PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

# FReMI – A MIDDLEWARE TO HANDLE MOLECULAR DOCKING SIMULATIONS OF FULLY-FLEXIBLE RECEPTOR MODELS IN HPC ENVIRONMENTS

RENATA DE PARIS

Dissertation presented as partial requirement for obtaining the master's degree in Computer Science from Pontifícia Universidade Católica do Rio Grande do Sul.

Supervisor: Prof. Dr. Osmar Norberto de Souza

Porto Alegre
2012

D278F De Paris, Renata

FReMI – a middleware to handle molecular docking simulations of fully-flexible receptor models in HPC environments / Renata De Paris. – Porto Alegre, 2012.

72 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Osmar Norberto de Souza.

1. Informática. 2. Biologia Computacional. 3. Biologia Molecular. 4. Banco de Dados. I. Souza, Osmar Norberto de. II. Título.

CDD 005.74

**Ficha Catalográfica elaborada pelo**
**Setor de Tratamento da Informação da BC-PUCRS**

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada *"FReMI: A Middleware to Handle Molecular Docking Simulations of Fully-flexible Receptor Models in HPC Environments"*, apresentada por Renata De Paris como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Bioinformática e Modelagem Computacional, aprovada em 12/04/2012 pela Comissão Examinadora:

Prof. Dr. Osmar Norberto de Souza – Orientador — PPGCC/PUCRS

Prof. Dr. Duncan Dubugras Alcoba Ruiz – — PPGCC/PUCRS

Prof. Dr. Aad van Moorsel - — *University of Newcastle*

Homologada em 24/04/2012, conforme Ata No. 009 pela Comissão Coordenadora.

Prof. Dr. Paulo Henrique Lemelle Fernandes
Coordenador.

PUCRS
**Campus Central**
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

# DEDICATION

*Of course there is no formula for*
*success except perhaps an unconditional*
*acceptance of life and what it brings.*

Arthur Rubinstein

# ACKNOWLEDGEMENTS

I don’t have enough words to thank all who helped me to arrive at this stage of my life.

First of all, I would like to thank my supervisor Prof. Osmar Norberto de Souza, for his support, for his guidance and constructive criticism during these two years of my Master. With Prof. Osmar I learn the most important principles to do research and how to appreciate it.

Special thanks are due to Prof. Duncan Ruiz for his valuable and constructive suggestions during the planning and development of this research work. Thanks for his trust in my potential, providing the opportunity to extend my research at the Newcastle University, Newcastle Upon Tyne, Great Britain. My most sincere thanks!

Other special thanks are due to Prof. Aad van Moorsel and Prof. Paul Watson, whom, in the end of 2011, kindly received me in the School of Computing Science at Newcastle University through the EU project ‘Computational Intelligence in Lifestyle-Management Infrastructure’, for their attention and enlightening discussions. I would like to acknowledge and thank Simon Woodman for his precious helps with Amazon EC2. And, extend my thanks to the members of the Trust and Security and Cloud Computing Groups for their helpful discussions, especially Winai Wongthai and Francisco Rocha for their luminous suggestions and appreciation of my work.

I would like to thank all the people who have been, in one way or another, involved in my work. I am particularly grateful to my friend and colleague Elisângela Cohen who gave encouragement, useful critiques. Her biology classes in LABIO were indispensable. Her comments, as well of her husband Marcelo Cohen, on the manuscript were very helpful. Also, I would like thank my friend Karina C. Motta Dall’Agno by her friendliness.

I want to thank my colleague Fabio Frantz for his bright collaboration and cooperation in the last step of the experimental results which were achieved through several discussions and uncountable weekends and holidays executing experiments. I also extend my thanks to the LABIO and GPIN colleagues.

I would like to acknowledge the Programa de Pós-Graduação em Ciência da Computação (PPGCC) at PUCRS and the National Research Council of Brazil (CNPq) for the M.Sc. scholarship.

Last, but not least, my most special thanks to my dear parents, Sérgio and Maria, for their immeasurable support and example of life; my brother Cléber and his wife Nadime for their continuous enthusiasm; and my fiancé Mateus for his personal support and unlimited patience during the many ups and downs of this journey. Thank you ever so much for understanding my absence during many weekends and holidays: you weren’t less important than my work!

