimated with the past and current distances. Similarly to migration latency, the performance gain is defined through Formula 6.7. Essentially, the gain (t_d) is estimated in time (milliseconds) for the purpose of comparison, and it is computed according to a relation between the $load_{src}$ and the $load_{dst}$ resources. The difference of computational load between such resources provides the performance gain in matters of load, which is then factorized into time when multiplied to Δt.

$$t_e = \frac{time_{mig} \times dist_{mig}}{dist_{dst}} \tag{6.6}$$

- t_e: estimation of migration delay;
- $time_{mig}$: time of past migration;
- $dist_{mig}$: distance of past migration;
- $dist_{dst}$: distance of estimated migration.

$$t_d = \frac{\Delta t \times (load_{src} - min_{load} - load_{dst})}{load_{src}} \tag{6.7}$$

- t_d: estimation of time gain;
- $load_{src}$: current load of source resource;
- min_{load}: estimated minimum load;
- $load_{dst}$: current load of destination resource.

Δt is employed as a time factor that delimits the base for estimating the improvement in performance for a migration move. This factor is introduced in the calculation since a more consistent time value truly representing the virtual simulation time cannot be used in the balancing scheme. When keeping the balancing system transparent for distributed simulations and consequently minimizing modifications on simulations' design framework, there is no technique or method that can provide a value that represents the time that a simulation runs. There is no prediction mechanism that can be used by the balancing system to obtain a simulation execution time because simulations can be designed for different purposes and objectives, and, hence, with different, unpredictable simulation execution times. Therefore, the balancing cycle interval is employed in the estimation calculations as the base parameter to define the performance gain in a short period of time whereas no value can be used in the balancing system to represent the total real simulation time.

With the list of migration moves containing the performance gain and migration latency estimations, the redistribution algorithm identifies the migrations that can really provide performance improvement for distributed simulations. In this case, improvement is basically accomplished by allowing migrations that produce a time gain larger than the migration delay. As defined in Algorithm 13, the filtering procedure is applied on the whole set of migration moves in each scope (local or inter-domain). Comparisons are conducted between the accumulated final estimations of performance gain and migration latency. For these accumulated estimations, it is assumed that the final migration latency is determined by the largest time and the performance gain by the sum of time gains, as described in Formula 6.8. For latency calculation, the largest value produces most of the delay added to simulations, and other values have a slight impact on the final result without majorly contributing to it, as detailed in Formulas 6.9 and 6.10. A simple comparison is performed between such accumulated estimations: performance gain ($timeGain$) and migration delay ($expectedMigrationDelay$). Filtering is repeatedly applied while gain is smaller than delay. The filtering is comprised of removing the migration move with the lowest gain from the list of migration moves ($eliminate\_smallest\_gain(mig\_moves)$). In each iteration, the accumulated gain and latency are calculated and compared until the accumulated gain shows performance improvement on the simulation time.

$$t_{ds} = \sum_{i}^{n} t_{d_i} \tag{6.8}$$

- t_{ds}: overall sum of migration time gains;
- t_{d_i}: time gain of each migration move.

$$t_{es} = t_{e_{largest}} + \frac{\alpha \times \sum_{i \neq e_{largest}}^{n} t_{e_i}}{(n-1)} \tag{6.9}$$

- t_{es}: overall estimation of migration delays;
- $t_{e_{largest}}$: largest migration delays in the set of migration candidates;
- α: influence of other migration delays on overall estimation;
- t_{e_i}: delay of a migration move candidate;
- $e_{largest}$: migration move with the largest delay.

$$\alpha = \frac{2 \times \bar{t_e} - 1 + t_{e_{largest}}}{2 \times t_{e_{largest}}} * (k+1) \tag{6.10}$$

- α: influence of other migration delays on overall estimation;
- $\bar{t_e}$: the mean of migration delay;
- $t_{e_{largest}}$: largest migration delay;
- k: number of migrations with delay larger than mean.

At the end of the migration-aware analysis (filtering), the resulting list of migration moves are forwarded to their respective resource to advance with the migration calls, as shown in Algorithm 10 and 11. Such calls are then issued to their respective MM in the LLB. As soon as the migration procedure finishes in the remote resource by restoring the migrating federate's execution state, new migration metrics are measured and registered in the balancing system to provide up-to-date measurements and improve the redistribution decision making in the next balancing cycles. In

Algorithm 13: Migration Latency Filtering Algorithm

```
1  Require: mig_moves
2  if mig_moves! = ∅ then
3      timeGain ⇐ calculateGain(mig_moves)
4      expectedMigrationDelay ← calculate(mig_moves)
5      while timeGain ≤ expectedMigrationDelay do
6          mig_moves ⇐ eliminate_smallest_gain(mig_moves)
7          timeGain ⇐ calculateGain(mig_moves)
8          expectedMigrationDelay ⇐ calculate(mig_moves)
9      end
10 end
11 Return: mig_moves
```

the inter-domain scope, after the migrations are issued to their respective source resources, adjustments are calculated to redefine $relFactor$ for the next balancing cycle ($adjust\_factor(relFactor, overloaded\_RSCs, moves)$).

6.2.3 Extension of the Migration-Aware Balancing System

The proposed migration-aware balancing scheme is able to avoid costly migrations and to not jeopardize simulations' performance through modifications in the load distribution. However, the proposed scheme presents some drawbacks and requires modifications to overcome some issues and improve the load redistribution. The needed modifications are incorporated as an extension to the current scheme. Under observation of the balancing behaviour, the extension attempts to mitigate the absence of migration history for simulation federates, to overcome the reduction in the number of redistribution possibilities after migration latency filtering is applied, to use the parallelism of load redistribution as an aspect to improve the filtering, and to reformulate the method used to analyze migration delays in filtering of migration moves.

6.2.3.1 Absence of Migration History

As delimited in the migration-aware balancing scheme, the migration information of each federate is kept locally in the Local Monitoring Interface of each LLB. Since an interface is instantiated for each federate and runs simultaneously with it in order to provide information related to a federate, the interface needs to store all the information locally and to provide it upon request of the LLB. Thus, a federate's migration information exists only if the federate has been migrated in any moment of its past execution time.

For initial execution of simulations, there is no migration data that can be used to filter and to determine current migration moves. Even for simulations that have advanced execution time, this migration information might be scarce if simulations are composed of a large number of federates when compared to the number of resources, or if load imbalances are not detected so often in the distributed system.

Since information of the previous migration process is vital for the migration-aware analysis, the scarcity of this information leads the balancing system to ignore migration delays when redistributing load. In this case, for every simulation and without exception, there exists a period of time in which the load balancing is not aware of the federate's migration delays. This situation occurs due to the absence of migration history, which feeds the proposed balancing system and allows it to react properly, based on each previous migration latency. Because of its importance, the lack of migration information needs to be prevented or minimized. Therefore, measurements of other federate's migration processes are used to calculate estimations and improve the filtering of load redistribution.

The most recent migration delay or the average of N migration delays of other federates can be used to determine and define a federate's unknown migration time. The most recent delay provides metrics that evidence the most accurate data about the conditions and load status of the communication resources for migration purposes. This technique also enables the matching of federates with similar migration distance, decreasing the discrepancies when assigning other migration delays to a federate. On the other hand, even though the technique increases the chances of existing migration data for the analysis, a small number of migrations still might lead to an absence of migration history. The average of delays is based on a set of federate's migration time values, so it does not represent the most recent measurement of resource conditions for federate migrations since the average might contain old samples. However, the absence of measurements is minimized with this approach since the calculation of the average requires the existence of only one migration time in the set of federates that is considered in the average calculation. The average also provides a value that represents the measurements of a set of federate migrations in the data sample, resulting in a more generic characteristic that can be applied to the estimation of a federate without any migration history.

With the utilization of migration averages, the set of N federates needs to be defined to feed the calculation of the average. In the distributed balancing system, the LLBs

aggregate some federates in a resource and can store the average migration information for the federates that run locally in the resource. A CLB can also aggregate all the migration information of federates that are running in a cluster of resources. The latter approach, aggregation of an average in a CLB, avoids the burden of transmitting averages from LLBs to a CLB for redistribution filtering and the introduction of more complexity into the redistribution algorithm. Therefore, a migration latency average is obtained based on the migration moves generated in the last balancing cycle and stored in the CLB; as soon as a federate migration procedure finishes, the migration information is transmitted to the CLBs involved with the particular migration process.

A set of federate migrations might contain load transfers with different characteristics based on transfer distance, execution state size, and time aspects. Since transfer distance and execution state size directly influence the migration time, they need to be considered in the estimation, as well as in the calculation of averages. Same or similar migration end-points are needed to calculate the estimation that best predicts the current migration of a federate. Since this condition leads to a scarce migration history due to the difficulty in matching migrations with the same source and destination resources, it cannot be used. Even similar migration distances between end-points can restrict the matching of a previous federate migration procedure to the characteristics of the current prediction of a migration move. As a result, as a matter of simplification for comparison and for estimation purposes, averages are then calculated for distance, state size, and migration delay; thus, the average distance and average state size are used to directly and proportionally obtain a migration delay.

6.2.3.2 Reduced Number of Redistribution Possibilities

In the migration delay analysis of the proposed scheme, the filtering of migration candidates is performed based on the set of migrations identified in the load redistribution algorithm. This filtering prevents costly migrations that might introduce a delay larger than the time gain produced by the simulation load rearrangement. However, the filtering might also prevent the balancing system from identifying beneficial migration moves. The initial migration moves are selected based on the most suitable resource, and federate candidates are selected based on a load analysis; after some migrations are excluded from the list, there might be other pair-matches or other federates that can still provide

Algorithm 14: Extended Main Load Balancing

```
1  while TRUE do
2      loads ⇐ query_MDS()
3      current_loads ⇐ filter_MDS_data(loads)
4      current_loads ⇐ normalize_loads(current_loads, benchmark)
5      overload_cand ⇐ select_overload(current_loads)
6      spec_loads ⇐ request_LLBs(overload_cand)
7      mng_loads ⇐ filter(current_loads, spec_loads)
8      mean, bds ⇐ calculate_mean_bds(mng_loads)
9      over, under ⇐ select(mng_loads, mean, bds)
10     mig_moves ⇐ redistribute_local(mng_loads)
11     while mig_moves ≠ ∅ do
12         moves ⇐ analyze_migration_latency(mig_moves)
13         moves_excluded ⇐ determine_moves_excluded()
14         add_RSCs_to_list(overloaded_RSCs, moves_excluded)
15         add_RSCs_to_list(neighbour_RSCs, moves_excluded)
16         sort_list_load(over)
17         sort_list_load(under)
18         moves ⇐ redistribute(overloaded_RSCs, neighbour_RSCs, moves_excluded)
19     end
20     mig_moves ⇐ analyze_migration_latency(mig_moves)
21     send_migration_moves(mig_moves)
22     if mig_moves = ∅ then
23         data_neighbours ⇐ request_Neighbour_Load_Data()
24     else
25         if relFactor ≥ random_number(1, 100) then
26             data_neighbours ⇐ request_Neighbour_Load_Data()
27         else
28             data_neighbours ⇐ ∅
29         end
30     end
31     wait( Δt )
32 end
```

performance gain with load transfers.

In order to enable more migration moves, an iterative load redistribution and migration filtering is introduce in the scheme. Consequently, the redistribution algorithm is executed while there still exists possibilities of modifying the simulation load distribution, and the iterations stop when there are no more overloaded resources or there are no more possible load changes (underloaded resources that can receive a federate). For enabling this iterative load redistribution, the existent proposed scheme needs to be modified. The redistribution algorithms, as shown in Algorithm 10 for local scope and in Algorithm 11 for inter-domain scope, are extended to accommodate the iterative load reorganization and filtering, as described in Algorithm 14 and Algorithm 15.

In the non-extended redistribution algorithm, as soon as a migration move is generated, the resources involved with the move are removed from, or unchecked in, the list of resources; thus, they are not used for the next pair match selection. Both

Algorithm 15: Extended Inter-Domain Redistribution

```
1  Require: neighbours_data
2  neighbours ⇐ identify_Neighbour_Less_Load()
3  order_neighbours_by_load(selectionParameter)
4  if neigbours = ∅ then
5      neighbours ⇐ select_Neighbours(extStD, localStD)
6      order_neighbours_by_load(selectionParameter)
7  end
8  foreach neighbour IN neighbours do
9      overloaded_RSCs ⇐ select(neighbour)
10     federates ⇐ select(spec_loads, overloaded)
11 end
12 if overloaded_RCSs ≠ ∅ then
13     sort_list_load(overloaded_RSCs)
14     sort_list_load(neighbour_RSCs)
15     moves ⇐ redistribute(overloaded_RSCs, neighbour_RSCs)
16     while moves ≠ ∅ do
17         final_moves ⇐ analyze_migration_latency(moves)
18         moves_excluded ⇐ determine_moves_excluded()
19         add_RSCs_to_list(overloaded_RSCs, moves_excluded)
20         add_RSCs_to_list(neighbour_RSCs, moves_excluded)
21         sort_list_load(overloaded_RSCs)
22         sort_list_load(neighbour_RSCs)
23         moves ⇐ redistribute(overloaded_RSCs, neighbour_RSCs, moves_excluded)
24     end
25 end
26 send_migration_moves(final_moves)
27 adjust_factor(relFactor, overloaded_RSCs, moves)
```

extended redistribution algorithms continue performing this same technique; however, they re-insert the resources that were in the migration moves and were rejected in the filtering (*add_RSCs_to_list* (*overloaded_RSCs*, *moves_excluded*) and *add_RSCs_to_list* (*neighbour_RSCs*, *moves_excluded*)) into the list of resources (overload and underloaded). As these resources are inserted in the lists, the lists are reorganized (*sort_list_load* (*overloaded_RSCs*) and *sort_list_load* (*neighbour_RSCs*)), and the load redistribution is executed again to identify more needed modifications on the load distribution, reconsidering the resources that were rejected in the previous iteration. This migration filtering and load redistribution is conducted in iterations while no more federate migration moves are generated ($moves \neq \emptyset$).

However, adding resources for redistribution analysis after the filtering might lead the matching process to an endless loop. It is most likely to match the same pair of resources for the purpose of performing the load comparison, and the selected migration move is then rejected in filtering. Thus, when observing the extension on algorithms, another modification is realized, adding the excluded migration moves for the redistribution analysis. During the evaluation of resources before pair-matches are conducted, the

resources' conditions are compared with the respective migration moves generated and rejected in the past iterations. The comparison consists in first determining if there is any other federate with a smaller migration delay than the federates selected in the past migration moves for the same distance. Federates are likely to present different execution state sizes and consequently different estimated migration times for the same migration distance. Thus, if there is any other federate that matches this condition, another attempt to create a migration move with the same pair resources is realized. On the other hand, if there is no other federate with a smaller migration delay, other underloaded resources are searched for and selected to match an overloaded resource following the redistribution procedure.

The load rearrangement and filtering iterations are finalized when there is no migration move produced by the redistribution algorithm. This attempt to perform additional searches for resources' pair-matches might not produce considerable improvement on simulation execution performance, but it can improve the balancing responsiveness to load imbalances.

6.2.3.3 Migration Parallelism of Load Redistribution

As described in Formulas 6.8 and 6.9, the migration moves defined in a balancing cycle tend to be performed in parallel since they are launched mostly at the same time and last for a certain time interval (migration delay). Because these migrations occur in a semi-parallel trend, the effect on the simulation execution time is calculated based on Formula 6.9, and the gain is estimated in Formula 6.8 as the sum of every migrating federate's execution time improvement. Nevertheless, the analysis based on these two estimations, migration delay and performance gain, leads to the production of migration moves that generate performance loss in a simulation. Since the overall gain estimation is obtained through a sum, the analysis wrongly allows the execution of more migrations than are necessary for the achievement of a gain. Therefore, the estimation of overall performance gain needs to be redefined by considering that just a sum as estimation cannot represent the real gain of a load redistribution.

Since the gain affects the simulation performance by speeding up the processing of a federate's tasks, the smallest gain determines most of the gain generated by the migration moves defined in the load rearrangement algorithm. The smallest gain corresponds to

the federate that is predicted to present the slowest processing, based on the difference of loads. Because a federate's load cannot be precisely predicted due to its dynamic execution characteristics, it is assumed that a migrating federate's execution acts according to information obtained through comparing the load of its hosting resource with that of the resource designated for migration. As a result, as described in Formula 6.11, the smallest performance gain estimation ($t_{d_{smallest}}$) is used to calculate the overall gain (t_{ds}). The gain of other federate migrations are considered in the calculation, but they have less influence on the final overall estimation value.

$$t_{ds} = t_{d_{smallest}} + \frac{\beta \times \sum_{i \neq d_{smallest}}^{n} t_{d_i}}{(n-1)} \tag{6.11}$$

- t_{ds}: overall time gain;
- $t_{d_{smallest}}$: smallest time gain in migration moves;
- β: influence of other time gains on overall estimation;
- $d_{smallest}$: migration move with the smallest time gain;
- t_{d_i}: time gain of a migration move.

$$\beta = \frac{2 \times \bar{t_d} - 1 + t_{d_{largest}}}{2 \times t_{d_{largest}}} * (k+1) \tag{6.12}$$

- β: influence of other time gains on mean;
- $\bar{t_d}$: the mean of time gain;
- $t_{d_{largest}}$: smallest time gain in migration moves;
- $t_{d_{largest}}$: largest time gain in migration moves;
- k: number of time gains larger than the mean.

As defined in Formula 6.12, β restricts the influence of the average gain in a set of migrations. Similarly to α in Formula 6.10, β consists in representing the influence that the performance gains exercise on the simulation execution time; the value of β

is determined proportionally to the amount of gain close to $t_{d_{largest}}$. k in the formula represents the number of time gains with value over $t_{d_{largest}}/2$. Even though, theoretically, the gain is limited by the smallest performance gain, the overall gain might be influenced by other factors, or it may present a different real final gain after the migrating federates resume their execution. Since all analysis is based on estimations, which might present final effects on simulations different from the ones that are expected, the final load redistribution gain also includes the estimations of other migrations. Consequently, other gains might also present a slight influence on the simulation performance; this influence is defined and represented by β. Based on this assumption, less migration moves are enabled and a more conservative responsiveness, regarding the awareness of migration latencies, is introduced.

6.2.3.4 Reformulation of Migration Analysis

As defined in Formula 6.7, the proposed analysis employs a static time interval (Δt) as a base for the gain estimation. This time interval is used to evaluate the migration moves because of the difficulty in predicting the period of time required for the conclusion of a simulation execution: the simulation can run for a long period of time or can end its execution in the next simulation time step. Consequently, the fixed parameter is defined as a base of comparison between migration delays and performance gain, but this value might not really represent the time that can be used to compare with delays. Moreover, the formula used in the analysis can be redefined to represent performance in time with improved accuracy.

In Formula 6.7, the design is defined by a rule of proportion to provide a gain when compared to the execution status of a federate placed on an overloaded resource. Consequently, the difference of load between a pair of resources dictates the amount of gain based on the time spent to produce the work for Δt with load on the source resource. The calculation based on proportionality provides a reasonable solution for obtaining a time value out of Δt, but load is used as the major aspect to perform the calculation. On the other hand, gain can be represented by the work that can be produced with the available resources and time ($w = t \times r$), in which available resources is inversely proportional to the load of a CPU. Therefore, as delineated in Formula 6.13, t_d is also obtained through a rule of proportion, but it is based on the difference of available resources between an

overloaded resource and an underloaded resource.

$$t_d = \frac{\Delta t \times (load_{src} - min_{load} - load_{dst})}{2 \times load_{dst}} \tag{6.13}$$

- t_d: estimated delay of a migration move;
- Δt: balancing time interval;
- $load_{src}$: load of source resource;
- min_{load}: estimated minimum load;
- $load_{dst}$: load of destination resource.

The use of a static, predefined time interval in the formula might mislead the migration delay analysis. The analysis might allow costly migration moves or block useful, fast migration procedures when compared to the total remaining simulation time. Thus, as an attempt to introduce some dynamic aspects on the time interval used in the decision-making of the migration delay analysis, the value of Δt is modified in certain circumstances to enable the production of migration moves that are considered costly in the analysis but possibly useful for the simulation performance.

In each balancing cycle, after migration moves are filtered, the value of Δt is incremented based on the ratio between the number of generated migration moves and the total number of source resources ($r = moves/RSCs$). With this approach, the balancing system might enable useful costly migration moves by introducing some tolerance to migration delays. The ratio represents the need to redistribute load for the distributed system and provides an assessment to identify and measure the amount of tolerance the balancing system can allow through the increase of Δt value. This adjustment on the time interval is performed only if there is no filtered migration move, and the approach is executed in iterations. The incremented time interval ($\Delta t'$) is obtained by adding up to 50% of its value based on the ratio ($\Delta t' = \Delta t + r \times \Delta t/2$). Such increments are applied until any migration move is in the filtered list. In summary, with this technique migration moves are blocked until the cost/benefit ratio reaches a threshold that enables it to pass through the migration delay filtering. Thus, even migrations considered as costly by the proposed scheme are allowed to be conducted; this is undergone with the expectation that the move is beneficial to the simulation performance.

6.3 Self-Adaptive Load Balancing System

As described in Chapter 5, the distributed balancing system attempts to decentralize the re-partitioning of simulation load by introducing a hierarchical, distributed architecture. In this architecture, the local balancing is performed as in Section 4.3.1 by collecting data individually and processing it in a central element. However, for inter-domain balancing, the scale of the system creates obstacles for existing only one central element that produces all simulation load redistribution. Thus, the redistribution phase in the balancing scheme is performed in a distributed manner, allowing more independence of balancing elements, avoiding synchronization, bottlenecks, and overheads, and providing fault tolerance. Based on this distributed scheme, extra components are added to introduce self-adaptation, and slight modifications are applied in the balancing algorithms.

As depicted in Figure 6.3, the Cluster Load Balancer (CLB) organizes all the process of the balancing scheme by triggering data gathering, detecting imbalances, reorganizing the load, and performing migration moves. In order to collect information about the resources and federates, the CLB accesses Monitoring Interfaces and its Local Load Balancers (LLBs). A Monitoring Interface facilitates data request to Monitoring Information Services, which is provided through Grid services. Relying the monitoring of resources on Grid computing, which includes a resource sharing system that organizes of resources, individuals, and institutions [40], Globus Toolkit [1] is used for providing Grid MDS services. The collected information comprises the processing queue of resources. Before requested data is delivered, filtering is employed to remove data that misleads the detection of imbalances.

A LLB is placed in every resource in order to intermediate requests of load information and migration calls from a CLB. The requested information regards federates' CPU consumption, and this information is collected from each federate by accessing the Federate Balancing Interfaces (FBI) through the Local Monitor Interface (LMol). A LLB also dispatches migration calls to their respective Migration Manager (MM). Likewise the LMol, an instance MM works for each federate in a resource.

As introduced in Section 3, the federate migration is performed in two phases in order to minimize the migration latency. With assistance of Grid services, the first part of this two-phase migration transfers configuration files and initiates the federate remotely. The second part of the migration procedure sends the federate's execution

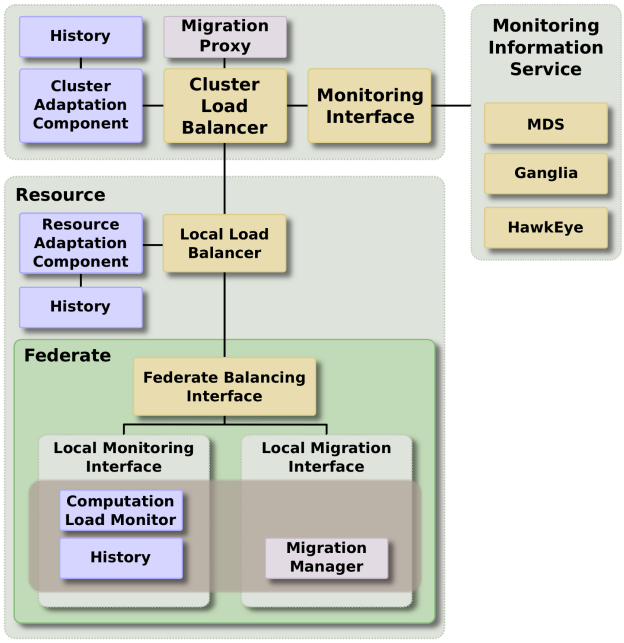


Figure 6.3: Self-Adaptive Balancing Scheme's Architecture

state and its incoming messages through peer-to-peer data transfers, and a Migration Proxy intermediates the data transmissions for migrations performed between remote parts. Adaptation components are added in the load balancing's architecture to observe the balancing actions. Every incoming or outgoing migration is registered by History component, which are placed in federates, resources, and CLBs. These elements store migration frequency data, and they contain a queue that is restricted to a certain size (H). Cluster Adaptation Component (CAC) works together with the CLB by collecting migration histories for determining the proper adjustments. A Resource Adaptation Component (RAC) is placed in each resource, aggregates migration history from the local resource History component, and produces a migration ratio and federate migration frequencies.

6.3.1 Redistribution Algorithm of the Self-Adaptive Balancing System

As detailed in Algorithm 16, the load redistribution is performed in inter-domain and local scopes by a CLB. For both scopes, monitoring data is collected from the resources and federates. This data is filtered and processed. Then, the algorithm first realizes a local load re-partitioning by detecting load imbalances and identifying the needed load changes. The load changes are created through local pair-match evaluations between resources based on a minimum resource load (min'), as described in Algorithm 17. In this algorithm, the difference of load between two resources and the number of federates in each of them is used to generate a federate migration.

After the local migrations are determined, a inter-domain load redistribution is performed based on its success rate (number of migrations versus number of migration candidates) in the previous balancing cycle. For this redistribution, a set of overloaded resources is selected according to $selectionParam'$, which is obtained from a calculation between the cluster load average and the neighbour cluster load average. Upon receiving the resources candidates for redistribution, the neighbour CLB evaluates them as delineated in Algorithm 18. The algorithm attempts to match each external overloaded resources with its internal underloaded resources. This inter-domain pair-match procedure is based on a minimum resource load (min') that determines if a federate (load) can be transferred to the underloaded resource. The three described thresholds regulate the balancing responsiveness towards load imbalances, so they receive adjustments in every balancing cycle according to the actions resulted from the adaptation technique, which is described in the next section.

6.3.2 Self-Adaptation Technique in the Balancing Scheme

Similarly to the reactive balancing scheme, the self-adaptation is provided in three phases: data collection, analysis, and execution of adjustments. The adaptation occurs in each cluster of resources managed by a CLB, without considering the adaptation performed in other CLBs. In order for the balancing system to self-adapt, the scheme needs to collect information that reflects the consequences of its re-distributions. Consequently, the recent migration past is analyzed to adjust the balancing thresholds. The proposed

Algorithm 16: Distributed Dynamic Load Balancing Algorithm

```
1  while TRUE do
2      loads ⇐ query_MDS()
3      current_loads ⇐ filter_MDS_data(loads)
4      current_loads ⇐ normalize_loads(current_loads, benchmark)
5      overload_cand ⇐ select_overload(current_loads)
6      spec_loads ⇐ request_LLBs(overload_cand)
7      mng_loads ⇐ filter(current_loads, spec_loads)
8      mean, bds ⇐ calculate_mean_bds(mng_loads)
9      over, under ⇐ select(mng_loads, mean, bds)
10     mig_moves ⇐ redistribute_local(mng_loads)
11     send_migration_moves(mig_moves)
12     if mig_moves = ∅ then
13         data_neighbours ⇐ request_Neighbour_Load_Data()
14     end
15     else
16         if relFactor ≥ random_number(1, 100) then
17             send_migration_moves_to_caller(migration_moves)
18         end
19         else
20             data_neighbours ⇐ ∅
21         end
22     end
23     neighbours ⇐ identify_Neighbour_Less_Load()
24     if neigbours! = ∅ then
25         overloaded_resources ⇐ select(firstNeighbour)
26         federates ⇐ select(spec_loads, overloaded)
27     end
28     else
29         overloaded_resources ⇐ filter_resources(extStD, localStD)
30         selectionParam′ ⇐ selectionParam ∗ selectionAdj
31         overloaded_resources ⇐ order_resources(selectionParam)
32         federates ⇐ select(spec_loads, overloaded)
33     end
34     send_to_neighbour(overloaded_resources, federates)
35     migration_moves ⇐ wait_for_migration_moves()
36     send_migration_moves(migration_moves)
37     adjust(relFactor, overloaded_resources, migration_moves)
38     adaptation()
39     wait( Δ )
40 end
```

adaptation technique employs number of migrations and time as the main metrics for adjusting the balancing system. The number of migrations is registered according to the time in which they occur, and they are restricted to a history (H). The history is used to limit the amount of migration information that is stored and exchanged among the adaptation components; it also determines the relevance of the recent migration moves when compared with older moves, providing means to detect migration cycles according to history sizes.

Triggered at the end of each balancing cycle, the adaptation components collect federates' migration frequency, resources' migration ratio, and a CLB's migration ratio. The

Algorithm 17: Local Pair-Match Evaluation Algorithm

```
1  Require: src_rsc, dst_rsc
2  min' ⇐ min * minLoadAdj
3  if dst_rsc < min' then
4       if number_fed(src_rsc) > 1 then
5            create_migration_move(src_rsc, dst_rsc)
6       end
7       else if number_fed(src_rsc) = 1 & src_rsc > (min' * φ) then
8            create_migration_move(src_rsc, dst_rsc)
9       end
10 end
11 else if (dst_rsc − src_rsc) > (min' * δ) then
12      if number_fed(src_rsc) > 1 then
13           create_migration_move(src_rsc, dst_rsc)
14      end
15      else if number_fed(src_rsc) = 1 & (dst_rsc − src_rsc) > (min * φ) then
16           create_migration_move(src_rsc, dst_rsc)
17      end
18 end
```

Algorithm 18: Inter-Domain Pair-Match Evaluation Algorithm

```
1  Require: int_rsc, ext_rsc
2  min' ⇐ min * acceptanceAdj
3  if int_rsc < min' then
4       create_migration_move(int_rsc, ext_rsc)
5  end
6  else if (dst_rsc − src_rsc) > (min' * δ) then
7       create_migration_move(int_rsc, ext_rsc)
8  end
```

migration frequency shows how often federates are being moved between shared resources. The migration ratio comprises the division of incoming migrations by the outgoing migrations in a resource. This ratio indicates the relative amount of federates that are kept in a resource. The adaptation analysis is realized in collective and individual scopes. The collective scope observes all the migration frequencies and the migration ratios as whole to determine balancing inefficiencies. In the individual scope, each federate and resource is evaluated regardless the rest of the system, so specific actions are triggered depending on particular load characteristics. At the end of the adaptation procedure, the adjustment modifications obtained from the adaptation analysis are imposed on the parameters that configure the balancing system, i. e., the thresholds that determine the imbalances and correct them.

6.3.2.1 Data Gathering for Self-Adaptation Analysis

Acting responsively, the adaptation system needs to constantly collect data about the balancing system in order to mould it according to real load redistribution needs. Thus, the adaptation system gathers migration information from all the simulation elements and all the shared resources. The information is retrieved together with the balancing scheme's monitoring data, so adaptation components are placed together with the balancing system, as depicted in Figure 4.1.

Placed in a resource, each Federate History (FH) registers the last H migration moves of a federate in a list. Upon request from the Federate Balancing Interface, FH provides its migration history list. The list is attached to the federate's monitoring information, CPU consumption and/or communication rate, and sent to a LLB. Once the information from federates in a resource is gathered by the LLB, it is processed by a Resource Adaptation Component (RAC) and re-attached to the respective federate's monitoring information, which is merged in a larger monitoring packet to be sent to a CLB for further monitoring processing. Thus, each RAC keeps track of a set of federates' migration history. The migration history is divided in H elements that register if a federate went through migration in certain balancing cycle. As a matter of simplification, 1 marks a migration and 0 marks no migration in such a history list.

Responsible for a resource, a Resource History (RH) saves the last H incoming and outgoing federate migrations. Triggered by a LLB, a RAC requests to RH the register of its H migration moves in the past balancing cycles. RAC then processes the retrieved migration information and adds it to the resource monitoring packet, which is sent to a CLB. Such processing generates a resource's migration ratio. As a result, a resource migration history stores the migration moves occurred in a resource in the past $H/2$ balancing cycles because each entry in the history corresponds to an incoming or outgoing number of migrations. More specifically, since the balancing system performs only one migration per balancing cycle for a resource, this history list contains only 1s and 0s.

Placed together with a CLB, a Cluster History (CH) stores the last H incoming and outgoing federate migrations for each CLB. Activated by its CLB in each balancing cycle, a CAC requests CH data regarding the H last migration moves. After receiving all the monitoring information from its LLBs, the CLB also extracts the federate frequencies and the resources ratios. At the end, the adaptation component processes the CLB migration

history to generated a ratio, the resource migration ratios to result in a cluster-resource migration ratio, and the individual federate frequencies to produce a general cluster migration frequency. Likewise RH, a CH registers the migrations occurred in the past $H/2$ balancing cycles. Each element in the history list shows the number of incoming or outgoing migrations performed by a CLB. However, in this case, the number of migrations might exceed 1 since a CLB realizes several inter-domain migrations per balancing cycle.

The method employed to process the gathered migration data is related to certain police that is adopted in the adaptation algorithm. In this work, the adaptation technique is restricted to a police based on migration frequency analysis. Therefore, balancing adjustments are performed when a federate presents too frequent migrations. In the same way for resources, the migration ratio is calculated based on incoming and outgoing migration frequencies.

6.3.2.2 Calculation of Frequencies and Ratios

The first part of interpreting the collected data is processing it and calculating the migration frequencies and ratios. These calculations provide means to validate the load balancing efficiency and determine the proper changes to improve it. All the calculations employ migration history (H) as the main determinant element. In the adaptation analysis, time is fundamental to retrieve ratios and frequencies since they present relevance for the time that they were obtained. Thus, the history comprises this time need, and it is completely based on balancing cycles since they conduct the pace in which migrations are produced in the system.

The calculations are first realized in each resource by an RAC. Basically, this component obtains federates' migration frequencies and resource's migration ratios. The migration frequency of each federate is obtained through the calculation of a weighted mean of its history list elements, as shown in Formula 6.14. The weights are disposed on the history elements to provide priority to the most recent migration moves since they reflect the current load balancing behaviour. As a result, as a simple method to emphasize on the most recent migration moves, each i element receives w_i weight of $H - i$ in the calculation, considering H as the size of the history.

$$\overline{freq} = (\sum_{i}^{H} f_i \times w_i) / \sum_{i}^{H} w_i \tag{6.14}$$

- H: history size;
- f_i: federate migration frequency (number of migrations – 1 or 0 – in a given balancing cycle);
- w_i: weight of federate migration frequency.

Each resource's migration ratio is calculated by dividing its weighted mean of $H/2$ incoming migrations by its weighted mean of $H/2$ outgoing migrations. The resource's migration weighted mean is obtained according to Formula 6.15. The mean is considered only if the sum (n_{sum}) of the first n elements in the history list is larger than zero. Since the recent past is highly relevant for the ratio calculation, the average is nullified if such a recent past does not present any register of migration move.

$$\overline{avg}_{rsc} = \begin{cases} (\sum_{i}^{H} x_i \times (H - i)) / \sum_{i}^{H} i & , n_{sum} > 0 \\ 0 & , n_{sum} = 0 \end{cases} \tag{6.15}$$

- H: history size;
- n_{sum}: sum of the first n elements in the history list;
- x_i: number of incoming of outgoing migrations in a given balancing cycle.

The parameter n that restricts the first elements of a history list is determined by Formula 6.16. In the formula, an n is selected from a pre-defined condition. The condition determines the minimum frequency in which the migrations occur in the resource's history list in order for the frequency calculation to be considered relevant to the adaptation analysis. It is assumed in this work a minimum resource migration frequency, which is represented by the migration move pattern in the history list that conforms with the following regular expression: $(100)^{+}$. Observing this history list as an arithmetic progression, Formula 6.17 represents this minimum frequency by defining x as the nth element in the progression in which its initial term is H and its common difference is -3. Therefore, with x defined, the sum of elements in such sparse progression is calculated

in Formula 6.16. The formula obtains the sum of elements in the same hypothetical migration sequence that is inferred for the calculation of x, considering that x is the last element in the sequence. The calculated sum is compared with the sum of the first n elements in the history, assuming that all the elements in the history are different from zero, i. e., equal to 1.

$$\sum_{i}^{n} h_i \leq (H + x) \times (\lceil H/3 \rceil + 1) \times 1/2 \tag{6.16}$$

- H: history size;
- h_i: federate migration frequency in a given balancing cycle;
- x: the nth element in the progression.

$$x = H + \lceil H/3 \rceil \times (-3) \tag{6.17}$$

- H: history size.

After the frequencies and ratios computed in each resource are retrieved by a CLB, a CAC merges all the information in a second step of calculations. The component initially computes the inter-domain (cluster) migration ratio by also employing Formulas 6.15, 6.16, and 6.17, likewise RAC in each resource. However, in this case, the incoming and outgoing migration frequencies present values that exceeds 1, so the weighted mean is not a value that ranges between 0 and 1, such as in the resource migration ratio calculation. Moreover, the CAC computes the the ratios' and frequencies' average. An arithmetic mean is used to obtain the migration ratios' average in a cluster while the frequencies' average is determined by Formula 6.18. Assuming R as the number of resources in a cluster and F_j the number of federates in a specific resource, the CAC calculates the frequency mean of a cluster based on the arithmetic mean of frequencies in each resource.

$$\overline{freq}_{avg} = \sum_{j}^{R} \sum_{i}^{Fj} freq_i/Fj/R \tag{6.18}$$

- R: number of resources in a cluster;

- F_j: number of federates in a resource;
- $freq_i$: federate migration frequency in a given balancing cycle.

6.3.2.3 Detection of Load Balancing Inefficiency

In this part of the adaptation process, the collected data is analyzed to identify irregularities in the load balancing system that might lead it to malfunction. All the gathered data, as well as the computed averages, are used to determine adjustments in load balancing thresholds or to flag some specific federates and/or resources. Modifications in the thresholds are defined from the analysis of collective ratios and frequencies, i. e., metrics that represent the load balancing system's behaviour for the entire cluster. Analysis of individual ratios and frequencies indicates that only certain resources and federates present some particular characteristic, such as cyclic oscillations of federate and external background load, which drive the load balancing system to react improperly. Thus, in this detection step, a pre-processing is applied on the collected data sample and comparison analyses are performed.

Initially, the CAC requires a pre-processing for the averages in order to represent the load balancing system's efficiency more accurately. These average adjustments regard the relations between the number of federates and resources in a cluster. Because the frequency average can be unrepresentative of the real load redistribution effects when the number of resources is largely inferior than the number of simulation federates, the frequency average is adjusted by multiplying the current average by the ratio $resources/federates$ given by Formula 6.19. In the formula, the average is increased in an inversely proportional rate to the ratio between the number of federates and the number of resources. This adjustment improves the perception of migration frequency in a cluster because a misleading highly reactive balancing behaviour can be hidden by a low average as a reflect of numerous federates running on the shared resources, which always leads to a interpretation of just sparse migrations in the system.

$$f_{adj}(r, f) = \begin{cases} (2 - r/f) & , r/f \leq 1 \\ 1 & , r/f > 1 \end{cases} \tag{6.19}$$

- r: total number of resources in a cluster;

- f: total number of federates in a cluster.