# FReMI – A MIDDLEWARE TO HANDLE MOLECULAR DOCKING SIMULATIONS OF FULLY-FLEXIBLE RECEPTOR MODELS IN HPC ENVIRONMENTS


## ABSTRACT

Molecular docking simulations of Fully-Flexible Protein Receptor (FFR) models are coming of age. However, they are computer intensive and their sequential execution can became an unfeasible task. This study presents a middleware, called Flexible Receptor Middleware (FReMI), to assist in faster docking simulations of flexible receptors. FReMI handles intensive tasks and data of totally fully-flexible receptor models in virtual screening and, provides the interoperability between a Web Fully-flexible Docking Workflow (W-FReDoW) and two different High Performance Computing (HPC) environments. FReMI uses internet protocols to communicate with W-FReDoW which helps to reduce the FFR model dimension with a data pattern. Also it sends tasks of docking simulations to execute in a HPC of dedicated cluster and; an alternative model of virtual cluster built on Amazon's Elastic Compute Cloud (EC2). The results are the FReMI conceptual architecture and two sets of experiments from execution of the FReMI. The first set reports the experiments performed with FReMI using a sample of snapshots from a FFR model on both HPC environments. The second one describes the experiments, on the complete data set, performed with FReMI and W-FReDoW shared execution in a MPI cluster environment on Amazon EC2 instances only. The last set of experiments results shows a reduction of the FFR model dimensionality, transforming it into a Reduced Fully-Flexible Receptor (RFFR) model, by discarding the non-promising conformations generated by W-FReDoW. It also reduces the total execution time to between 10-30% of that of FReMI's only execution, which, in turn, decreased near 94% with respect to the serial execution.



**Keywords: Middleware, Cluster, Amazon EC2, Molecular Docking Simulations.**

# FReMI – UM MIDDLEWARE PARA EXECUTAR SIMULAÇÕES DE DOCAGEM MOLECULAR DE MODELOS DE RECEPTORES TOTALMENTE FLEXÍVEIS EM AMBIENTES DE ALTO DESEMPENHO

## RESUMO

Simulações de docagem molecular de modelos de receptores totalmente flexíveis (*Fully-Flexible Receptor* - FFR) estão se tornando cada vez mais frequentes. Entretanto, tais simulações exigem alto nível de processamento e sua execução sequencial pode se tornar uma tarefa impraticável. Este trabalho apresenta um *middleware*, chamado Middleware de Receptores Flexível (*Flexible Receptor Middleware* – FReMI), que auxilia a reduzir o tempo total de execução nas simulações de docagem molecular de receptores totalmente flexíveis. FReMI manipula uma quantidade intensiva de dados e tarefas para executar a triagem virtual de modelos de receptores totalmente flexíveis, e provê interoperabilidade entre o *web workflow* de docagem de receptores flexíveis (*Web Fully-flexible Docking Workflow* - W-FReDoW) e dois diferentes ambientes de alto desempenho (*High Performance Computing* – HPC). FReMI utiliza protocolos de internet para comunicar com o W-FReDoW , o qual auxilia na redução da dimensão do modelo FFR por meio de um padrão de dados. Além disso, FReMI envia tarefas de simulações de docagem para serem executadas em um cluster dedicado e também em um alternativo modelo de cluster virtual construído por meio de nuvens de computadores elásticos da Amazon (*Amazon's Elastic Compute Cloud* – EC2). Os resultados apresentam uma arquitetura conceitual do FReMI e dois conjuntos de experimentos a partir da execução do FReMI. O primeiro conjunto relatou os experimentos realizados com FReMI, usando uma amostra de *snapshots* a partir de um modelo FFR e os dois ambientes HPC. O segundo conjunto descreveu os experimentos, com um conjunto de dados completo, executando FReMI e W-FReDoW apenas em um ambiente de cluster MPI construído com as instâncias da Amazon EC2. Os resultados do último conjunto de experimentos apresentaram uma redução na dimensionalidade do modelo FFR, transformando ele um modelo de receptor flexível totalmente reduzido (*Reduced Fully-Flexible Receptor Model* – RFFR), por meio do descarte de conformações não promissoras identificadas pelo W-FReDoW. Além disso, a redução do tempo total de execução do FReMI com o W-FReDoW foi entre 10 a 30% a partir da execução separada do FReMI, e de aproximadamente 94% do FReMI a partir da sua respectiva execução sequencial.