The migration ratio average also needs to be adjusted, introducing awareness of the inter-domain migration ratio and the relation between number of federates and number of resources, as described in Formula 6.20. In this case, the opposite case of the relation between the number of federates and the number of resources influences the interpretation of the migration ratio. In order to introduce awareness of such a situation, the migration ratio average is modified by ratio $federates/resources$, which is computed in Formula 6.21. Moreover, because resources can experience intra-domain and inter-domain migrations, their migration ratio may contain some migrations that belong to inter-domain redistribution; thus, an adjustment regarding the inter-domain migration moves is provided, which is represented by β in Formula 6.22. The formula considers the inter-domain incoming and outgoing migration frequencies that need to be excluded from the migration ratio average. For each balancing cycle (i) in the history list ($H/2$), the number of *in* migrations is summed to the *out* migrations. This sum is divided by the current number of resources in order to obtain the ratio of inter-domain migrations per resource. An weighted mean is computed with all this ratio values in the history list. At the end, this inter-domain ratio shows the amount of inter-domain migrations that have been performed by the load balancing system in each resource. Consequently, the complement of this ratio represents the amount of intra-domain migrations in each resource, and it is used to specify the migration ratio value to the local scope.

$$avg_{rsc}^{adj} = avg_{rsc} \times \alpha \times \beta \tag{6.20}$$

- avg_{rsc}: average resource migration ratio.

$$\alpha = \begin{cases} (2 - f/r) & , f/r \leq 1 \\ 1 & , r/f > 1 \end{cases} \tag{6.21}$$

- r: total number of resources in a cluster;
- f: total number of federates in a cluster.

$$\beta = 1 - \frac{\sum_i^{H/2}((in_i + out_i)/rsc) \times w_i}{\sum_i^{H/2} w_i} \tag{6.22}$$

- H: history size;
- in_i: number of incoming migrations into a cluster in a given balancing cycle;
- out_i: number of outgoing migrations into a cluster in a given balancing cycle;
- rsc: number of resources in a cluster;
- w_i: weight for the ratio in a given balancing cycle.

At this point in the analysis, a CAC contains the cluster migration ratio, the resource migration ratio average, the federate migration frequency average, a list of resource migration ratios, and a list of federate migration frequencies. All these metrics are used to determine the current adaptation, which comprises adjustments to both inter-domain and local (intra-domain) load balancing. The cluster migration ratio is the main parameter to determine the inter-domain balancing adjustment, as described in Algorithm 19. According to the algorithm, the ratio ($ratio_{clb}$) is evaluated; if it presents value in the interval delimited by t and $1/t$ thresholds, adjustments are calculated to modify the balancing parameters. The threshold t represents the difference between incoming and outgoing migrations, and it determines how selective the adaptation is with the load balancing responsiveness. Furthermore, current cluster incoming ($clusterINMig$) and outgoing ($clusterOUTMig$) migration frequencies are used to identify a decay for the last balancing parameter adjustment when no new adaptations are needed for the current balancing efficiency.

For analyzing the intra-domain balancing efficiency, resource ratio average and federate frequency average are employed in the adaptation analysis, as described in Algorithm 20. Initially CAC assess the migration ratio by checking if the ratio lies in the interval delimited by t_{rsc} and $1/t_{rsc}$. Likewise in the case of the inter-domain adjustment, t_{rsc} is pre-defined and corresponds to the difference allowed between the number of incoming and outgoing intra-cluster migrations. If a need of adaption is identified, the respective adjustment is calculated. CAC also analyzes the frequency average by evaluating if the general migration frequency in a cluster exceeds a threshold, t_{freq}. The threshold represents minimum migration frequency considered normal by the adaptation algorithm and is obtained through Formulas 6.23 and 6.24. This minimum frequency is pre-defined in

Algorithm 19: Distributed Redistribution Adaptation Algorithm

```
1  Require: ratio_clb, clusterINMig, clusterOUTMig
2  if ratio_clb < t & ratio_clb > 1/t then
3      if ratio_clb ≥ 1 then
4          ajdust_Acceptance_Threshold()
5      end
6      else if ratio_clb < 1 then
7          adjust_Overloaded_List_Threshold()
8      end
9  end
10 else
11     if clusterINMig == 0 then
12         if past_Increase_IN then
13             decay_IN_factor()
14         end
15     end
16     else if clusterOUTMig == 0 then
17         if past_Increase_OUT then
18             decay_OUT_factor()
19         end
20     end
21 end
```

the system and presents a sparse migration pattern that can be represented by the following regular expression: $(1001)^{+}$. To obtain this minimum allowed migration frequency, it is assumed that a element i in the migration history list receives weight $H - i$ for the weighted mean calculation for a federate's migration frequencies, which is described in Formula 6.14. According to this assigned weight, the second formula determines the number of elements (n') for the pre-defined sparse migration frequency; the formula interprets the nth element in the progression (a_n) as H, the first element weight in the history list, and obtains n' based on a common difference of 3. Then, the first formula defines the sum (S') of elements in the progression according to the number of elements (n'). CAC uses S' as the threshold t_{freq} to assess the general migration in a cluster. If the migration frequency exceeds the threshold, its respective adjustment is calculated.

$$S' = \frac{3n'^2 - n'}{2} \tag{6.23}$$

- n': number of elements in a progression for a defined sparsity.

$$n' = \lfloor \frac{a_n + 2}{3} \rfloor \tag{6.24}$$

- a_n: the value of the nth in the progression.

Algorithm 20: Local Redistribution Adaptation Algorithm

```
1  Require: ratio_avg^rsc, freq_avg
2  if ratio_avg^rsc < t_rsc & ratio_avg^rsc > 1/t_rsc then
3      if ratio_avg^rsc ≥ 1 then
4          adj_rsc = calc_rsc_ajdust_factor1()
5      end
6      else if ratio_avg^rsc < 1 then
7          adj_rsc = calc_rsc_ajdust_factor2()
8      end
9  end
10 if freq_avg > t_freq then
11     adj_freq = calc_freq_adjust_factor()
12 end
13 if adj_rsc > 0 || adj_freq > 0 then
14     if adj_rsc == 0 then
15         adj_final = adj_freq
16     end
17     else if adj_freq == 0 then
18         adj_final = adj_rsc
19     end
20     else if adj_rsc > adr_freq then
21         adj_final = calc_local_adjustment1()
22     end
23     else
24         adj_final = calc_local_adjustment2()
25     end
26     calc_decay_fraction()
27 end
28 else
29     decay_adjust_factor()
30     adj_final = 0
31     resource_list = identify_unstable_rsc()
32     federate_list = identify_unstable_fed()
33 end
```

After both adjustments related to frequency average and migration ratio average are obtained, CAC checks the need for adaptation, which is denoted by $adj_{rsc} > 0$ or $adj_{freq} > 0$. If any to the values is larger than zero, further calculations are required. On the other hand, if both values are equal to zero, the final adjustment is not computed, but a decay is applied to the last adjustment, and a search for unstable resources and federates is realized. For this search, each federate and resource is evaluated individually. The evaluations consist in comparisons with t_{freq} for determining unstable federates and comparisons with t_{rsc} and $1/t_{rsc}$ for identifying unstable resources. These elements are marked in order not to be considered in the next $H/2$ load balancing cycles.

The marking of unstable federates and resources prevents the load balancing system to perform load redistribution based on them. Some simulation federates or shared resources produce unstable load due to cyclic load changes, excessive simulation load, or heterogeneity of resources. A federate or an external process might oscillate its load

dynamically in certain frequency that makes the balancing system reorganize the load often. Also, the excessive load or the heterogeneity of resources might cause the balancing system to move load constantly due to the creation of new overloaded resources through the redistribution procedure. Thus, such unstable elements are exceptional cases in the system, which require additional handling, and being marked enables a decrease of improper migrations.

6.3.2.4 Adjustment of Thresholds for the Load Balancing System

After needed adaptations are detected, adjustments are calculated according to the amount of divergence that the load balancing presents when compared with its expected behaviour. At the end of the process, the adaptation involves the modifications in three load balancing parameters: overloaded list selection, inter-domain acceptance, and minimum resource load. These parameters are directly related to the distributed load balancing scheme, so modifying them results in an increase or decrease of migrations. Also, all the balancing parameters initially present small values, enabling high responsiveness and a large number of migrations. These migrations are then gradually reduced if adjustments are detected by the adaptation system.

The overloaded list selection parameter determines the number of resources that are selected for the inter-domain redistribution analysis, as described in Algorithm 16. This balancing parameter is directly modified by *selectionAdj*, which contains the respective adaptation value. This value is calculated based on the range $1..1/t_{clb}$, which corresponds to the range for a larger amount of outgoing migrations. A migration ratio ($ratio_{clb}$) in this range means that outgoing migrations are required to balance the system's load, but such migrations are produced excessively. Consequently, the parameter's adjustment proportionally increases to the proximity of $ratio_{clb}$ to 1, as computed with the following formula: $ratio_{clb} \times (ratio_{clb} - 1/t_{clb})/(1 - 1/t_{clb})$.

The inter-domain acceptance is responsible for matching external load with internal resources of a cluster, as delineated in Algorithm 18. The parameter is updated according to *acceptanceAdj*, which receives the adaptation value determined in the last balancing cycle. The value is obtained through the following formula: $ratio_{clb} \times (t_{clb} - ratio_{clb})/(t_{clb} - 1)$. Likewise the calculation of the adjustment for the overloaded list selection parameter, the formula considers the distance from $ratio_{clb}$ to 1: a closer $ratio_{clb}$ to 1 reflects in a

larger adjustment for the acceptance. However, in this case, the range $t_{clb}..1$ is observed since this adjustment is focused on the number of incoming migrations. Thus, when $ratio_{clb}$ is larger than 1, the balancing system is receiving load, but if this ratio is in the range, the CLB is accepting load excessively.

As presented in Algorithm 2, the minimum resource load is used in the intra-domain re-partitioning procedure to determine migration pairs through load comparisons. Modifying this balancing parameter considerably affects the pair-match algorithm. In Formula 6.25, the local adjustment related to the resource ratio is calculated by determining the proximity of r_{rsc} to 1, i. e., a high incidence of unnecessary migration moves. In this case, the migration ratio adjustment is proportional to this proximity. In Formula 6.26, the migration frequency adjustment is calculated according to the migration frequency, which ranges between 1 and t_{freq}. In the range, 1 means the maximum migration frequency, and t_{freq} the minimum acceptable frequency. A closer value of $freq$ to 1 requires a larger, proportional adjustment, what is provided in the formula. The final local adjustment (adj_{final}) is calculated with both ratio and frequency adjustments. According to Algorithm 20, method $calc\_local\_adjustment1()$ uses Formula 6.27 to calculate the final adjustment if adj_{rsc} is larger than adj_{freq}. On the other hand, if adj_{rsc} is smaller of equal to adj_{freq}, method $calc\_local\_adjustment2()$ employs Formula 6.28 to compute the final adjustment value. For both formulas, more weight is given in the sum to the parameter that denotes more drastic modifications in the load balancing system.

$$adj_{rsc} = \begin{cases} (r_{rsc} - t_{rsc})/(1 - t_{rsc}) & , r_{rsc} \geq 1 \\ (t_{rsc} - r_{rsc})/(t_{rsc} - 1) & , r_{rsc} < 1 \end{cases} \tag{6.25}$$

- r_{rsc}: average federate migration ratio for the resources in a cluster;
- f_{rsc}: threshold defined for a resouces' migration ratios.

$$adj_{freq} = \frac{freq - t_{freq}}{1 - t_{freq}} \tag{6.26}$$

- $freq$: average federate migration frequency in a cluster;
- t_{freq}: threshold determined for federates' frequencies.

$$adj_{final} = adj_{rsc} \times (1 - \frac{adj_{freq}}{adj_{rsc}}) + adj_{freq} \times (\frac{adj_{freq}}{adj_{rsc}}) \tag{6.27}$$

- adj_{rsc}: adjustment determined based on the migration ratio analysis;
- adj_{freq}: adjustment determined based on the migration frequency analysis.

$$adj_{final} = adj_{freq} \times (1 - \frac{adj_{rsc}}{adj_{freq}}) + adj_{rsc} \times (\frac{adj_{rsc}}{adj_{freq}}) \tag{6.28}$$

- adj_{rsc}: adjustment determined based on the migration ratio analysis;
- adj_{freq}: adjustment determined based on the migration frequency analysis.

6.4 Predictive Load Balancing System

In order to build a predictive balancing scheme, prediction methods, functions, and components are incorporated into the distributed balancing scheme presented in Chapter 5. Consequently, the proposed balancing scheme is structured on the same hierarchical organization of balancing elements. Moreover, the balancing process basically presents similar order of tasks as the aforementioned distributed balancing scheme, which consists in monitoring resources, re-arranging simulation load, and migrating federates. The proposed architecture contains additional elements for calculating and providing predictions, and the protocols are modified to be able to manage policies based on load predictions.

6.4.1 General Architecture of the Predictive Balancing System

Based on the architecture described in Chapter 5 and as depicted in Figure 6.4, a Cluster Load Balancer (CLB) is responsible for managing a set of resources (cluster) and the federates placed on them. This management comprises of gathering load information from resources and federates, analyzing the collected data and data from neighbour CLBs, and triggering federate migrations. In order to coordinate the whole the balancing process, the CLB is connected with neighbour CLBs and accesses the Monitoring Interface, Local Load Balancers (LLBs), and the Prediction Interface.

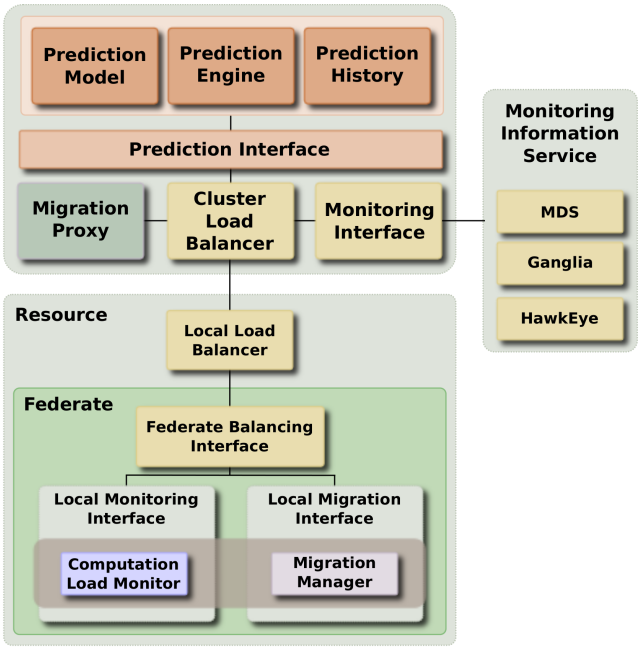


Figure 6.4: Predictive Balancing Scheme's Architecture

The Monitoring Interface introduces transparency for the access to monitoring data from a Monitoring Information Service (MIS). Providing load information related to the cluster monitoring level, the MIS accesses Grid services for gathering application-independent load metrics. Grid is a resource management system widely used to coordinate the execution of distributed applications on shared resources [40]. The access to the resource load monitoring information is made through Grids Index Service, which obtains the data from Monitoring and Discovery Service.

An LLB acts as an interface for each resource and its containing federates. Placed in every resource, LLB is focused on locally managing simulation federates by gathering monitoring data, applying data aggregation methods, and forwarding migration calls. The gathering of monitoring data consists of forwarding a call from a CLB through a LLB and its Federate Balancing Federates, which triggers each Local Monitoring Interface to retrieve information regarding a federate's CPU consumption. The dispatching of migration calls is triggered by a CLB call, which ends in the Migration Manager (MM).

The migration procedure is performed in two-phases, similarly to the federate migration described in Section 3. The federate migration is divided in a transfer of static, initialization files and dynamic execution status data. Through Grid services, the first

part of the transfer and initialization of the federate at the remote resource is performed. In the second part of the migration, a peer-to-peer transfer is realized between the MMs at the local resource and at the remote resource to enable a migrating federate to restore its execution state. A Migration Proxy is used for migrations between unreachable resources.

A Prediction Interface is introduced in the architecture to enable and facilitate the access to predictions of collected sample data in the balancing system. This interface allows transparent access to the Prediction Engine. The engine processes the incoming data based on a history and a prediction model. The prediction data history is stored in the Prediction History component. This element is dependent on the prediction model since different metrics need to be saved to generate a prediction. For the prediction model used in this work, smoothing and trend values are stored in a list. The list is updated according to the balancing system needs. The model is determined in the Prediction Model component and fed by the Prediction Engine with the information provided by the CLB and the Prediction History elements.

6.4.2 Load Redistribution Algorithm

The predictive balancing scheme presents a redistribution algorithm divided in three phases: monitoring of distributed load, rearrangement of simulation entities, and migration of federates. The basic structure of the proposed algorithm, as well as the ordering of major tasks, follows the computational load redistribution algorithm designed and described in Chapter 5. As delineated in Algorithm 21, in order to enable responsiveness to dynamic load changes, the balancing system is required to evaluate load distribution periodically. This consequently imposes the scheme to work in balancing cycles of Δ time units. In this work, the balancing cycle's time interval is a predefined value limited by the data refresh rate of the external, third-party monitoring tool.

The monitoring phase directly dictates the responsiveness of the balancing system through the timely provision of precise distributed load status to the balancing system. Besides the importance of the refresh rate of data for redistributing load, the precision and the interpretation of the collected data considerably influences the balancing performance. This influence motivates the search for new monitoring metrics and new methods of evaluating the load distribution. In this case, prediction techniques are ap-

Algorithm 21: Distributed Dynamic Load Balancing Algorithm

```
1  while TRUE do
2      loads ⇐ query_MDS()
3      current_loads ⇐ filter_MDS_data(loads)
4      current_loads ⇐ normalize_loads(current_loads, benchmark)
5      current_loads ⇐ prediction(current_loads, old_loads, mig_RSCs)
6      old_loads ⇐ current_loads
7      overload_cand ⇐ select_overload(current_loads)
8      spec_loads ⇐ request_LLBs(overload_cand)
9      mng_loads ⇐ filter(current_loads, spec_loads)
10     mig_moves ⇐ local_bal(mng_loads, SP)
11     mig_moves ⇐ local_bal(mng_loads, MP)
12     mig_moves ⇐ local_bal(mng_loads, LP)
13     send_migration_moves(mig_moves)
14     if mig_moves = ∅ then
15         data_neighbours ⇐ request_Neighbour_Load_Data()
16     end
17     else
18         if relFactor ≥ random_number(1, 100) then
19             data_neighbours ⇐ request_Neighbour_Load_Data()
20         end
21         else
22             data_neighbours ⇐ ∅
23         end
24     end
25     neighbours ⇐ identify_neighbour_Less_Load()
26     while neigbours! = ∅ do
27         overloaded_resources ⇐ select(firstNeighbour, SP)
28         overloaded_resources ⇐ select(firstNeighbour, MP)
29         overloaded_resources ⇐ select(firstNeighbour, LP)
30         federates ⇐ select(spec_loads, overloaded)
31         eliminate_first(neigbours)
32     end
33     send_to_neighbour(overloaded_resources, federates)
34     migration_moves ⇐ wait_for_migration_moves()
35     send_migration_moves(migration_moves)
36     wait( Δ )
37 end
```

plied to the collected data, requiring modifications on the balancing algorithm to bear this new evaluation of resources' and federates' load status.

According to the main algorithm, data gathering is the first step in the monitoring procedure. After raw data about the resources' load status is retrieved, filtering is applied to eliminate unneeded or misleading values. Information that is not needed by the detection mechanism or data related to resources that are not managed by the balancing system substantially influences the detection calculations and is removed. In order to properly compare and identify imbalances in an heterogeneous environment, a normalization method is employed. The gathered data is normalized through benchmarks to allow the appropriate comparison of resources' loads according to their computational capacities. Finally, predictions are calculated for each resource's load status value. For

the prediction model employed in the balancing system, this calculation involves the current load value, the previous smoothed load value, and the migration status of the resource in the current balancing cycle. The previous smoothed load value is a specific requirement defined by the prediction model. The migration status is used to identify the resources selected for migration in the previous cycle; this information is used to adjust the prediction parameters, so the load oscillations caused by the migration moves are properly considered in the calculations, improving the system's responsiveness. For each resource, short-term, medium-term, and long-term predictions are calculated. These values are used later in the algorithm to detect imbalances and determine a reconfiguration of load on the resources.

Based on the gathered and filtered data, overloaded resources are identified and federate-specific load status data is collected from each overloaded resource. Upon receiving this load information, another round of filtering is applied to identify the manageable overloaded resources, which can have load relief through federate migrations. Similarly to resources' loads, predictions for federates' loads are calculated, so they can be used in the same projection in the future as the resources.

Together with part of the monitoring (the detection), the redistribution phase defines a load rearrangement towards a decrease of imbalances. This phase also needs to be modified in order to accommodate the prediction metrics in its decision-making procedure. Instead of being purely based on the load status of a recent past, the modifications enable the analysis of future simulation load behaviour. This analysis provides a predictive reaction to load oscillations and a decrease of time gap between the instant in which data is gathered and the moment when the data is observed in the balancing system. This time difference is mainly generated in the inter-domain redistribution process due to the asynchronous characteristics between CLBs' gathering of monitoring data.

The redistribution phase initiates with the local balancing calls for each type of prediction (SP, MP, and LP). Summarized in Algorithm 22, the local balancing consists in matching the overloaded resources with the underloaded resources according to the required type of prediction. The type of local balancing designates the type of prediction that is used for the analysis. Three local redistribution processes are realized independently, giving more importance to the prediction closer to the current time through the order of local redistribution calls, as shown in Algorithm 21. At the beginning of the

Algorithm 22: Local Predictive Balancing Algorithm

```
1  Require: mng_loads, type
2  if mng_loads! = ∅ then
3      mng_loads ⇐ order(mng_loads, type)
4      mean, bds ⇐ calculate_mean_bds(mng_loads, type)
5      over, under ⇐ select(mng_loads, mean, bds)
6      mig_moves ⇐ redistribute_local(mng_loads, over, under, type)
7      mng_loads ⇐ clean_migrations(mng_loads, mig_moves)
8  end
9  Return: mig_moves
```

Algorithm 23: Local Predictive Pair-Match Evaluation Algorithm

```
1  Require: src_rsc, dst_rsc, min, φ, δ
2  min, δ, φ ⇐ adjust_Parameters(src_direction, dst_direction, type)
3  if dst_rsc < min then
4      if number_fed(src_rsc) > 1 then
5          create_migration_move(src_rsc, dst_rsc)
6      end
7      else if number_fed(src_rsc) = 1 & src_rsc > (min ∗ φ) then
8          create_migration_move(src_rsc, dst_rsc)
9      end
10 end
11 else if (dst_rsc − src_rsc) > (min ∗ δ) then
12     if number_fed(src_rsc) > 1 then
13         create_migration_move(src_rsc, dst_rsc)
14     end
15     else if number_fed(src_rsc) = 1 & (dst_rsc − src_rsc) > (min ∗ φ) then
16         create_migration_move(src_rsc, dst_rsc)
17     end
18 end
```

local redistribution, an ordering is performed according to the type of prediction that is defined. This ordering allows the procedure to properly match overloaded and underloaded resources. At the end of procedure, all the migration moves are returned, and the respective resources are removed from the list for the next local balancing call. For either overloaded or underloaded resource, a local pair-match analysis is performed, as described in Algorithm 23. In this analysis, the resources' loads are compared according to predefined and adjusted thresholds. If the difference of load between an overloaded and an underloaded resource exceeds certain value and justifies a need of load transfer, a migration move is generated and returned.

As detailed in Section 5.4, the inter-domain balancing is activated based on the defined local migrations and on the success rate of the inter-domain migrations in the previous balancing cycle. For continuing with the inter-domain redistribution, cluster load is requested from the neighbour CLBs. This load consists in a set of averages that represent the general load of managed resources for each prediction in a given moment, which does

not need to be in the same instant that the information was requested. In order to prepare these averages for sending to the requesting CLB, the data sample (list of resources) is filtered. Since sudden spikes of computational load might considerably mislead the calculation of the CLB average load, the discrepant values need to identify and removed from the list. For a large list of resources, the computational load status in the data sample generally tend to present a Gaussian distribution, and the outliers in this list might influence the average to differ from the normal distribution. Interquartile range is used to determine the outliers in the data sample and provide a selection of representative resources. The interquartile range provides a robust statistical method with a breakdown point of 25%, which assures an efficient technique for detecting outliers and improves the representation accuracy of the CLB average load. An average is calculated with the middle fifty values of the sample data for each type of load prediction. The averages are returned to the caller CLB, which can proceed with its inter-domain balancing.

With the averages returning from the neighbour CLBs, a selection is performed to identify imbalances between the domains. For each type of prediction value, resources with load larger than a threshold are selected as candidates for further, remote redistribution analysis. This threshold is determined by a relation between the local CLB average and a neighbour's average ($\alpha \times load_{local} + (1 - \alpha) \times load_{remote}$). These resources are prepared to be sent to a neighbour CLB according to their selection; this preparation comprises selecting a candidate federate and aggregating prediction values for each resource.

Upon receiving resources selected for inter-domain, a neighbour CLB initiates its inter-domain redistribution analysis. The analysis consists in three sets of evaluations, each one defined by a type of prediction: SP, MP, and LP. Performed sequentially, the evaluations farther from the current time receive less priority and are performed with the remaining non-selected resources. Similarly to the local part of the redistribution phase, the evaluations encompass sequential comparisons between the remote resources (overload) and the local resources (underloaded), as described in Algorithm 24. Before the lists of remote and local resources are analyzed, both are ordered according to the type of prediction metric that is set to be used. For each selected pair of resources, a comparison based on thresholds is performed, as described in Algorithm 25. In the algorithm, the thresholds are first adjusted according to the defined type of prediction

Algorithm 24: Inter-Domain Remote Predictive Balancing Algorithm

```
1  Require: external_loads, internal_loads
2  if external_loads! = ∅ AND internal_loads! = ∅ then
3      external_loads ⇐ order(external_loads, type)
4      internal_loads ⇐ order(internal_loads, type)
5      foreach external_loads do
6          foreach internal_loads do
7              src_rsc ⇐ select(external_loads)
8              dst_rsc ⇐ select(internal_loads)
9              mig_moves ⇐ evaluate(src_rsc, dst_rsctype)
10             external_loads ⇐ clean(external_loads, mig_moves)
11         end
12     end
13 end
14 Return: external_loads, mig_moves
```

Algorithm 25: Inter-Domain Predictive Pair-Match Evaluation Algorithm

```
1  Require: int_rsc, ext_rsc, min, δ, type
2  min, δ ⇐ adjust_Parameters(src_direction, dst_direction, type)
3  if int_rsc < min then
4      create_migration_move(int_rsc, ext_rsc)
5  end
6  else if (dst_rsc − src_rsc) > (min ∗ δ) then
7      create_migration_move(int_rsc, ext_rsc)
8  end
```

and the load tendencies of both resources. If the load difference between the resources is large enough to justify an imbalance (threshold), a federate migration is generated. At the end of the selection, the migration calls are grouped and the resources involved with the migrations are removed from the list for the next selection call.

Finally, the neighbour CLBs respond with the migration calls after the remote part of the inter-domain redistribution is performed. As soon as the calls arrive at the CLB, they are forward to their respective resource to initiate the federate migration process. The resources selected for migration are put in a list, so this list is used to correct the prediction calculations in the next balancing cycle.

6.4.3 Forecasting Method for Obtaining Load Status

Independent of the prediction model employed in the balancing scheme and as formerly mentioned balancing algorithm description, the load forecasting (prediction) is performed in short-term, medium-term, and long-term. These method of employing different predictions in the algorithm is an attempt to provide cautious load changes through short-term predictions and preventive load preventive load redistribution through medium-term and

Algorithm 26: Predictive Adjustment

```
1  Require: src_direction, dst_direction, type
2  if src_direction ≥ 0 AND dst_direction < 0 then
3      adjust(min, δ, φ, type, COND1)
4  end
5  else if src_direction ≥ 0 AND dst_direction ≥ 0 then
6      adjust(min, δ, φ, type, COND2)
7  end
8  else if src_direction < 0 AND dst_direction < 0 then
9      adjust(min, δ, φ, type, COND2)
10 end
11 else if src_direction < 0 AND dst_direction ≥ 0 then
12     adjust(min, δ, φ, type, COND3)
13 end
14 Return: min, δ, φ
```

long-term predictions. Consequently, the majority of the load changes are determined by the short-term values, and strong tendencies of load increase or decrease are also considered to be prevented. In the case of the balancing system, the period that defines the forecasting is based on balancing cycles, so the short, medium, and long terms are defines as 1, 3, and 5 respectively.

Another aspect previously mentioned is the modification of the balancing algorithm with adjustments. In each balancing algorithm (local or inter-domain), adjustments are applied on the pair-match thresholds. These modifications on the parameters are dependent on the type of validation and the uni-dimensional direction that a resource's load is tending, as described in Algorithm 26. Longer-term predictions increment the decision-making in the proposed balancing scheme by offering an ability that enables preventive reaction to load oscillations, and a projection in the future can substantially increase the discrepancies between values and the prediction's degree of uncertainty. Thus, these predictions are not considered with the same weight in the comparisons; the thresholds for these types of prediction (medium and long terms) receive larger values that are proportional to their projection time in order to provide more tolerant imbalance detection analyses.

The load tendency direction is also employed in the adjustment procedure, as detailed in Algorithm 26. This direction of load oscillation is calculated for each medium or long term prediction: medium-term direction (md) and long-term direction (dl); it provides means to define if a load variation is stabilizing, continues to increase or decrease, or is on a verge of a inversion. The direction is simply obtained by calculating the difference between a projection and the short-term prediction, as described in Formula 6.29.

According to the algorithm, the directions of both source and destination resources are used in a simple mechanism to define a type of adjustment ($COND1$, $COND2$, and $COND3$). $COND1$ represents the situation in which thresholds are less tolerant to imbalances since both resources tend to increase the load gap between them: the overloaded resource shows increasing load and the underloaded resource shows decreasing load. $COND2$ represents an intermediary situation in which the thresholds receive an slight increase in the tolerance to imbalances because one of the resources is stabilizing (inversion of load variation): the overloaded resource shows decreasing load or the underloaded resource shows increasing load. Finally, $COND3$ represents the situation in which the thresholds receive the largest tolerance since both resources present inversion in their load tendency (stabilization): overloaded is tending to decrease load and underloaded is tending to increase.

$$dm_k = f_{m+k} - f_{m+i}, k > i \tag{6.29}$$

- f_{m+k}: prediction of the load (projection) in k balancing cycles;
- f_{m+i}: short-term load prediction (smoothing);
- k: number of future balancing cycles;
- i: current balancing cycle.

The main objective of the proposed balancing system is the provision of a prediction method that offer preventive responses to load changes, so load oscillations need to be considered in order to provide more realistic predictions. However, when load transfers are performed through federate migrations, load oscillations are also produced. These variations are not part of the simulation load behaviour, but they need to be computed in the scheme to introduce awareness to them. Thus, for the calculation of predictions, resources selected for migration are considered, so the prediction model's parameters are modified accordingly. As described in the next subsection, the parameters received values to give more emphasis to the current gathered value than to the past load values, and consequently a more accurate adaptation of the load trend calculation, which substantially influences the computing of predictions.

6.4.4 Prediction Model Employed in the Predictive Balancing System

As a consequence of the balancing behaviour, the computation load data is gathered and spaced in uniform time intervals. This data is composed of a list of values along the time, and these successive values are naturally ordered according their time. Because of this list's particular characteristic of presenting a natural temporal ordering of its values, the sequence of loads can be interpreted as a time series whose values oscillate along with the time. Based on this assumption, techniques for processing and analyzing time series are applied to the list of computational load values of each node to generate smoothing or provide certain prediction. Therefore, one of such techniques can be employed to obtain a more steady current load value and to define the load characteristics for a determined future.

Observing the load data sample as series of averages of data subsets, the moving average model arises as a technique that presents low processing cost with reasonably useful smoothing and prediction. The moving average model offers a simple method for identifying the characteristic variation behaviour of the current analyzed value, which is linearly dependent on its prior consecutive data values. The model is regularly employed with time series to obtain a more consistent value out of short term oscillations and to identify trends or cycles for longer term periods, which also facilitates the filtering of noise generated by abrupt load oscillations. The simple moving average is one technical analysis of unweighted average of determined data values. For this technique, all values are interpreted similarly, and they equally influence the average's calculation. In this case, the average is a central value that is obtained from the past and future values in the chosen subset of the data sequence without connection to time; this average can be proportionally influenced by old values, and it does not describe any trend in the subset, which impedes the estimation of predictions.

On the other hand, an weighted moving average allows to assign different importance to values in a series of a subset of data points; such importance is set to change arithmetically along with the elements in the series. In particular to the case of time series, the importance can be defined to decrease according to the distance from determine time. For a long subset of elements, the past values exercise substantial influence on the current data value. In order to lower this strong influence but still keep it on the final

calculated smoothing average, the calculation of an exponential weighed moving average (EWMA) is employed. With the EWMA, exponentially decreasing weights are assigned to the past values, so in a sequence they are all taken into account even with negligible influence.

Based on the EWMA, three exponential smoothing techniques [52] can be used to determine load predictions: single, double, and triple exponential smoothing. The first one offers a simple smoothing of the data, totally based on the EWMA. In this technique, the past observations are weighted proportionally equally, so the exponential smoothing assigns exponentially decreasing weights over time, as in a geometric progression. However, there is no extrapolation of the data based on the current and the past data samples. The double exponential smoothing incorporates the concept of trend in the smoothing calculation. Trends are highly relevant since they allow the identification of a tendency in the computational load and enable the production of forecasting. The triple exponential smoothing (Holt-Winters) adds the seasonality (periodicity) to the previous two smoothing technical analysis. In this case, the calculation reflects seasonal fluctuations of the data sample, which obey particular patterns during a fixed, determined period of time. Since this smoothing requires the before-hand knowledge about the oscillation patterns in and the time length of a seasonality, this last smoothing technique is not used for obtaining predictions in the balancing scheme.

The double exponential smoothing is applied on the gathered load data to produce certain forecasting based on a short-term load trend. The smoothing is represented by Formulas 6.30 and 6.31. The associated formulas define a smoothed value based on the current load $elem_i$ and the previous smoothed value sum_{i-1}, which partially contains the history of all previous load values. The trend is added as the factor obtained from the second formula; this factor is a exponentially decreasing weighted moving average based on the differences between smoothing values and their previous smoothing values. The trend allows to extrapolate the range between the current load value and the previous smoothing and provides means to determine a future tendency in the observed data. In Formula 6.32, a forecasting is calculated based on the current smoothing value and its respective trend factor. The prediction of load variations is given by m, which represents the number of future time intervals for the predictive projection.

$$sum_i = \alpha \times elem_i + (1 - \alpha) \times (sum_{i-1} + t_{i-1}), 0 \leq \alpha \leq 1 \tag{6.30}$$

- $elem_i$: current computational load value;
- sum_{i-1}: load smoothed in the previous analysis;
- t_{i-1}: trend estimation.

$$t_i = \gamma \times (sum_i - sum_{i-1}) + (1 - \gamma) \times t_{i-1}, 0 \leq \gamma \leq 1 \tag{6.31}$$

- t_{i-1}: trend estimation in the previous analysis;
- sum_i: load smoothed in the current analysis;
- sum_i: load smoothed in the previous analysis.

$$f_{m+i} = sum_i + m \times t_i, m > 0 \tag{6.32}$$

- sum_i: load smoothed in the current analysis;
- m: number of future balancing cycles for smoothing extrapolation;
- t_i: current trend estimation.

The smoothing method is used for any $i \geq 0$, so the initial smoothing is set as $sum_0 = elem_0$ because of the lack of previous smoothing values or at least one previous element to support the calculation. The calculation of the load trend requires at least two past smoothing values, so the initialization of the formula parameters comprehends assigning 0 to t_0 and performing $t_1 = elem_1 - elem_0$. Moreover, both smoothing formulas are delimited by two constants (α and γ); both smoothing and trending constants are set to work in conjunction.

6.5 Experimental Results

The extended balancing scheme are designed to improve the distributed balancing scheme in general or specific load distribution situations, depending on the aspects which are observed in each case. In order to evaluate these balancing extensions, experiments have been conducted in scenarios for each balancing scheme to be compared with the distributed balancing scheme described in Chapter 5. For the experiments, a general test bed was used for all of them; the test-bed was slightly modified for the analysis of certain schemes, based on their evaluation requirements. Intending to produce the same experimental scenario a defined in 4.7.1 for comparison purposes, two clusters of computing servers inter-connected through a fast-Ethernet link were used as common experimental environment in all experiments. The simulation scenario and synthetic load described in 4.7.1 also were replicated and slightly modified in some evaluations.

For all experimental analyses, the results were generated from the calculation of an average of 10 runs for each plotted point in the graphs. The averages presented a confidence interval with confidence level of 95% based on a Student's distribution [51, 46].

6.5.1 Evaluation of the Delay-Based Balancing System

In this experimental analysis, the delay-based redistribution scheme's effectiveness is compared with the distributed balancing system. In order to conduct the experiments, an additional computing server contained an Intel i7 CPU with 8 cores and 6 gigabytes of RAM. This external computing server was located in a different network, so it was connected to the Dell cluster through a fast-Ethernet network link and to the IBM cluster through either a 10 base T network link or a fast Ethernet network link. The difference on link speeds between the external computing server and the clusters were intentionally introduced to generate a known, controlled bottleneck in the simulation system.

All the simulation federates were evenly placed on the 56 computing servers of both clusters. The HLA RTI executive was deployed on the external computing server and coordinated all the experimental simulations. The proposed balancing systems were also placed on each computing server of both clusters. A LMA was deployed in each cluster computing server, and a GM was running in each cluster management node.

For the simulation scenario used in the experiments, a modification on the common

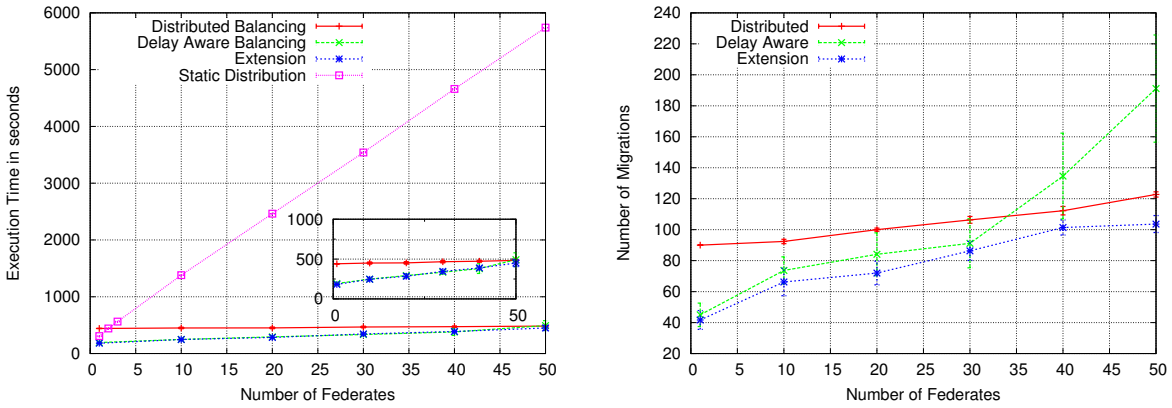
testbed is added. The communication synthetic load is altered, only communication load is produced in the system, without any interference from the computational load, which was negligible. Generally, 200 federates composed the simulations and organized 1 to 10 tanks each in 100 time steps. In order to control the communication imbalances, the tanks updates were reduced to around 700 bytes while special objects were introduced in some federates to apply a 54000-byte load, producing considerable communication overhead.

The experiments were divided in two study groups. In one group, the 10 base T network connection dictated the simulation performance due to the simulation dependency characteristics. The connection with the smaller bandwidth intentionally generated a bottleneck for the distributed simulation when it was employed in the experiments. In the other group, an external background communication load was inserted in the communication link between the management node of the Dell cluster to generate a communication imbalance. The load consisted of a highly intensive consumption of network resources by running a client process that constantly transmitted and received large amounts of data to a server process.

6.5.1.1 Imbalances Caused by Network Resources

A static, restricted communication imbalance was introduced in the experimental simulations in order to observe the responsiveness of the proposed balancing schemes. As shown in Figure 6.5a, every simulation federate controlled one object (tank). Some federates were selected to additionally coordinate the update of special objects, which generated considerable communication overload in the system. The curves in the graph show the influence of the communication delay on simulation performance as the number of communicative federates increased in the simulations. The static distribution presented the worst simulation performance with the induced communication imbalance since it did not modify the distribution according to the simulation communication behaviour. This linear, fast execution time increase was a consequence of the severe bottleneck introduced between the IBM cluster and the external computing server. All the proposed and distributed balancing systems presented curves that grew similarly. However, observing the inset graph in Figure 6.5a, the proposed schemes showed an improvement, which achieved almost double the performance improvement when compared with the

distributed approach for some cases.