**Palavras-Chave: Middleware, Cluster, Amazon EC2, Simulações de docagem molecular.**

# LIST OF FIGURES

# LIST OF TABLES

# ABREVIATIONS

| | |
|---|---|
| 3-D | Three-Dimensional |
| AMI | Amazon Machine Image |
| API | Application Programming Interface |
| AWS | Amazon Web Services |
| CPU | Central Processing Unit |
| EBS | Elastic Block Store |
| EC2 | Amazon Elastic Compute Cloud |
| ETH | Ethionamide |
| FEB | Free Energy of Binding |
| FReDD | Flexible Receptor Docking Database |
| FReMI | Flexible Receptor Middleware |
| FFR | Fully-Flexible Receptor |
| GB | Gigabyte |
| GUI | Graphical User Interface |
| HPC | High Performance Computing |
| HTTP | Hypertext Transfer Protocol |
| IaaS | Infrastructure as a Service |
| InhA | Enoyl Reductase from *Mycobacterium tuberculosis* |
| LGA | Lamarckian Genetic Algorithm |
| KB | Kilobytes |
| MD | Molecular Dynamics |
| MPI | Message Passing Interface |
| MTC | Many-Task Computing |
| NADH | Nicotinamide Adenine Dinucleotide – Reduced Form |
| NFS | Network File System |
| PDB | Protein Data Bank |
| PIF | Pentacyano(isoniazid)ferrate(II) |
| P-SaMI | Self-adaptive Multiple Instances Pattern |
| REST | Representacional State Transfer |
| RDD | Rational Drug Design |
| RFFR | Reduced Fully-Flexible Receptor |
| RMSD | Root Mean Squared Deviation |
| S3 | Amazon Simple Storage Service |
| SMP | Symmetric Multiprocessing |

| | |
|---|---|
| SSH | Secure Shell |
| SWfWS | Scientific Workflow Management Systems |
| TCL | Triclosan |
| TCP | Transmission Control Protocol |
| URI | Universal Resource Identifier |
| W-FReDoW | Web-Flexible Receptor Docking Workflow |
| XML | Extensible Markup Language |

# SUMARY

# 1. INTRODUCTION

Large scale scientific experiments have had an ever increasing demand for high performance distributed computing. This typical scenario is found in bioinformatics, which needs to perform computer modelling and simulation on data varying from DNA sequence to protein structure to protein-ligand interactions. It produces sets of data flow that are processed by an iterative sequence of tasks, software or services [COU10].

Rational Drug Design (RDD) constitutes one of the earliest medical applications of bioinformatics [LUS01]. RDD aims to transform biologically active compounds into suitable drugs [KAP08]. *In silico* molecular docking simulation is one the main steps of RDD. It is used to identify and optimize drug candidates by computationally examining and modelling molecular interactions between ligands or small molecules and a target protein or receptor [KAP08]. The best ligand orientation and conformation inside the binding pocket is computed in terms of an estimated Free Energy of Bind (FEB) by a software, for instance, AutoDock4.2 [MOR09]. In order to mimic the natural, *in vitro* and *in vivo*, behaviour of ligands and receptors, their plasticity or flexibility should be treated in an explicit manner [MAC10b].

Generally, molecular docking algorithms consider receptors as rigid bodies; however, receptors are inherently flexible in the cellular environment [MAC10b]. A major approach to incorporate the explicit flexibility of receptors in molecular docking simulations is by means of snapshots derived from a molecular dynamics simulation [6] trajectory of the receptor (reviewed by [ALO06]). The resulting receptor model is called a Fully-Flexible Receptor (FFR) model. Organizing and handling the execution and analysis of molecular docking simulations of FFR models and flexible ligands are not trivial tasks. The dimension of the FFR model can become a limiting step because, instead of performing docking simulations in a single, rigid receptor conformation, we must do it for all ***n*** conformations that make up the FFR model [MAC10b]. ***n*** can vary from hundreds to thousands to millions of conformations. Therefore, the high computing cost involved in using FFR models to perform practical virtual screening of thousands or millions of ligands may turn it unfeasible. For this reason, we have been developing methods [MAC10b] to simplify or reduce the FFR model dimensionality. We dubbed this simpler representation of a FFR model a Reduced Fully-Flexible Receptor (RFFR) model. A RFFR model is achieved by eliminating redundancy in the FFR model through clustering its set of conformations, thus generating subsets which should contain the most promising conformations [MAC10b].