(a) Performance Comparison

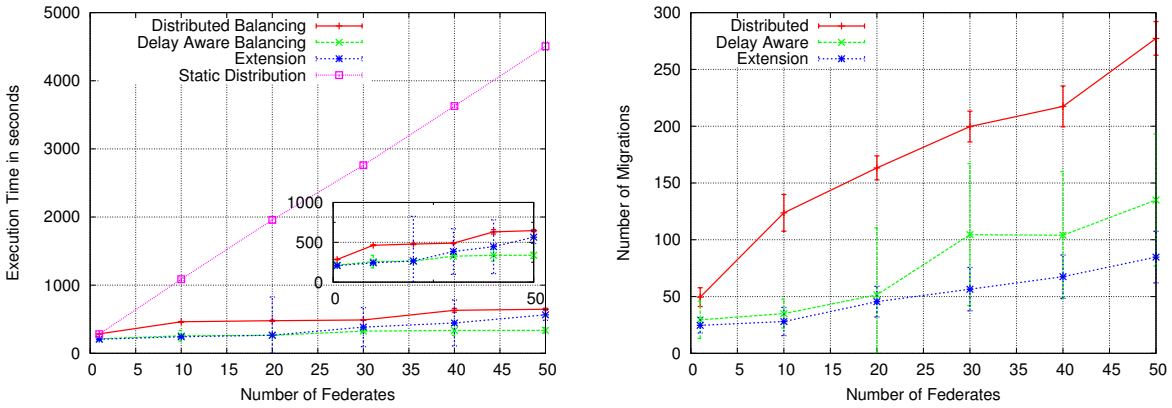
(b) Comparison of Number of Migrations

Figure 6.5: Balancing Scheme Analysis for Increasing Amount of Static Communication Load in Presence of a Network Imbalance

For the same simulations, Figure 6.5b describes the number of migrations required by each balancing scheme to obtain a simulation performance improvement. The proposed balancing system presented less migrations than the distributed balancing system for simulations with up to 30 federates. This steep increase of the number of migrations originates with the instability of the proposed scheme and is caused by the measured communication delay oscillations. When observing the results of the extension approach, there was a significant decrease in the number of migrations with the improvement of the precision of the balancing. The large amount of migrations of the distributed approach was a result of the inter-relations between balancing components in the distributed scheme that constantly moved federates between domains, and a large effort was spent to achieve such a distribution. Even with this simulation scenario that does not impose large peer-to-peer data transfers for federate migrations, the large amount of migrations noticeably influenced the final simulation performance result.

In this experiment, the same static, restricted communication imbalance was imposed to the distributed virtual simulations, but the simulations generated a communication load that changed dynamically. In this case, the responsiveness of the balancing schemes was analyzed since the dynamic imbalances require fast, effective redistribution of simulation elements. As depicted in Figure 6.6a, the proposed balancing schemes were more

effective in reducing the simulation execution time, and the extended scheme was able to outperform the delay-based redistribution for simulations with more than 20 communicative federates. The extension also showed a reduced amount of oscillations when compared to the regular scheme, as described by the standard deviation of the curves. These oscillations are more evident in the curves that represent the number of balancing migrations in Figure 6.6b. According to the curves, both proposed schemes increased the balancing efficiency, since they reduced the amount of migrations to achieve better performance gain. Nevertheless, when comparing the extended and regular approaches, the extended approach presented significantly smaller variations in its curve. This conforms with the decrease of communication delay oscillations and consequently the improvement of balancing precision.



(a) Performance Comparison

(b) Comparison of Number of Migrations

Figure 6.6: Balancing Scheme Analysis for Increasing Amount of Dynamic Communication Load in Presence of a Network Imbalance

6.5.1.2 Imbalances Caused by External Communication Load

An experimental scenario containing external background communication load was produced in order to analyze the responsiveness of the proposed balancing schemes when compared to the distributed balancing scheme. In this scenario, both clusters were connected to the external computing server through the same fast Ethernet network link. In this case, the Dell cluster presented slightly better communication conditions since it contains a Myrinet optical switch connecting its internal computing servers. However, a

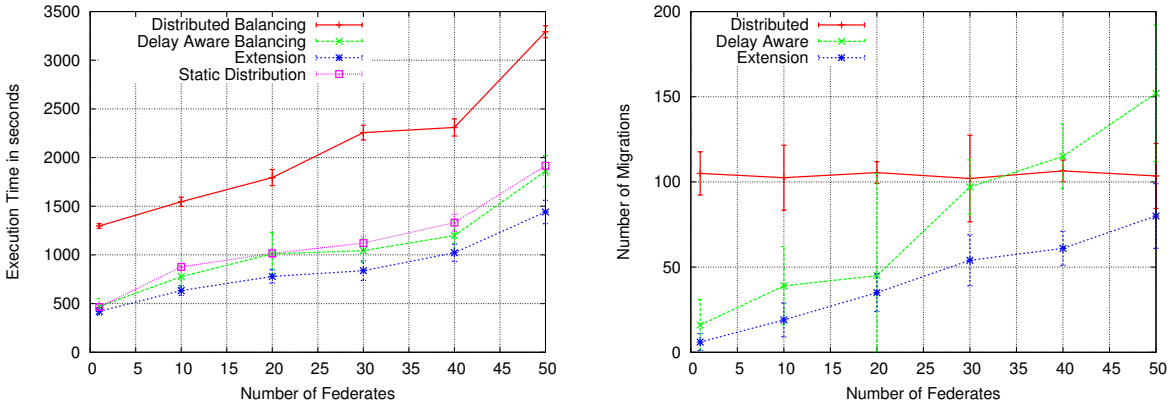
process was placed in one of the computing servers of the Dell cluster in order to produce excessive communication load. This load was applied for λ amount of time (180 seconds) and then was kept deactivated for γ (40 seconds). This activation and deactivation behaviour, together with the dynamic simulation load changes, was introduced in the scenario as an attempt to mimic the unpredictability of a real shared network, in which any application might be consuming the resources at any moment. Thus, the communication load dynamically generated imbalances and overhead for the distributed simulations, and the reaction of the balancing schemes were observed.

As shown in Figure 6.7a, the arrangement of simulation load defined by the distributed balancing approach presented a worse simulation execution time than the simulations with just the static deployment. This situation was generated by the unawareness of the distributed scheme of the external background load of this balancing system. The external background process produced some communication overhead that stressed the network, but the situation was worsened when the distributed balancing placed the most communicative federates in the network bottleneck. The delay-based balancing scheme did not present much performance gain to the simulations since the communication delay oscillations misled the balancing scheme to generate too many migrations. The extended scheme was able to decrease the simulation time, but it offered only a slight performance gain. The curves in Figure 6.7b fairly represented the unawareness of the distributed scheme, which did not modify its reaction to the external communication load. The regular proposed scheme showed a growing, varying amount of migrations as the number of federates increased; this reflects the instability of the balancing due to the variations in its metrics.

6.5.2 Evaluation of the Migration-Aware Balancing System

In this balancing evaluation, experiments have been conducted to evaluate the effectiveness of the migration-aware balancing scheme, as well as its extension, observing the balancing efficiency gain when improving simulation performance. In the experiments, the proposed migration-aware balancing scheme, the extended version of the proposed approach, and the distributed balancing scheme are compared using the testbed.

As a slight modification on the experimental scenario, additional objects were incorporated into federates in order to increase their execution state size, but these objects



(a) Performance Comparison

(b) Comparison of Number of Migrations

Figure 6.7: Balancing Scheme Analysis for Increasing Amount of Static Communication Load in Presence of External Background Overhead

did not participate in the simulation model for the purpose of minimizing the influence of communication load and emphasizing aspects of computational load. The number of these additional objects assigned to each federate were 1, 200, 400, and 600. Since the network resources provided a high bandwidth connection for data transfers, even a high number of objects exerted little influence on the migration latency. Thus, a latency proportional to the number of objects was also added to the migration process to simulate the transmission in high latency networks.

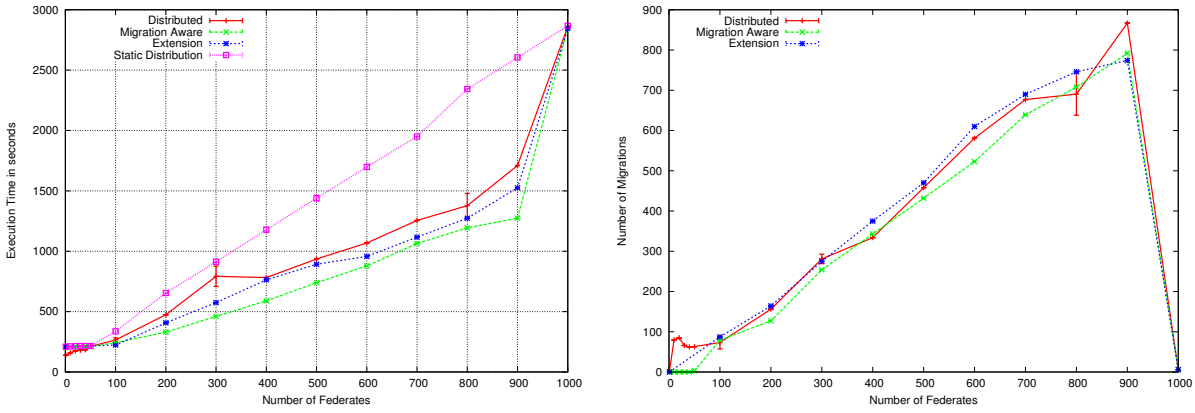
6.5.2.1 Experiments with Static Simulation Load

As described previously, the simulation in this study case scenario presents static load that constantly consumed the CPU resources with the same intensity. In this set of experiments, the efficiency of reacting to static imbalances is evaluated by comparing the distributed, migration-aware, and extended balancing schemes with a base line, initial even static distribution of load. Also, in this case study, four different migration delays are applied on the system in order to analyze the efficiency of detecting delays and reorganizing simulation load distribution by the migration-aware approaches.

As shown in Figure 6.8a, the three balancing schemes were able to present similar performance gains for the simulation. They also showed approximately the same amount of migrations required to reach simulation performance improvement, as described in

Figure 6.8b. Since the core algorithms of proposed balancing schemes are based on the distributed balancing approach, it was expected that they would show simulation performance improvements with close values in simulations that have little migration overhead. However, in Figure 6.8a, migration latency was increased by adding 199 objects to the migration transfer, and the distributed balancing scheme presented a significant performance loss due to the migration delays when comparing its curve to the imbalanced static distribution's curve for simulations under 500 federates. Consequently, the discrepancy in simulation performance gain increased between the migration-aware approaches and the distributed approach, presenting a maximum difference of 49% in simulation time. This improvement in performance was confirmed in Figure 6.9b, which depicts the amount of migrations in which simulation performance improved. In this case, where the simulation performance gain reached the maximum difference between the distributed and extended migration-aware techniques in the graph, the extended migration-aware technique presented a 69% decrease in the number of migrations. This significant improvement in the balancing efficiency of the extended scheme outperformed the migration-aware scheme due to the modifications in the migration delay analysis, such as allowing more parallel migrations, whose delays did not sum up to result in an influence on simulation time. A particular balancing behaviour was also evidenced in both Figures 6.8 and 6.9, as well as any other experimental result; when the experiments reached 1000 federates, all curves in any simulation performance analysis graph converged to a single point, and all curves in any number-of-migration analysis graph showed zero or nearly zero migrations. This specific behaviour developed from the saturation of the distributed environment; the federates which were deployed on shared resources caused very high overload on the entire environment, and no migration move was able to improve the simulation performance.

As the migration latency introduced in the federate transfers increased, the difference in performance gain between the migration-aware and distributed balancing schemes grew. As shown in Figures 6.10a and 6.11a, the distributed balancing technique only caused overhead to simulations with any number of federates. This was a result of its lack of awareness of migration latency because its redistribution algorithm continued to perform migration as the migration delays were nonexistent. Figures 6.10b and 6.11b highlight the performance loss introduced by the distributed technique; it produced a larger number of migrations, generating 5.04 times more migrations for 400 objects and



(a) Performance Analysis

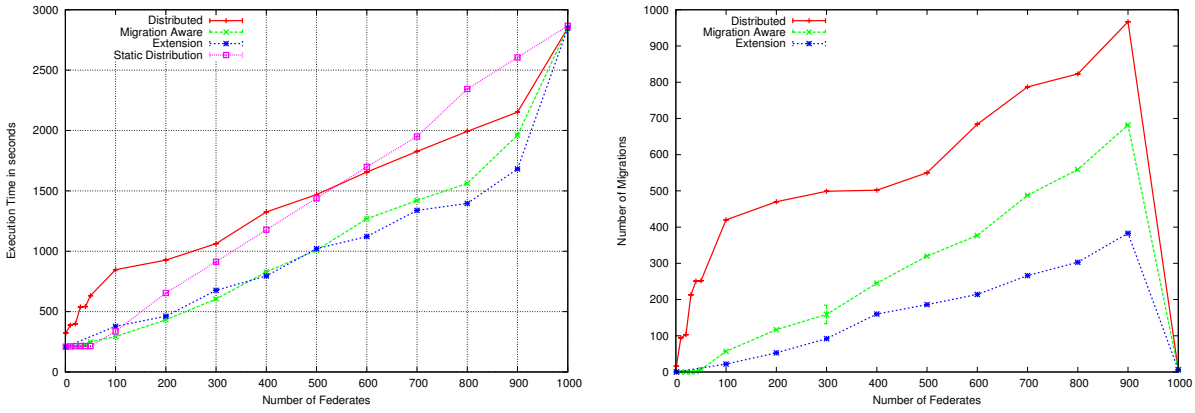
(b) Number of Migrations

Figure 6.8: Performance Gain Analysis for an Increasing Number of Federates with Static Load and Migration Latency of 1 object

8.84 times more migrations for 600 objects with simulations of 700 federates. When comparing the extended approach to the original migration-aware technique, the extended approach was able to improve balancing efficiency: to decrease simulation time and the number of migrations. For the 400-object migration scenario, the extended version could improve performance gain 22.6% with a reduction of 29% in the number of migrations, and for the 600-object migration scenario, the extended version produced 22.8% performance improvement and a 27.2% reduction in migrations. These results show that the extension for the migration-aware scheme could improve the balancing efficiency, even allowing more migrations to be performed when conducting filtering, as described in Sections 6.2.3.2 and 6.2.3.3. However, the migration-aware scheme showed a trend among the migration analysis graphs (200, 400, and 600); as the amount of migration latency increased between the experiments (graphs), this balancing technique presented a number of migrations that were closer to the extended version of it. The behaviour originated from the migration latency, which was prevented due to its high values, even without the improvements on the load redistribution scheme.

6.5.2.2 Experiments with Dynamic Simulation Load

In this case study scenario, simulations with dynamic load changes were used to evaluate the balancing schemes. Such simulations presented load that oscillated during run-time:



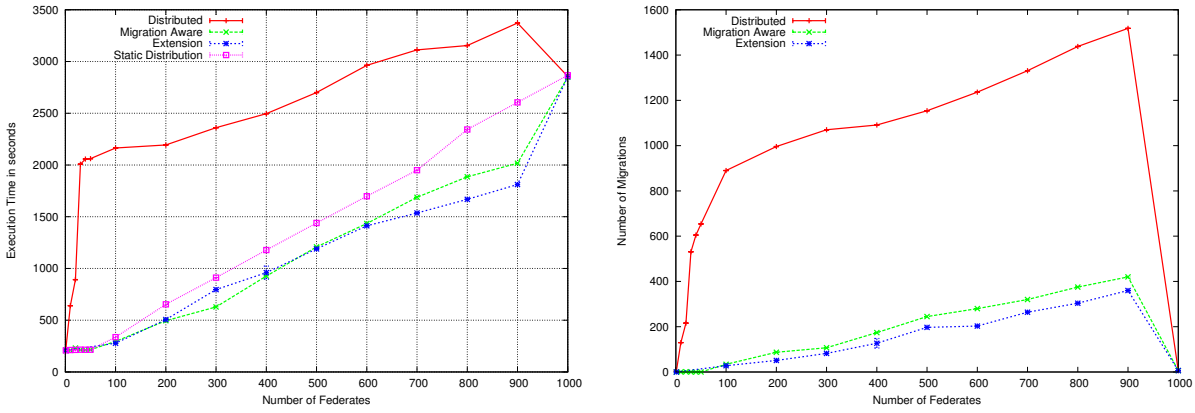
(a) Performance Analysis

(b) Number of Migrations

Figure 6.9: Performance Gain Analysis for an Increasing Number of Federates with Static Load and Migration Latency of 200 objects

totally consuming CPU resources or producing little load. The introduction of dynamic load changes enabled the observation of the balancing response to unpredictable load behaviour. As described in the previous section, the three balancing schemes were evaluated by comparing them with a baseline: a simulation statically deployed and without any dynamic balancing. Four experiments were conducted in this study case; each of them was conducted with a different migration latency in order to analyze the balancing efficiency loss as migration latencies were introduced in the system.

As depicted in Figure 6.12a, the low latency scenario allowed the balancing schemes to perform their redistribution of simulation load as no migration delay existed. The distributed balancing scheme showed a slightly better simulation performance gain: a maximum of 6.1% time decrease when compared with other balancing approaches. It also produced less migrations when reaching similar performance gain: a reduction of 48.8% in number of migrations, as described in Figure 6.12b. This occurred due to the enforcement of producing essential migrations in the migration-aware balancing system when migration delays did not influence the performance gain. However, when a 200-object latency was introduced in the federate migrations, this favourable scenario changed, as shown in Figures 6.13a and 6.13b. The distributed approach presented a decrease in efficiency; it performed a larger number of migrations, 77.7% more migrations, and resulted in worse simulation times for all simulations. The migration-aware approach also



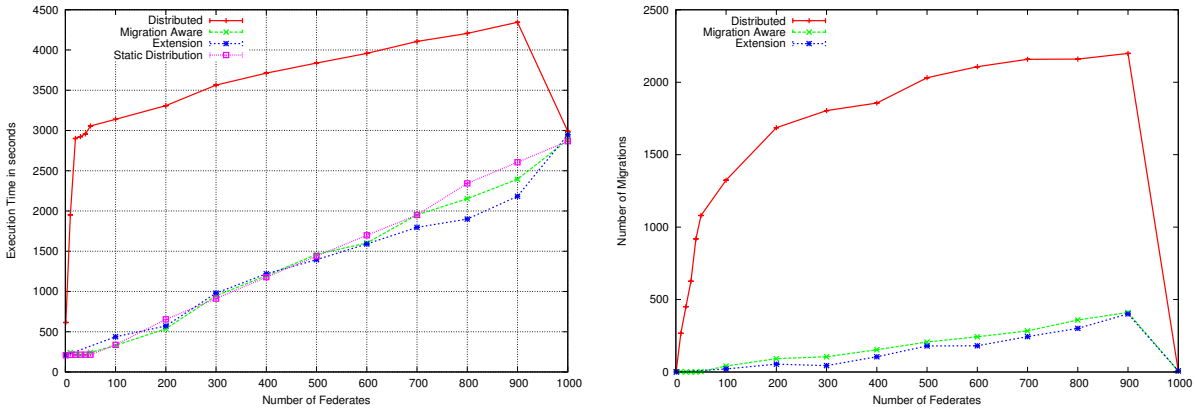
(a) Performance Analysis

(b) Number of Migrations

Figure 6.10: Performance Gain Analysis for an Increasing Number of Federates with Static Load and Migration Latency of 400 objects

introduced performance loss to simulations through a larger number of migrations, resulting in a similar amount of migrations as the distributed approach. The extension still provided performance gain to the simulations with a decrease in number of migrations when compared with the other two approaches. The improvement in efficiency of the extended scheme was achieved through formulas that calculate performance, which produced a lower value that increased as the imbalances grew.

In the simulations described in Figures 6.14a and 6.11a, all the balancing systems were unable to produce any simulation performance gain. The migrations in this case generated substantial time overhead on the system and any modification on the load distribution was useless due to the time spent transferring migrating objects. The distributed balancing scheme, unaware of the migration delays, caused a large performance loss in the simulations; this can be observed in the large amount of migrations produced, as detailed in Figures 6.14b and 6.15b. The migration-aware scheme also generated overhead in the simulations instead of improvements in execution time, but the loss it introduced was much smaller than the distribution approach's loss: 23% smaller for the 400-object scenario and 43% smaller for the 600-object scenario. The extension also did not provide any improvement, but the overhead produced for the simulation time was much smaller than the original migration-aware technique: 21.1% for the 400-object migrations and 21.7% for the 600-object migrations. These cases resulted from the pre-



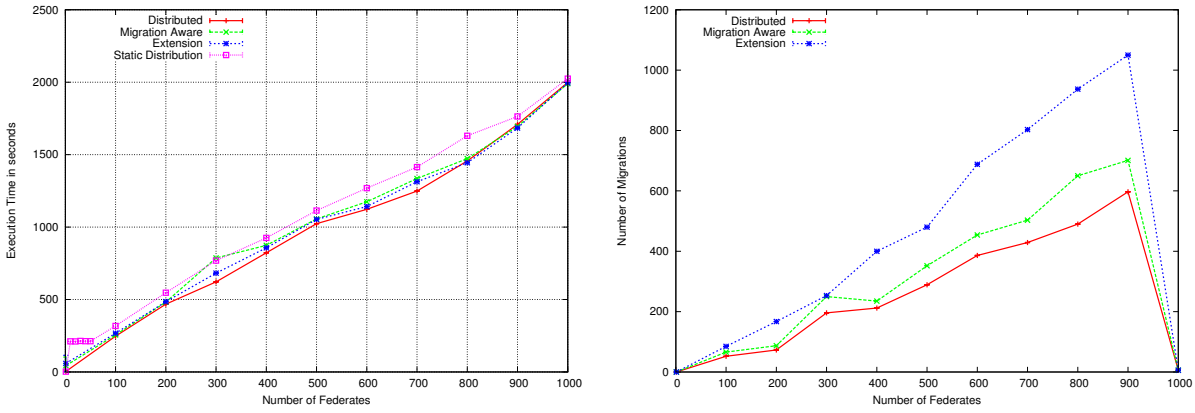
(a) Performance Analysis

(b) Number of Migrations

Figure 6.11: Performance Gain Analysis for an Increasing Number of Federates with Static Load and Migration Latency of 600 objects

vention of costly migrations: 29% less in the 600-object migrations. However, for the 400-object scenario, the migration-aware approach produced less migrations than the extension did. In this case, the extension enabled the creation of a group of migrations that produced less overall migration delays to simulations. Using the concept that migrations have a high probability of occurring in a parallel manner, the extended technique enabled migrations that could achieve performance improvement even with considerable migration delays.

The migration-aware balancing scheme and its extension were able to improve simulation performance in such dynamic load scenarios. The results showed substantial improvement due to the chosen case study, which had an emphasis on migration delay. In experimental scenarios with simulations composed of federates with different migration delays, the schemes are still able to detect the delays and produce load redistribution properly. However, the overall gain provided by the balancing systems is in this case similar or slightly less than the described results because of the smaller influence of migration delay on simulation time. In another simulation scenario, federates with dynamically-changing migration delays can also be detected; however, because of the varying migration time, the estimations might be misled proportionally to the frequency and intensity of delay variations: increasing or decreasing the balancing responsiveness.



(a) Performance Analysis

(b) Number of Migrations

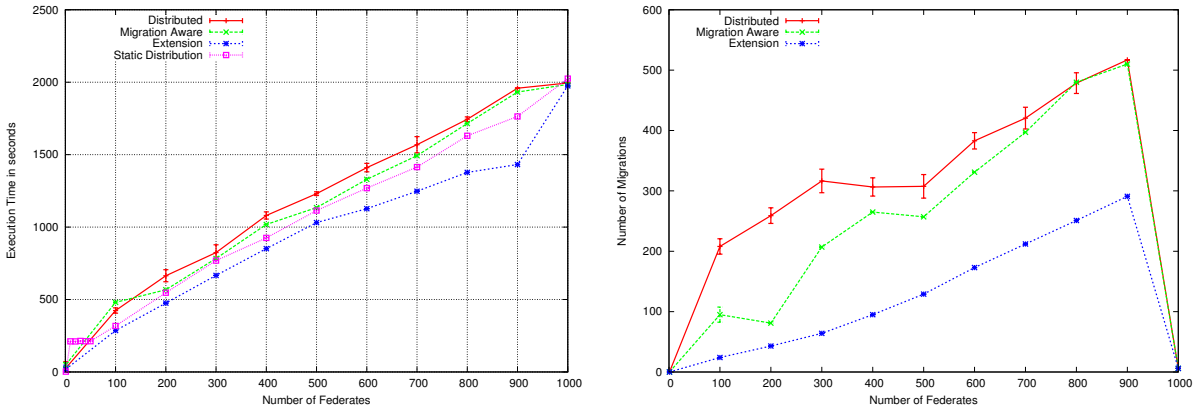
Figure 6.12: Performance Gain Analysis for an Increasing Number of Federates with Dynamic Load and Migration Latency of 1 object

6.5.3 Evaluation of the Self-Adaptive Balancing System

In this efficiency and effectiveness analysis, the self-adaptive load balancing system is evaluated through a series of simulation experiments that compares the centralized and distributed balancing schemes with the adaptation technique applied on the distributed scheme. As configurations to the adaptive scheme, different history sizes (6, 10, and 16) for the adaptation system were used in the experiments, and t_{clb} and t_{rsc} received value 3. Moreover, since the the self-adaptive balancing system was first applied on the computational load redistribution algorithm of the distributed balancing scheme, only computational load was observed on the experiments; thus, simulation communication load was reduced to produce negligible load, which caused no interference on computational load analysis.

In this set of experiments, the four experimental scenarios presented in the distributed balancing system in Chapter 5 were realized to show the efficiency of the proposed scheme: static scenarios with increasing number of federates 6.16a and external load 6.16b, and dynamic scenarios with increasing number federates presenting run-time load changes 6.17a and moving external load 6.17b.

As presented in all experiments, the self-adaptive solution was able to provide performance improvement as the distributed balancing scheme did. However, when observing the number of migrations, as depicted in graphs of Figure 6.18, the adaptive solution,



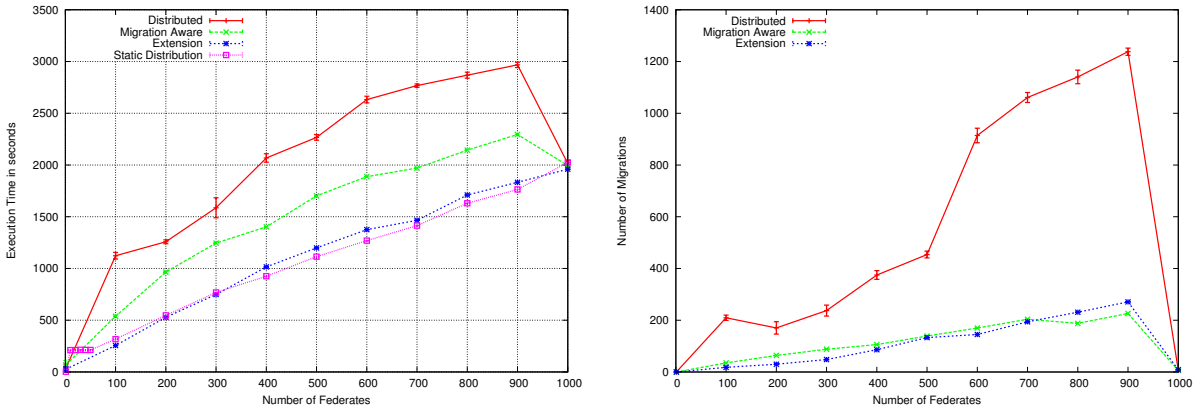
(a) Performance Analysis

(b) Number of Migrations

Figure 6.13: Performance Gain Analysis for an Increasing Number of Federates with Dynamic Load and Migration Latency of 200 objects

with any history size, was able to reduce the number of migrations for the distributed approach. For any case, the number of migrations drastically reduced for experiments with 1000 federates due to the saturation of the distributed system. A deeper analysis showed that history size 16 decreased the balancing responsiveness due to high influence of migration past on the recent moves, as describe in the small number migrations in Figure 6.18a and in the higher simulation time in Figure 6.16a. On the other hand, history size 6 led the balancing to an unstable behaviour due to the high weight to recent past, as delineated in higher simulation times in Figures 6.16b and 6.17a and in the increase in instability in Figures 6.18b and 6.18c. The adaptive solution with history size 10 showed the most stable adjustments in all the experiments since it best adapted the balancing system for the migration latencies and cyclic load oscillations.

The previous experiments showed that performance gain was achieved even with large number of migrations. This particular characteristic resulted from short migration delays, but experiments with large number of federates showed an 9.7-second migration latency average. This latency was hidden by simultaneous migrations and by highly intensive computing load that overlapped migration latencies. Consequently, in the experiments in Figure 6.19, migration time was observed when comparing the distributed load balancing scheme with the proposed self-adaptive solution. Slower migrations can be caused in real simulation by communication distances and federate sate size, but in these experiments,



(a) Performance Analysis

(b) Number of Migrations

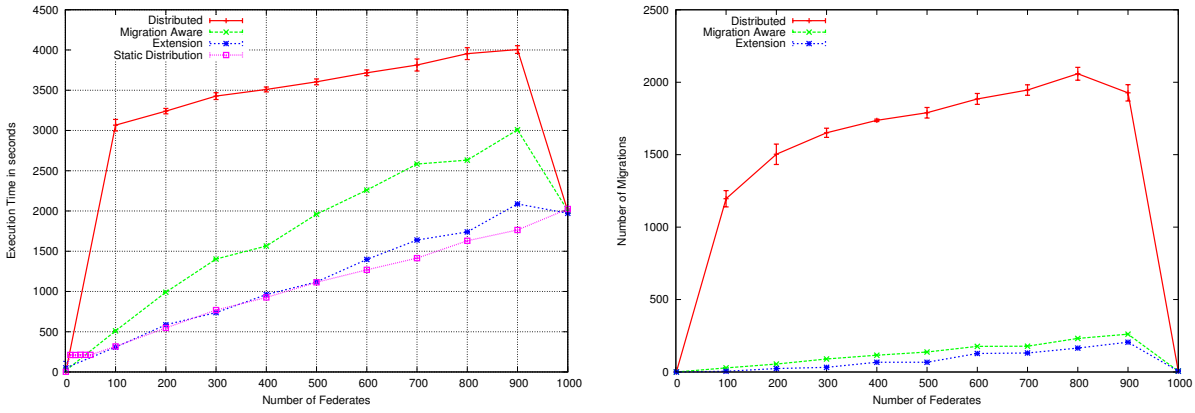
Figure 6.14: Performance Gain Analysis for an Increasing Number of Federates with Dynamic Load and Migration Latency of 400 objects

a delay was intentionally introduce in the second phase of federate migrations. For such experiments, a scenario with static load composed of 500 federates was used. The curves showed that the numerous migrations caused by the non-adaptive scheme influenced the performance negatively when the additional delay exceeded 5 seconds since part of these larger delays was not covered by the long simulation time. Thus, simulations more sensitive to migration latency benefited from the adaptive approach.

6.5.4 Evaluation of the Predictive Balancing System

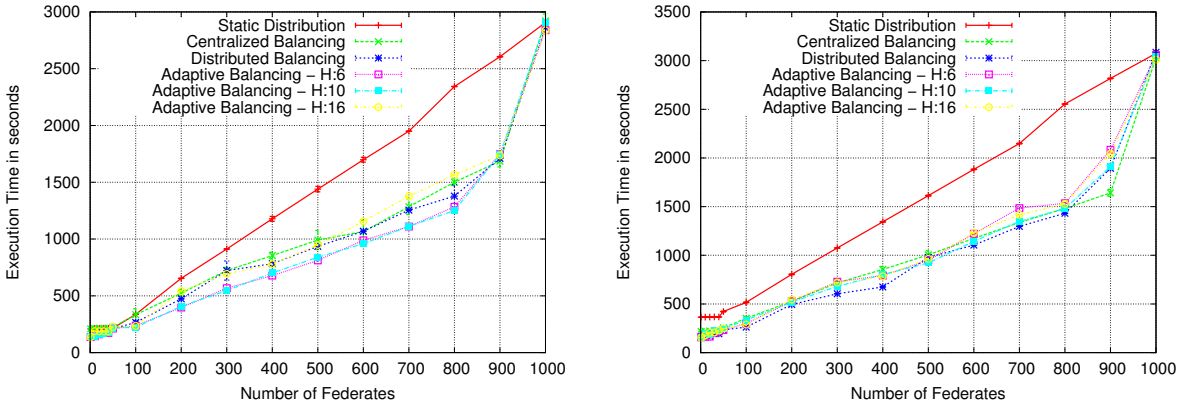
In this experimental analysis, the predictive balancing system was compared with the distributed balancing system. No modifications on the common testbed were needed for this analysis. Only the communication load was set to negligible load values in the experiments in order not to produce any interference on computational load analysis when balancing the distributed simulations.

In this first experimental scenario, federates with static load were deployed on all resources to evaluate the performance gain of the proposed balancing scheme. As shown in Figure 6.20a, both schemes produce similar performance improvement when comparing their simulation execution times with the baseline for an increasing number of federates. However, when the number of migrations of both schemes are compared, a slight difference appears, as depicted in Figure 6.20b. In this graph, the predictive scheme produces



(a) Performance Analysis (b) Number of Migrations

Figure 6.15: Performance Gain Analysis for an Increasing Number of Federates with Dynamic Load and Migration Latency of 600 objects



(a) (b)

Figure 6.16: Performance Analysis for an Increasing Number of Federates and External Background Load

more migrations than the distributed approach. This increase in the number of migrations was expected because of the changes for the predictive redistribution algorithm. By its definition, the predictive balancing is expected to increase the number of migrations, since 2 other migration opportunities (medium and long term predictions) are available for every resource. Even though these additional migrations showed a slight decrease of efficiency regarding the ratio performance gain and number of migrations, they enabled the system to prevent some load imbalances.

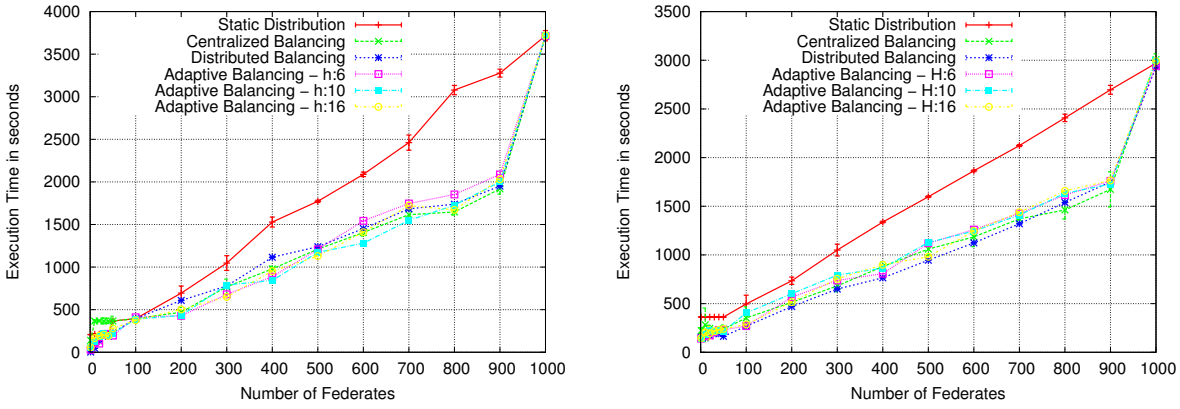


Figure 6.17: Performance Analysis for an Increasing Number of Federates and External Process with Dynamic Load Changes

The experimental test case in Figure 6.21 described the detection and reaction of the balancing system to external background load. This external load comprises an application that constantly runs and intensively consumes the computational capacity of a resource. Both the predictive and the distributed balancing schemes presented very close experimental results related to performance gain, as shown in Figure 6.21a. Since the external application was just adding constant load to the system, the application was not able to cause abrupt load variations. Consequently, both balancing systems could detect the presence of the external load and equally respond to imbalances. As presented in Figure 6.21b, the analysis of the number of migrations also showed the same result as in the previous scenario: the predictive balancing system with larger number of migration moves.

In the last experimental scenario, simulations with load that changed during run-time were employed to evaluate the balancing responsiveness. This behaviour of dynamically changing the load consisted in conditioning simulation federates to produce computational load that varied between high to low intensity on a periodic basis, causing the simulations to exhibit sporadic abrupt load variations. Also, a dynamic external background load was added to the system, contributing to the production of intense load variations in the distributed environment. As depicted in Figure 6.22a and in the presence of abrupt load oscillations, the predictive balancing scheme demonstrated a slight

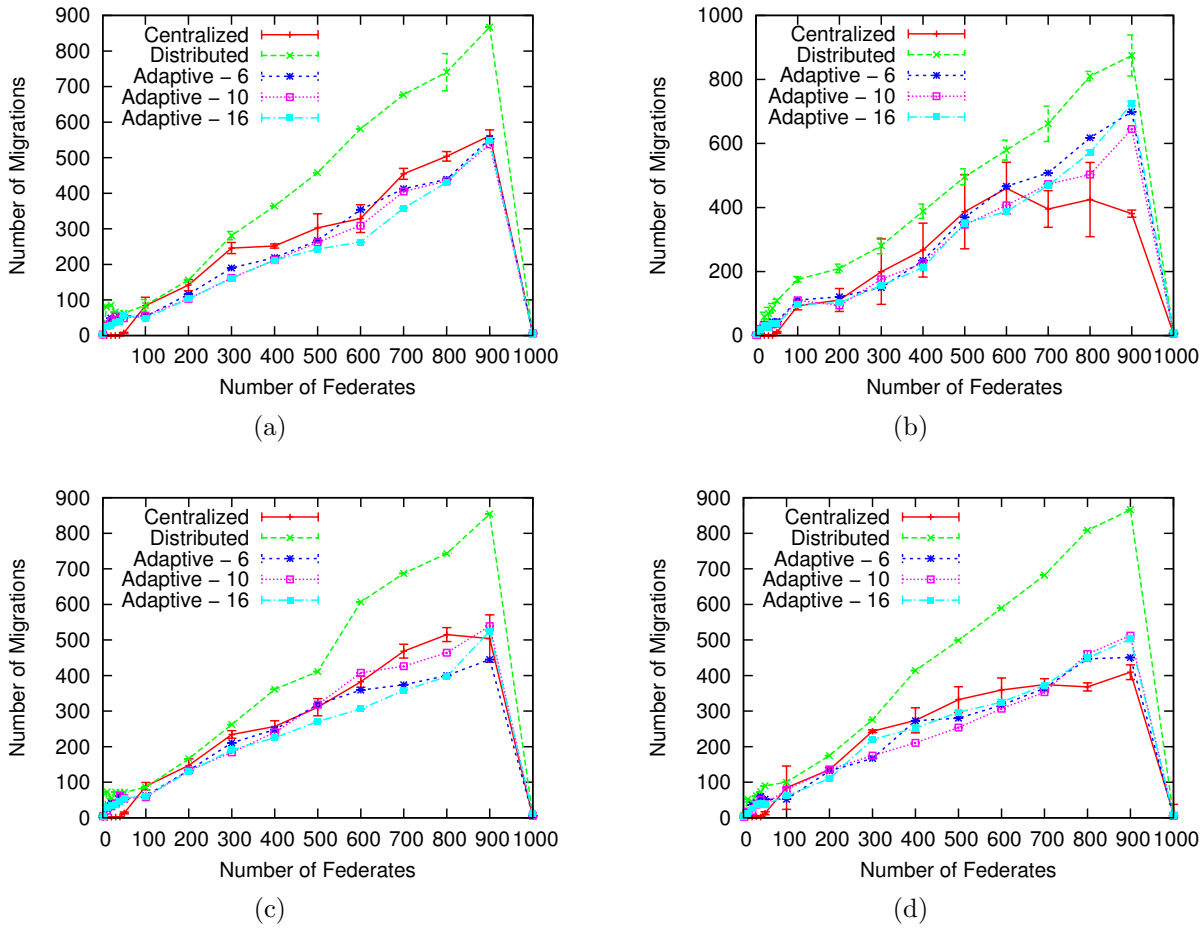


Figure 6.18: Comparative Studies - Number of Migrations versus Comparison of Number of Migrations

decrease in simulation time although the performance gain provided by the distributed balancing system reached a level that is hard to be improved for this scenario, as shown in Chapter 5. This increase of performance is a result of the capacity of the proposed system to detect tendencies in the load variations and smooth the abrupt load changes. The distributed balancing scheme was highly susceptible to these variations, producing many unnecessarily precipitated federate migrations, as shown in Figure 6.22b. Even thought predictive scheme, by design concept, would produce more migrations to enable responsiveness to future trends, the scheme showed a slight decrease in the number of migrations. This difference shows that many unnecessary migrations were avoided during the simulations, presenting a number of migrations larger than the preventive migrations