At present, to develop a RFFR, we still need to perform thousands, and in the near future, this number should grow to hundreds of thousands of molecular docking simulations of a particular target receptor modelled as a FFR model. However, the sequential execution of docking simulations of FFR models by software, such as AutoDock4.2 [MOR09], is computationally very expensive [MAC10b], hence demanding days, weeks, or even months of CPU time. As a result, to make use of parallel

processing is paramount to enhance the performance of the high-throughput molecular docking simulations of FFR models and molecule database, minimizing docking CPU time without lowering the quality of the RFFR models produced.

The present study aims to contribute to the reduction of the overall execution time of molecular docking simulations of FFR models using parallel programming algorithms to perform such experiments in HPC environments. Additionally, a careful selection of a set of conformations [HUB10] has been used to eliminate non-promising conformations which, in turn, allows a simplification of the FFR model dimensionality, generating an RFFR model, and permits docking simulations of flexible receptors even faster. To this end, the middleware called FReMI (Flexible Receptor Middleware) was built to provide communication between the Web Fully-flexible Docking Workflow (W-FReDoW) and two different HPC environments; a local cluster infrastructure and a virtual cluster on cloud computing.

Cloud computing is a new and promising trend for delivering information technology services as computing utilities [BUY09]. It offers software as a service by means of companies such as Amazon Web Services (AWS) [AWS12]. These commercial tool packages, which assure high scalability and support ubiquitous access [BUY09] by service companies, have increased the interest of the scientific community. Web services companies provide customers with storage and CPU power on an on-demand basis, and allows researchers to dynamically build their own environments and access them from anywhere in the world at any time.

This dissertation presents two results: The FReMI conceptual architecture and the experimental results. The FReMI conceptual architecture was created to show the functions and data used to handle the many tasks within middleware, and its interoperability features with W-FReDoW and HPC environments. The experimental results, which are obtained from FReMI execution, constitutes of two set of experiments. The first set reports the experiments performed with FReMI using a sample of snapshots from a FFR model on two HPC environments; the Atlântica cluster and the virtual MPI cluster on Amazon EC2. The second one described the experiments, on the complete data set, performed with FReMI and W-FReDoW shared execution in a MPI cluster environment on Amazon EC2 instances only. The experimental results showed that W-FReDoW reduced the total execution time to between 10-30% of that of FReMI´s only execution, which, in turn, decreased near 94 % with respect to the serial execution time.

### 1.1 Motivation

Demand for the pharmaceutical industry in marketing drugs with minimum toxic side effects is growing potential as new diseases are surfacing. According to PricewaterhouseCoopers, due to the growth and aging of population, the global market for medicines is growing. Nevertheless, the production of new drugs still is a complex and challenging process, since in addition to spending a

long time, requires large investments in technology resources. Currently, new computational tools and methodologies are being developed to improve the RDD process at different stages. One these enhancements include the incorporation of protein flexibility in the molecular docking process [ALO06]. The search for methods that reduce the computational time involved in the molecular docking process, and to investigate accurately chemical and biological information about ligands and receptors is extremely important to identify and advance the RDD process [KAP08].

Thus, the major motivation of this study is to increase the computing speed and efficiency of the large scale molecular docking experiments, reducing the number of conformations of the FFR model and increasing the number of simulations performed simultaneously by multi-core and/or multi-processors. Hence, a heuristic function is used to distribute tasks which are executed in parallel by HPC environments in order to execute molecular docking simulations only for conformations that present to be most promising gradually during the interactions of the ligand-receptor complex.

For the above reasons, the contribution of this study is to connect software components which allow a set of services runs multiple processes on one or more machines providing the interoperability between them through FReMI middleware. As a consequence, reduce excessive runtime during large-scale virtual screening jobs as well as streamline the RDD process.

## 1.2 Objectives

### 1.2.1 Main Objective

The main objective of this study is to develop a middleware to handle many tasks and provide the interoperability between web servers and high performance environments for parallelizing massively molecular docking simulations of subgroups of conformations from FFR models. Consequently, future molecular docking experiments, with different ligands, will not use all the conformations that make up a FFR model, but instead, only those which are significantly more promising. Thus, the total time spent in the molecular docking experiments for each FFR model can be considerably reduced, and new ever great virtual libraries can be exploited more effectively.

### 1.2.2 Specific Objectives

The specific objectives of this dissertation are:

- To develop a heuristic function to create balanced tasks queues according to the priorities of different subsets of conformations of FFR models. This function sorts the placing of a fraction proportional of snapshots in the task queue in order of its importance belonging to each group of snapshots active.

– To define a parallel programming model that is able to distribute of a huge amount of tasks along the multi-processing nodes belonging to the HPC environments.