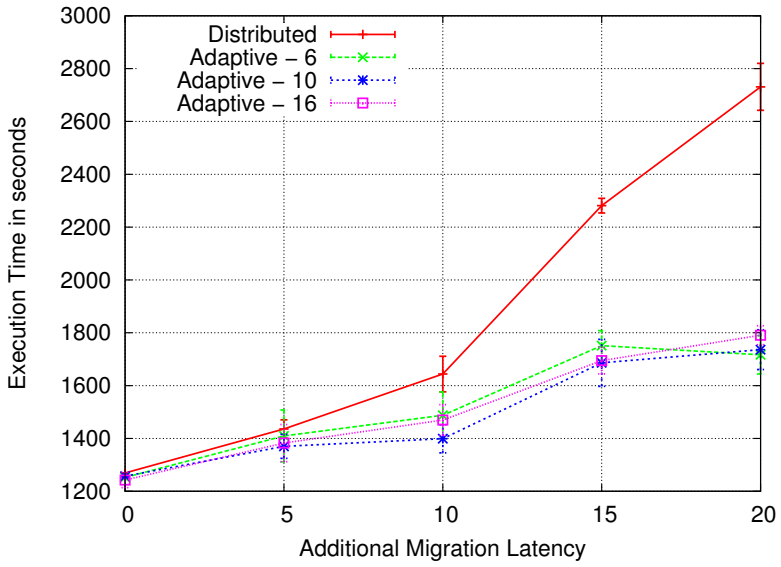
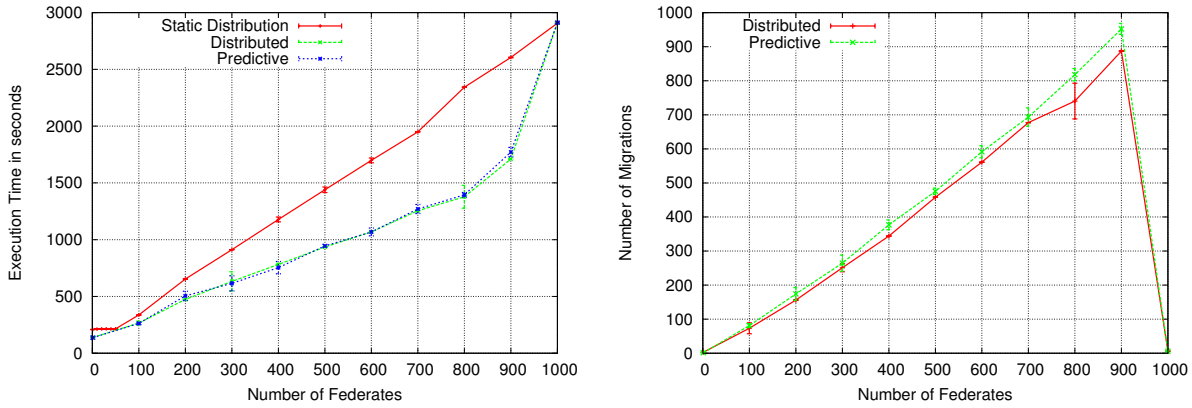


Figure 6.19: Analysis with Increasing Migration Latency



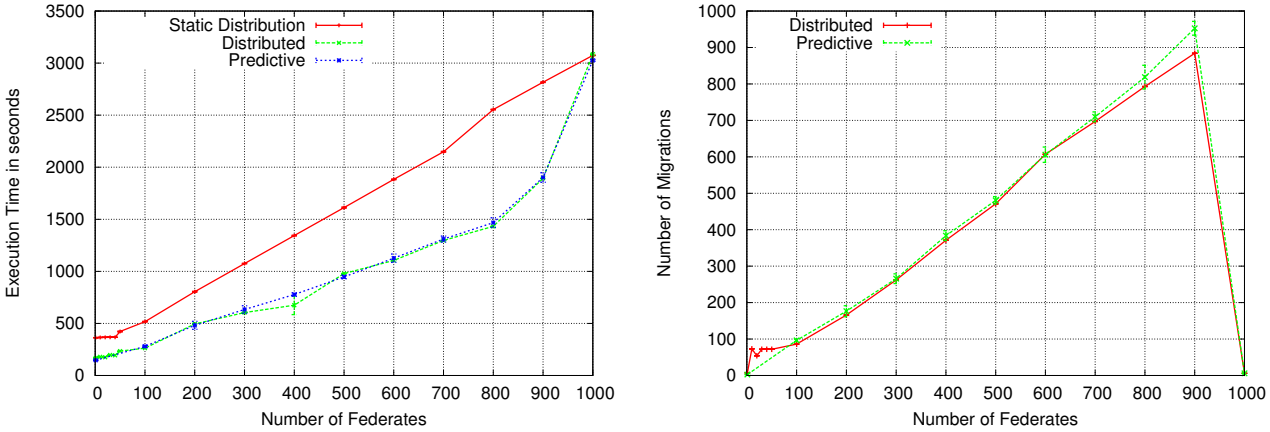
(a) (b)

Figure 6.20: Performance Analysis for an Increasing Number of Federates with Static Load

of the proposed scheme.

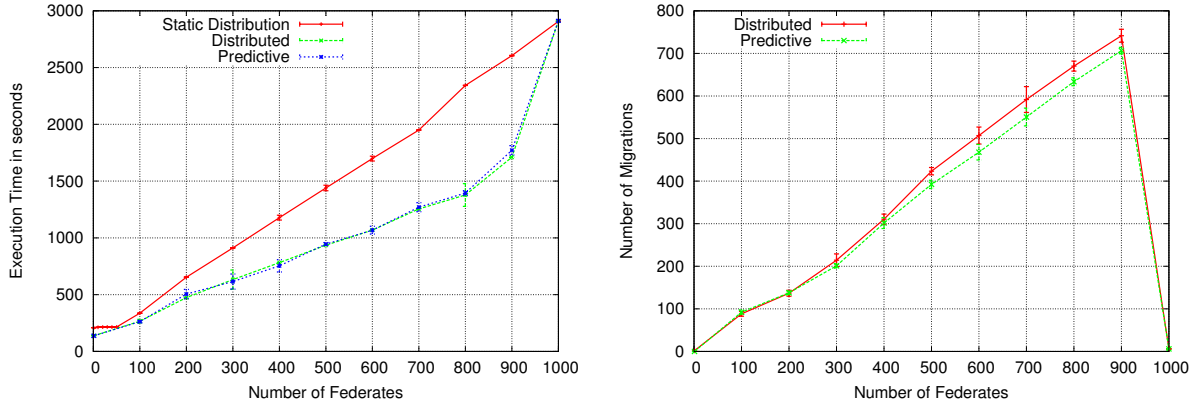
6.6 Summary

Based on the distributed balancing scheme, four balancing extensions were developed. Through the observation of aspects that could lead the distributed balancing system



(a) (b)

Figure 6.21: Performance Analysis for an Increasing Number of Federates and External Background Load



(a) (b)

Figure 6.22: Performance Analysis for an Increasing Number of Federates with Dynamic Load and External Background Process

to loose efficiency, each extension was designed according to a specific balancing situation or behaviour. The balancing extensions thus comprise incorporating analysis of migration latency, migration history, communication delay, and load forecasting in the distributed balancing scheme. The analysis of migration latency enabled the development of migration-aware redistribution algorithm that adds migration delay in the decision-making of the scheme. The migration history metric evidences the balancing efficiency in the previous load moves, which allowed to produce a self-adaptive balancing system. The

awareness of communication delay allowed the part of the distributed balancing scheme that treated communication load analyzed the current conditions of the communication resources before producing federate migration calls. The load forecasting enabled the design a predictive balancing scheme, which used smoothing to avoid precipitated load moves and determined load predictions to prevent future imbalances.

Through the experimental results, each extended balancing scheme proved to be valuable in improving the balancing efficiency in each respective case by decreasing the amount of migration moves to reach better or similar simulation performance gain when compared with the distributed balancing system. In general, the evaluation of the schemes consisted in to compared the reduction of simulation time and the number of migrations of the extension balancing scheme with the distributed balancing scheme. In the communication delay based scheme's evaluation, both balancing techniques reduced the number of costly migrations and provided a method of measuring the actual status load of communication resources. The extended version of the extension improved the precision of the delay-based balancing scheme by decreasing the oscillations of the measured metrics. This increase of accuracy allowed the balancing system to avoid unnecessary precipitated load transfers, reducing the number of migrations. In the migration-aware scheme's evaluation, both schemes, migration-aware and its extended version, were able to react to load imbalances according to migration latencies that were imposed by the experimental distributed simulations. The extended version of the extension also improved the original migration-aware balancing approach by allowing necessary costly migrations or grouping such migrations so their influence on simulation performance would be minimized. In the self-adaptive scheme's evaluation, the adaptation technique of the scheme generated adjustments for the balancing system's parameters to control the system's responsiveness. These adjustments are based on the incoming and outgoing migration moves performed. The experiments showed that the adaptation technique considerably reduced the number of federate migrations without hampering the balancing effectiveness. In the predictive scheme's evaluation, the extension produced more federate migrations in static scenarios when compared with the distributed balancing approach. However, in dynamic simulation scenarios, the predictive solution provided a decrease in the number of migrations and a slight increase in performance. The experimental results in all evaluations demonstrated that the extended load redistribution schemes acted as

expected, but further work and analysis are required in order to improve the performance gain for HLA-based simulations.

Chapter 7

Conclusion

In this thesis, a set of load balancing schemes are presented; such schemes are designed to dynamically redistribute the load of HLA-based simulations. The balancing systems for HLA simulations are needed due to the lack of load imbalance awareness of HLA, which can lead distributed simulations to performance loss. Many balancing systems have been presented aiming to organize the load of distributed simulations. Because of the limitations of such systems and envisioning the need of a balancing scheme that can be properly used for HLA simulations' benefit, the aforementioned schemes have been developed.

As common characteristics in the developed balancing schemes, their balancing process is divided into three sequential phases: monitoring, redistribution, and migration. Their architectural components are placed hierarchically and according to the topological distribution of resources, facilitating the gathering of load status data and the dispatching of migration move calls. Grid Services are employed in the balancing systems as third-party mechanism that provides reliable monitoring information and data transfer between resources. The monitoring information, retrieved from Web MDS tool, feeds the balancing system and enables the detection of imbalances. The data transfer is enabled through the access to Grid RFT, which allows the transfer of data without loss and enables the design of a two-phase migration protocol for accomplishing the redistribution of simulation load.

Federate migration protocol is first designed and implemented in order to enable the development of balancing systems. Without an efficient migration mechanism, the re-

sponsiveness of load balancing schemes to load imbalances is compromised, and they cannot provide or show substantial performance gain improvement for distributed simulations. By using the designed federate migration procedure, a centralized dynamic balancing system is devised. In such system, data from the whole distributed environment and simulation needs to be gathered into one balancing component, which detects imbalances and defines a re-arrangement of load. Due to the drawbacks of this centralized design, to rely on only one component to perform the redistribution algorithm of the balancing scheme, a distributed balancing system is devised. This distributed system inter-connects its components to perform hierarchical balancing in a cluster of resources and distributed balancing among clusters. Based on this distributed design and on the aspects that might lead the design to balancing efficiency loss, extensions are developed. The characteristics added to the distributed balancing scheme are communication-delay awareness, self-adaptation, load oscillation prediction, and migration-delay awareness.

7.1 Summary of Contributions

The contributions produced in the work of this thesis consist in a federate migration protocol and a set of load balancing schemes, which were mentioned previously. The list of contributions is presented by delineating first the development of an optimized federate migration protocol and then the load balancing schemes.

- An optimized federate migration protocol has been designed and implemented. Besides being essential in enabling load balancing systems for HLA-based simulations, federate migration greatly influences the simulations performance in proportion to the time spent on transferring federates. The interactivity of federates in a simulation leads to an inter-dependency among them. Even if the entire simulation does not stop during a federate migration, a set of federates dependent on the migrating federate stops. As a result, federate migration is improved in order to minimize its latency in simulations. The optimized federate migration technique presented in this thesis requires less time and resources to be performed. Based on previous approaches and being simulation freeze-free, the technique offers higher simulation performance by preventing unnecessary processing and message exchange (PJPS events). In order to maintain certain transparency, keep consistency and minimize

the simulations overload, a simulation agent is used in the migration scheme adding to manage federate migration transparently and avoid unnecessary messages and computing. Moreover, based on this optimized migration process, a two-phase migration is developed for the balancing schemes. This two-phase procedure avoids the delay spent to transfer static files for initializing migrating federates and minimizes the influence of third-party tools for this transfer. This technique uses WS GRAM for federate submission to remote resources, Grid RFT for reliable transfer of federate initialization static files, and a Migration Proxy for mediating the transfer of the migrating federate's dynamic execution status.

- A centralized, dynamic balancing approach has been built to provide a dynamic load redistribution scheme that can improve HLA-based simulations distributed in large-scale environments. This scheme considers communication and computational loads in its design and detects heterogeneity and external background load, which are inherent aspects of large-scale shared environments. The proposed balancing scheme consists of a hierarchical design that essentially monitors resources and applications, load reallocation, and load migration. The hierarchical architecture of the scheme minimizes balancing overhead while load redistribution is realized. Balancing involves normalizing load distribution on shared resources and maximizing overall resource utilization. Grid services are employed in the balancing approach to help support the system's proper scaling according to the magnitude of the balanced system. The WS MDS grid service is used to provide monitoring information from the managed resources, and WS GRAM and Grid RFT are accessed to enable federate migration. During the process of data gathering, aggregation techniques are applied on the collected data when sending it up in the hierarchy of balancing elements. The CLB balancing component is designed to apply such data aggregation tools and to perform the whole load redistribution algorithm if it is the root element in the hierarchy.

- A distributed balancing scheme has also been developed to avoid the limitations of the centralized design, such bottlenecks, balancing overheads, and single point of failure. In this distributed scheme, redistribution of both computational and communication load is performed for HLA-based simulations, and its design is based

on the centralized scheme. Thus, the distributed balancing system is aware of heterogeneity of resources, considers external background load, and employs proximity analysis. Due to the limitations of some communication balancing techniques, such as simulation dependency, limitation of simulation parallelism, and lack of awareness of the network topology of shared resources, the proximity analysis is more appropriate for employing in a distributed scheme for balancing communication load of HLA RTI based simulations. This balancing system is organized hierarchically, presenting distributed data gathering and a distributed/hierarchical data analysis. This decentralization of gathering avoids bottlenecks and overheads. The redistribution algorithm is also modified to support hierarchical analysis of load imbalances in local scope and distributed analysis in inter-domain scope. In order to support the inter-domain load redistribution, inter-relation policies are inserted between the interaction between CLBs and enable the proper load transfer between clusters of resources. The policies rule the actions of CLBs in certain region of the resources' topology, defining different neighbourhoods for each single CLB.

- A delay-based balancing scheme has also been designed to constantly measure the load of shared resources and a distributed simulation's communication rates, analyze them, and perform the proper modifications. The design of this scheme aims at a more comprehensive communication balancing system that can minimize the network latencies by employing a more general redistribution analysis. The balancing system considers the proximity of resources to federates' communication destinations and the networking resources' conditions in its decision-making when performing the detection and redistribution of load. Together with the static topological information about the network resources, the environment is also monitored to detect the data exchange delay and redefine the network transmission capabilities for the rearrangement of federates. The system is designed under a hierarchical structure that decreases the overhead caused by measuring and analyzing simulation entities, facilitating the management of simulations. However, the detection technique employed in this scheme does not adapt to abrupt and severe communication changes, which was commonly produced by communicative federates. Due to the magnitude of the balanced simulations, the collected load data sample might hide communication rates that represent an overload of federates. Therefore, ex-

tensions for the detection phase of the balancing scheme are also developed. These extensions improve the detection of the delay-based balancing system by employing modified filtering and analysis techniques, enabling to react properly and achieve a more precise deployment of federates.

- A self-adaptive dynamic load balancing scheme is developed to reduce the number of migrations caused by high responsiveness to load imbalances. This scheme is based on the same balancing functionality and architecture as the distributed balancing approach. However, it introduces an adaptation technique that detects load redistribution irregularities caused by particular characteristics originated from the distributed load or environment. The self-adaptation technique observes the migration history as a metric to identify the effectiveness or efficiency in determining a redistribution of simulation load. The migration history comprises the migration moves performed in the recent past, which are grouped in different scopes: federates, resources, and clusters. The comparison of this metric with the current load status of distributed resources evidences the need of modifications on the parameters that rule the behaviour of the balancing system. With this self-adaptation design, unnecessary and precipitated migration moves are prevented.

- A predictive balancing scheme has been devised to provide means to decrease or prevent precipitated modifications on load distribution. The scheme incorporates prediction metrics and techniques in the redistribution algorithm to enable the system to avoid reactions to abrupt load variations and to allow a preventive detection of some load imbalances. First smoothing allows the balancing system to avoid abrupt load oscillations that might generate precipitated federate migrations. Then, three levels of prediction, short, medium, and long term, are provided to prevent future load variation trends based on the recent past. Based on a prediction model, the proposed system generates load forecasting, which require modifications of detection and load reallocation methods to analyze, identify, and react to significant and potential imbalances. Due to the characteristics of the load samples gathered by the balancing system, the prediction model used in the system is based on time series, which enables the utilization of second exponential moving average to define a trend for the simulation load.

- A migration-aware balancing scheme has been developed to to introduce awareness of migration latencies in the load redistribution algorithms of the distributed balancing system. The scheme incorporates the analysis of migration delays in the decision-making of its redistribution algorithm. For each migration procedure, the amount of time spent to conclude the second phase of the two-phase migration protocol is measured. This time corresponds to the delay a migration procedure introduced into a migrating federate's execution, so the value is used to evaluate the future migration move decisions in the future balancing cycles. The evaluation considers the amount of federates affected by the migration through the calculation of estimations and the benefit in matter of time cause by migration move. An extension to this design is also developed to improve the analysis of cost and benefit of federate migrations for a simulation. The extension avoids the absence of migration delays recorded by the balancing system, increases the number of possibilities in defining migration moves, considers migration parallelism in the analysis, and reformulates the analysis formulas through more accuracy in the calculations and reinforcement mechanisms to enable a pseudo-dynamic time interval parameter.

- Experiments have been extensively conducted to evaluate all the federate migration and balancing designs that have been developed. The results showed the simulation performance gain produced by the developed balancing systems.

7.2 Future Research Directions

The future research work consists in further analyzing and studying the already developed distributed and extended balancing schemes. This analysis might enable the proposal of new extensions based on other aspects that might restrict the balancing efficiency or the improvement of the existing extensions through new methods of increasing precision and efficiency. New paradigms are also intended to be explored in order to allow HLA-based simulations better use shared distributed resources in large-scale environments.

- The delay-based communication balancing scheme is based on thresholds and static parameters to perform the detection and redistribution of simulation federates. The introduction of parameters that adapt according to the current status of communi-

cation resources can make the balancing scheme more adaptable to the volatility of communication load and communication delay. In order to provide such adaptable parameters, further analysis of the scheme will be performed by studying *error*, smoothing α and β, minimum amount of transmitted data, and range selection of representative communication delays in the data sample. Not only modifications on the parameters will be needed, but also alterations on the redistribution scheme will be required to assimilate.

- In order to restructure the distribution of simulation federates, the delay-based scheme needs to perform federate migrations. Besides adding time and delays into the simulation execution, these migrations consume resources, computational and communication. Thus, further research is needed to identify the influence of migrations on the communication delays and produce the correct modifications on the scheme to consider such influences.