– To apply communication protocol and files pattern to achieve the acknowledgment of data sending and receiving between FReMI middleware and W-FReDoW web environment.

– To execute molecular docking simulations of a snapshots sample from FFR model on FReMI middleware. For executing this experiment the development is separated into two parts. First, perform the snapshots sample in FReMI using a dedicated cluster and a virtual cluster on Amazon EC2, then compare the performance between them. Second, perform the W-FReDoW and FReMI shared execution using a complete snapshots data set from a FFR model.

## 1.3 Research Methodology

The following are activities performed during the development of this dissertation.

- A study on molecular docking simulations of FFR models to undertake information about the problem in study and the works which have been done by group research of the LABIO-PUCRS (Laboratório de Bioinformática, Modelagem e Simulação de Biossistemas da Pontifícia Universidade Católica do Rio Grande do Sul). Also, the review of the literature to identify relevant aspects about parallel execution and middleware.
- Design of the FReMI conceptual architecture to model the data and control streams inside and outside of the middleware proposed.
- Development of FReMI with its procedures and the heuristic functions used to handle the snapshots will be process by HPC environments.
- Execution of a sample of snapshots using only FReMI in a dedicated cluster, and afterward in a virtual cluster on Amazon EC2 to compare the performance.
- Execution of a complete FFR model, with 3,100 snapshots, using FReMI and also W-FReDoW on Amazon EC2. Firstly, FReMI executes all the snapshots without W-FReDoW to discover the spent time to execute all the conformations in several processes. Subsequently FReMI and W-FReDoW was executed together with the purpose of reduce the execution time result of the first simulation.

## 1.4 Dissertation Overview

This dissertation is organized in eight sections as follows:

- Chapter 2 introduces the major bioinformatics and computing concepts necessary for a better comprehension of this dissertation. It starts with an overview of the RDD process and

molecular docking simulations, and close with explanation of the approach used to consider the explicit flexibility of receptors based on the MD simulations

- Chapter 3 describes the basic aspects of parallel computers used to execute the molecular docking simulations in this study. It starts with a summary of computing hardware based on multi-core and multi-processor machines. After that, an overview about on Cloud Computing and Amazon Web Services, focussing on those services relevant to FReMI execution.
- In Chapter 4 is described the materials and methods used during the development of this dissertation in order to build and execute FReMI. It is divided into three parts. Firstly, the main tools employed by this study: AutoDock4.2, Atlântica local cluster, Amazon EC2 cluster. After, the the data mining techniques [MAC10a] [MAC11a] [MAC11c] to cluster the snapshots with similarities features from a FFR model; the P-SaMI data pattern [HUB10] to to achieve the RFFR model by means of selecting of promising snapshots; and W-FReDoW to execute P-SaMI and prepare the AutoDock4.2 input files. Finally, the MPI [MPI09] and OpenMP [OMP11] libraries used on C programming language to make the hybrid parallel programming model as well as the XML and HTTP POST internet communication protocols used to link W-FReDoW and FReMI execution.
- Chapter 5 presents the first result of this dissertation, i.e., the FReMI conceptual architecture. It includes the representation of this architecture and the detailed specification of every function with the data and control flows. Also, it reports the W-FReDoW functions that are able to provide communication with FReMI.
- Chapter 6 presents the second set of results of this dissertation. Two sets of experiments are performed using the FReMI conceptual architecture defined in Chapter 5. The first set reports the experiments with FReMI using a sample of snapshots data set from a FFR model on two HPC environments. The second one describes the experiments, on the complete data set, performed with FReMI and W-FReDoW shared execution in a virtual cluster on Amazon EC2 only.
- Chapter 7 reports related works. These works are associated with the current approaches used to accelerate the AutoDock4.2 execution.
- Chapter 8 summarizes the conclusions of the study which have led to this dissertation. Additionally, it draws the main contributions and gives some directions for future work.

# 2. BACKGROUND

This chapter introduces the major bioinformatics and computing concepts necessary for a better comprehension of this dissertation. It starts with an overview of the RDD process and molecular docking simulations, and close with explanation of the approach used to consider the explicit flexibility of receptors based on the MD simulations.

## 2.1 Rational Drug Design (RDD) and Molecular Docking

According to Stoddard et al. [STO93] RDD refers to the systematic exploration of the three-dimensional (3-D) structure of a receptor in order to find potential ligands that might bind to the target with high affinity and specificity. This process involves a set of four steps which are described by Kuntz [KUN92] and outlined below.