- The self-adaptive balancing scheme totally depends on the migration history; more specifically, the analysis of this migration history for dynamically adapting the redistribution algorithm is based on $H/2$ migration history parameter and on a particular migration behaviour in this history. The size of considered history and its analysis substantially determines the adaptation of the balancing scheme; the frequency, defined by the migration behaviour, also greatly influences the reinforcement applied on the balancing system to dynamically adjust the balancing system. Therefore, further experiments and analysis are needed to detect the self-adaptation benefit and efficiency in order to improve balancing efficiency and make the analyses more precise; also, a different adaptation technique will be added to the scheme, so the distributed simulations can obtain better performance improvement.

- The migration delay also exercise influence on the self-adaptive balancing scheme; it determines the reinforcement that is needed in the adjustments. Research work will be performed to incorporate migration awareness into the self-adaptive scheme, so the adjustments are consistent with the migration latency, allowing or restricting migrations.

- For the predictive balancing scheme, other prediction models will be explored to

be used in the scheme and increase simulation performance gain. Methods for directly detecting cyclic load oscillations will be also studied; the oscillations might lead the balancing system to wrongly adapt to certain pattern that does not really corresponds to the real simulation load behaviour. A deeper analysis of the balancing behaviour will be performed to improve reinforcement predictions, calibrate balancing thresholds, and allow the design of dynamic adjustments according to load variations.

- For the migration-aware balancing scheme, additional experiments will be performed to observe the responsiveness of the balancing schemes in presence of simulations containing federates with variable execution state size. These variations in size require further improvement on estimations to determine the correct migration costs. The difficulty in defining Δt still remains due to the complexity in predicting the total simulation execution time, so additional studies are required to define better flexible (variable) time values to determine more precise estimations; the balancing system might measure the reaction of the system distribution and simulation performance to enforce the modifications on such time interval.

- Another future research work is to merge all the current balancing schemes in one scheme to provide self-adaptation and migration-awareness and to use predictions to prevent imbalances. With this merged balancing design, the load redistribution can provide better efficiency in specific situations in which the separated balancing schemes cannot detect. However, the merging of schemes can be a hard task and require considerable efforts to conciliate many different aspects and characteristics with interfering with each other.

- There might be different metrics that can exercise great influence on the simulations' result during run-time. For instance, metrics related to a feedback on the user experience in accessing an HLA-based simulation application interface. Such metrics can really provide a direct assessment about a specific aspect that is examined and delimits goals to the balancing systems; in this case, the aspect observed is the user experience in distributed simulations. In other cases, different aspects are simulation application related and can be studied.

- Other aspects related to the HLA framework that might be useful for balancing

purposes and can be explored to improve the balancing efficiency. Moreover, the balancing schemes can be adjusted to accommodate some other characteristics of or modifications on the HLA framework implementation to improve data gathering through different, additional balancing metrics. For instance, the implementation methods and mechanisms of HLA DDM can provide valuable information about the communication load according to certain situations, which can allow the communication balancing schemes to rearrange load more properly.

- For attending the scalability demand of large-scale distributed simulations, additional studies are required to introduce HLA-based simulations into clouding computing, adding or exporting mechanisms for the management of simulation elements. The cloud computing can provide a flexible solution that enables scaling out and in to attend certain client demand through software as a service, platform as a service, and infrastructure as a service concepts [5]. These characteristics can allow the development of simulation applications that can bear a large need of access through fast resource provisioning.

- The growing interest in devising systems for coordinating the transportation system through vehicular area networks impels and motivates the research on distributed simulations. Research work efforts on developing such simulations can enable the test of transportation systems with recurrent scenarios in distributed environments and can further improve the detection of specific situations provided by run-time analysis of input data from the real world into the virtual environment.

Bibliography

[1] Globus. University of Chicago. 7 Feb. 2008.

[2] Ws mds: Cluster monitoring information and the glue resource property. The Globus Toolkit. 7 Feb. 2008.

[3] Ieee standard for modeling and simulation (m&s) high level architecture (hla) framework and rules. IEEE Standard 1516-2000, September 2000.

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

[5] M. Armbrustand, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53:50–58, April 2010.

[6] J. Aronson, V. Manikonda, W. Peng, R. Levy, and K. Roth. An hla compliant agentbased fast-time simulation architecture for analysis of civil aviation concepts. spring siso simulation interoperability workshop. In *SISO Simulation Interoperability Workshop*. Spring, 2003.

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

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

[9] B. Bali, M. Bubak1, W. Funika, T. Szepieniec, R. Wismller, and M. Radecki. Monitoring grid applications with grid-enabled omis monitor. In F. Riviera, M. Bubak, A. G. Tato, and R. Doallo, editors, *Proceedings of the First European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pages 230–239. Springer, February.

[10] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello. A new adaptive middleware for parallel and distributed simulation of dynamically interacting systems. In *Proceedings of the 8th International Symposium on Distributed Simulation and Real-Time Applications*, pages 178–187. IEEE Computer Society, 2004.

[11] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello. An adaptive load balancing middleware for distributed simulation. In *Workshop on Middleware and Performance (WOMP)*, pages 864–872, 2006.

[12] L. Bononi, G. D'Angelo, and L. Donatiello. Hla-based adaptive distributed simulation of wireless mobile systems. In *Parallel and Distributed Simulation, 2003. (PADS 2003). Proceedings. Seventeenth Workshop on*, pages 40 – 49, june 2003.

[13] L. Bononi, G. D'Angelo, and L. Donatiello. A new adaptive middleware for parallel and distributed simulation of dynamically interacting systems. In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*, pages 40–49. IEEE Computer Society, 2003.

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

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

[16] A. Boukerche and S. K. Das. Reducing null messages overhead through load balancing in conservative distributed simulation systems. *Journal of Parallel and Distributed Computing*, 64(3):330–344, 2004.

[17] A. Boukerche, S. K. Das, and A. Fabbri. Swimnet: A scalable parallel simulation testbed for wireless and mobile networks. *Wireless Networks*, 7:467–486, 2001. 10.1023/A:1016770526188.

[18] A. Boukerche and A. Fabbri. Partitioning parallel simulation of wireless networks. In *Proceedings of the 32nd conference on Winter simulation*, WSC '00, pages 1449–1457, San Diego, CA, USA, 2000. Society for Computer Simulation International.

[19] A. Boukerche and C. Tropper. A static partitioning and mapping algorithm for conservative parallel simulations. In *Proceedings of the 8th workshop on Parallel and distributed simulation*, pages 164–172. IEEE Computer Society, 1994.

[20] A. Boukerche and M. Zhang. Towards peer-to-peer based distributed simulations on a grid infrastructure. In *Proceedings of the 41st Annual Simulation Symposium*, pages 212–219. IEEE Computer Society, 2008.

[21] A. G. Bruzzone, R. Mosca, R. Revetria, E. Bocca, and E. Briano. Agent directed hla simulation for complex supply chain modeling. *Transactions of the Society for Modeling and Simulation International*, 81(9):647–655, 2005.

[22] M. Bubak, M. Malawski, and K. Zajac. Towards the crossgrid architecture. In *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 16–24, London, UK, 2002. Springer-Verlag.

[23] C. Burdorf and J. Marti. Load balancing strategies for time warp on multi-user workstations. *The Computer Journal*, 36(2):168–176, 1993.

[24] W. Cai, S. J. Turner, and H. Zhao. A load management system for running hla-based distributed simulations over the grid. In *Proceedings of the Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, pages 7–14, Washington, DC, USA, 2002. IEEE Computer Society.

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

[26] C. D. Carothers and R. M. Fujimoto. Background execution of time warp programs. In *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*, pages 12–19. IEEE Computer Society, 1996.

[27] C. D. Carothers and R. M. Fujimoto. Efficient execution of time warp programs on heterogeneous, now platforms. *IEEE Transactions on Parallel and Distributed Systems*, 11(3):299–317, mar 2000.

[28] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5):440 – 452, sept. 1979.

[29] C. Chen, L. Tang, X. Feng, and K. Lin. A distributed forest fire fighting simulation system based on hla. In *Edutainment*, pages 1107–1111, 2006.

[30] D. Chen, S. J. Turner, W. Cai, and M. Xiong. A decoupled federate architecture for high level architecture-based distributed simulation. *J. Parallel Distrib. Comput.*, 68:1487–1503, November 2008.

[31] M. Choe and C. Tropper. On learning algorithms and balancing loads in time warp. In *Proc, of the 13th workshop on Parallel and distributed simulation*, pages 101–108. IEEE Computer Society, 1999.

[32] A. Cimino, F. Longo, and G. Mirabelli. Operators training in container terminals by using advanced 3d simulation. In *2010 Summer Simulation Multiconference*, SummerSim '10, pages 64–69, San Diego, CA, USA, 2010. Society for Computer Simulation International.

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

[34] Z. Fanchao, S. J. Turner, and H. Aydt. Symbiotic simulation control in supply chain of lubricant additive industry. In *Distributed Simulation and Real Time Applications, 2009. DS-RT '09. 13th IEEE/ACM International Symposium on*, pages 165–172, oct. 2009.

[35] R. A. Fisher. On a distribution yielding the error functions of several well known statistics. In *Proceedings of the International Congress of Mathematics*, pages 805–813, 1924.

[36] 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.

[37] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In Springer-Verlag LNCS 3779, editor, *IFIP International Conference on Network and Parallel Computing*, pages 2–13, 2005.

[38] I. Foster and C. Kesselman. The globus project: A status report. In *IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.

[39] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[40] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[41] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33:30–53, October 1990.

[42] B. P. Gan, Y. H. Low, S. Jain, S. J. Turner, W. Cai, W. J. Hsu, and S. Y. Huang. Load balancing for conservative simulation on shared memory multiprocessor systems. In *Proceedings of the 14th workshop on Parallel and distributed simulation*, pages 139–146. IEEE Computer Society, 2000.

[43] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*, page 218. W. H. Freeman & Co., New York, NY, USA, 1990.

[44] D. W. Glazer and C. Tropper. On process migration and load balancing in time warp. *IEEE Transactions on Parallel and Distributed Systems*, 4(3):318–327, 1993.

[45] 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.

[46] R. V. Hogg, J. Mckean, and A. T. Craig. *Introduction to Mathematical Statistics*. Prentice Hall, 7 edition, 2012.

[47] J. Huang, Y. Du, and C. Wang. Design of the server cluster to support avatar migration. In *Proceeding of The IEEE Virtual Reality 2003 Conference (IEEE-VR2003)*, pages 7–14, Los Angeles, USA, 2003.

[48] D. R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7:404–425, July 1985.

[49] J. Jiang, R. Anane, and G. Theodoropoulos. Load balancing in distributed simulations on the grid. In *Proceedings of the International Conference on Systems, Man and Cybernetics*, pages 3232–3238. IEEE Computer Society, 2004.

[50] M. R. Jiang, S. P. Shieh, and C. L. Liu. Dynamic load balancing in parallel simulation using time warp mechanism. In *Proceedings of the 1994 International Conference on Parallel and Distributed Systems*, pages 222–229. IEEE Computer Society, 1994.

[51] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume 2. Wiley, 2 edition, 1995.

[52] E. S. Gardner Jr. Exponential smoothing: The state of the artpart ii. *International Journal of Forecasting*, 22(4):637–666, 2006.

[53] C. Kim, T. Lee, S. Hwang, and C. Jeong. Grid-based parallel and distributed simulation environment. *Parallel Computing Technologies*, 2763:503–508, 2003.

[54] T. Lee, S. Yoo, and C. 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. Springer Berlin / Heidelberg.

[55] M. Lees, B. Logan, T. Oguara, and G. Theodoropoulos. Hla_agent: Distributed simulation of agent-based systems with hla. In *Workshop on HLA-Based Distributed Simulation on the Grid in the 4th International Conference Computational Science (ICCS 2004)*, pages 881–888, Krakw, Poland, June 2004. Springer Berlin / Heidelberg.

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

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

[58] 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. IEEE Computer Society.

[59] J. Lüthi and S. Groβmann. Ft-rss: A flexible framework for fault tolerant hla federations. In *Workshop on HLA-Based Distributed Simulation on the Grid in the 4th International Conference Computational Science (ICCS 2004)*, pages 865–872, Krakw, Poland, June 2004. Springer Berlin / Heidelberg.

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

[61] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[62] P. Mehra and B. W. Wah. Physical-level synthetic workload generation for load-balancing experiments. In *International Symposium on High Performance Distributed Computing*, pages 208–217, 1992.

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

[64] R. Minson and G. Theodoropoulos. Distributing repast agent-based simulations with hla. In *Proceedings of the 2004 European Simulation Interoperability Workshop of Simulation Interoperability Standards Organisation and Society for Computer Simulation International*, Edinburgh, Scotland, July 2004.

[65] J. Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18:39–65, March 1986.

[66] K. L. Morse, R. Brunton, J. M. Pullen, P. McAndrews, A. Tolk, and J. Muguira. An architecture for web-services based interest management in real time distributed simulation. In *DS-RT '04: Proceedings of the Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'04)*, pages 108–115, Washington, DC, USA, 2004. IEEE Computer Society.

[67] K. L. Morse, D. L. Drake, and R. P.Z. Brunton. Web enabling an rti an xmsf profile. In *Proceedings of the IEEE 2003 European Simulation Interoperability Workshop*, Stockholm, Sweden, June 2003.

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

[69] K. Pan, S. J. Turner, W. Cai, and Z. Li. A service oriented hla rti on the grid. In *Proceedings of the IEEE International Conference on Web Services*, 2007.

[70] G. Pedrielli, P. Scavardone, T. Tolio, M. Sacco, and W. Terkaj. Simulation of complex manufacturing systems via hla-based infrastructure. In *IEEE Workshop on Principles of Advanced and Distributed Simulation*, pages 1–9, june 2011.

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

[72] J. M. Pullen, R. Brunton, D. Brutzman, D. Drake, M. Hieb, K. L. Morse, and A. Tolk. Using web services to integrate heterogeneous simulations in a grid environment. In *Workshop on HLA-Based Distributed Simulation on the Grid in the 4th International Conference Computational Science (ICCS 2004)*, pages 835–847, Krakw, Poland, June 2004. Springer Berlin / Heidelberg.

[73] I. Rodero, F. Guim, J. Corbalan, L. Fong, and S. M. Sadjadi. Grid broker selection strategies using aggregated resource information. *Future Generation Computer Systems*, 26:72–86, 2010.

[74] K. Rycerz, B. Balis, R. Szymacha, M. Bubak, and P. Sloot. Monitoring of hla grid application federates with ocm-g. In *DS-RT '04: Proceedings of the Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'04)*, pages 125–132, Washington, DC, USA, 2004. IEEE Computer Society.

[75] K. Rycerz, M. Bubak, M. Malawski, and P. Sloot. Interactive grid computing: Adapting high level architecture-based applications to the grid. In P. Doerffer and J. Rybicki, editors, *TASK Quarterly, Scientific Bulletin of the Academic Centre of Gdansk*, volume 8, pages 549–559. TASK Publishing, 2004.

[76] K. Rycerz, M. Bubak, M. Malawski, and P. Sloot. Support for effective and fault tolerant execution of hla-based applications in the ogsa framework. In *Workshop on HLA-Based Distributed Simulation on the Grid in the 4th International Conference Computational Science (ICCS 2004)*, pages 848–855, Krakw, Poland, June 2004. Springer Berlin / Heidelberg.

[77] K. Rycerz, M. Bubak, M. Malawski, and P. Sloot. Grid support for hla-based collaborative environment for vascular reconstruction. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 48, Washington, DC, USA, 2006. IEEE Computer Society.

[78] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. A grid service for management of multiple hla federate processes. In *PPAM*, pages 699–706, 2005.

[79] K. Rycerz, M. Bubak, P. Sloot, and V. Getov. Problem solving environment for distributed interactive applications. In *Proceedings of the CoreGRID Integration*

Workshop, pages 129–138, Krakow, Poland, October 2005. Academic Computing Centre CYFRONET AGH.

[80] K. Rycerz, A. Tirado-Ramos, A. Gualandris, S. F. P. Zwart, M. Bubak, and P. M. A. Sloot. Interactive n-body simulations on the grid: Hla versus mpi. *International Journal of High Performance Computing Applications*, 21(2):210–221, 2007.

[81] 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.

[82] R. Schlagenhaft, M. Ruhwandl, C. Sporrer, and H. Bauer. Dynamic load balancing of a multi-cluster simulator on a network of workstations. In *Proceedings of the 9th workshop on Parallel and distributed simulation*, pages 175–180. IEEE Computer Society, 1995.

[83] J. S. Steinman, C. A. Lee, L. F. Wilson, and D. M. Nicol. Global virtual time and distributed synchronization. *ACM SIGSIM Simulation Digest*, 25:139–148, July 1995.

[84] K. Su and W. Fu. Designing hardware-in-the-loop simulation system for missile with high level architecture. In *International Conference on Mechatronic Science, Electric Engineering and Computer*, pages 1525–1527, aug. 2011.

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

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

[87] 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. IEEE Computer Society.

[88] A. Tirado-Ramos, K. Zajac, Z. Zhao, P. M. A. Sloot, G. D. van Albada, and M. Bubak. Experimental grid access for dynamic discovery and data transfer in distributed interactive simulation systems. In *International Conference on Computational Science*, pages 284–294, 2003.

[89] K. Uchiyama, Y. Iwai, T. Ichinoseki, and K. Kayama. Applying artificial intelligence theory to helicopter saf simulation based on hla/rti (1). In *Computational Intelligence and Multimedia Applications, 2001. ICCIMA 2001. Proceedings. Fourth International Conference on*, pages 210–214, 2001.

[90] C. E. Weatherburn. *A first course in mathematical statistics*. The English Language Book Society and Cambridge University Press, 1961.

[91] L. F. Wilson and W. Shen. Experiments in load migration and dynamic load balancing in speedes. In *Proceedings of the 1998 Winter Simulation Conference*, pages 483–490. IEEE Computer Society, 1998.

[92] P. Wu, G. Cai, D. Zhang, Z. Zhou, and Y. Chen. Hla-based multi-aricraft combat simulation system. In *2nd International Asia Conference on Informatics in Control, Automation and Robotics*, volume 3, pages 331–334, march 2010.

[93] Z. Xiao, B. Unger, R. Simmonds, and J. Cleary. Scheduling critical channels in conservative parallel discrete event simulation. In *Proceedings of the 13th workshop on Parallel and distributed simulation*, pages 20–28. IEEE Computer Society, 1999.

[94] 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.

[95] Y. Xie, Y. M. Teo, W. Cai, and S. J. Turner. A distributed simulation backbone for executing hla-based simulation over the internet. In *Workshop on Grid Computing & Applications, Proceedings of the 2nd International Conference on Scientific and Engineering Computation*, pages 96–103, June 2004.

[96] 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. IEEE Computer Society.

[97] 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. Springer Berlin / Heidelberg.

[98] K. Zajac, M. Bubak, M. Malawski, and P. Sloot. Towards a grid management system for hla-based interactive simulations. In *Proceedings of the 7th International Symposium on Distributed Simulation and Real-Time Applications*, pages 4–11. IEEE Comp. Society, 2003.

[99] 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.

[100] K. Zajac, A. Tirado-Ramos, Z. Zhao, P. M. A. Sloot, and M. Bubak. Grid services for hla-based distributed simulation frameworks. In *European Across Grids Conference*, pages 147–154, 2003.

[101] P. Zhang and G. Gong. Atmosphere-affected flight simulation system. In *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, pages 431–437, San Diego, CA, USA, 2009. Society for Computer Simulation International.

[102] Y. Zhang, G. Theodoropoulos, R. Minson, S. J. Turner, W. Cai, and Y. Xie. Grid-aware large scale distributed simulation of agent-based systems. In *Proceedings of the 2005 European Simulation Interoperability Workshop of Simulation Interoperability Standards Organisation and Society for Computer Simulation International*, Toulouse, France, June 2005.

[103] Y. Zhou and M. Wang. Launch vehicle testing simulation system. In *International Conference on Computational and Information Sciences*, pages 921–924, Los Alamitos, CA, USA, 2011. IEEE Computer Society,.

[104] 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. IEEE Computer Society.

[105] W. Zong, Y. Wang, W. Cai, and S. J. Turner. Grid services and service discovery for hla-based distributed simulation. In *DS-RT '04: Proceedings of the Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'04)*, pages 116–124, Washington, DC, USA, 2004. IEEE Computer Society.