1. The first step consists in finding the macromolecule of pharmacological importance or target receptor (protein, DNA, RNA or others) [MAN09]. Experimental 3-D structures of proteins are found, for instance, in structural databases such as the Protein Data Bank (PDB) [BER00]. Computational analysis of these target receptors may reveal possible binding sites.
2. Based on the possible binding sites found in the first step, a set of ligands is selected that can fit into this binding region or pocket in the receptor. Usually, the ligands are found in databases of compounds like ZINC [IRW05]. The different conformations and orientations that each ligand can fit into a receptor binding pocket are simulated in this step by docking software.
3. The ligands that bind successfully to the receptor binding pocket and which are able to inhibit or enhance its activity, depending on the objective of the project of drug design, are bought or synthesized, and then experimentally tested.
4. Based on the experimental results, the manufacture of a new drug is conducted or the RDD process returns to step 1.

One of the most important steps of the RRD process is the *in silico* molecular docking simulation in step 2. Molecular docking simulations can be assessed *in vitro.* However, assaying a target receptor against a database, like ZINC [IRW05] which holds more than 23 millions of compounds, does not constitute a rational approach [ALO06]. Hence docking simulations are used for lead compound discovery, typically by computationally screening a large database of organic molecules for putative ligands that fit into a binding site [WEI04].

Molecular docking simulations sample hundreds of thousands of orientations and conformations of a ligand inside the protein binding site and evaluate the free energy of binding (FEB), and rank the orientations/conformations according to their scores [HUA07]. The majority of

molecular docking methods treat the ligands as flexible, but the receptors are treated as rigid molecules [MAC11c].

For computing the quality of the fit between receptors and ligands, the docking algorithms rank the best docking results in terms of the estimated Free Energy of Binding (FEB) and the Root Mean Squared Deviation (RMSD). Thus, the more effective ligand-receptor association is evidenced by docking algorithms when the FEB (in kcal/mol) is more negative and the RMSD is close to zero Å (in the cases where the final docked position is known). Over 60 types of software currently investigate different methods to find the best fit between a receptor and possible ligands [PLE11]. Some examples are AutoDock [MOR09], DOCK [LAN09], GOLD [JON97], and FlexE [CLA01]. Figure 2.1 illustrates a molecular docking process.

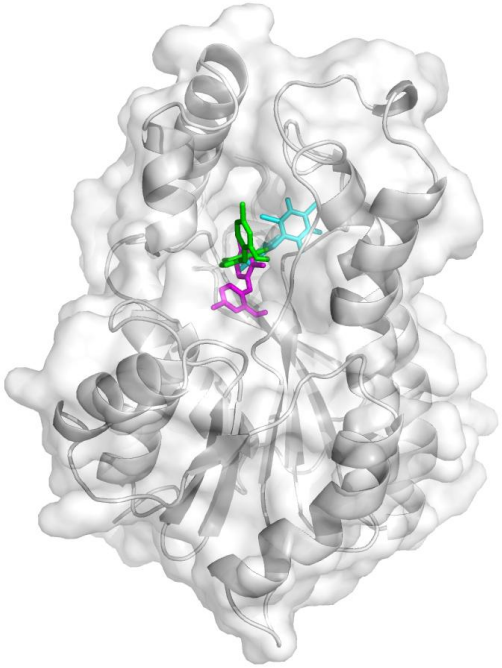

Figure 2.1: 3-D Representation of the molecular docking process. The protein receptor, with secondary structures represented by ribbons and a transparent molecular surface, is coloured grey. The TCL ligand is in sticks. The initial position of the ligand is in cyan and two different positions along a docking simulation are shown in green and magenta. The protein receptor is the enzyme Enoyl-Reductase or InhA (PDB ID: 1P45) from *Mycobacterium tuberculosis* [KUO03].

However, proteins are inherently flexible systems and this flexibility is frequently essential to determine their functions [COZ08]. According to Plewczynski et al. [PLE11] only 60% of the best docking algorithms correctly predict the pose of ligands when a single, rigid receptor conformation is considered. Therefore, realistic docking simulations need to take into account the molecular flexibility, for both receptor and ligand, since in many cases the key-lock model does not work well and the induced-fit model is more appropriate [WON08].

Nonetheless, to consider the plasticity or flexibility of a receptor and ligand in docking simulations is still challenging. Therefore, new computational approaches are being developed to simulate flexible receptors [MAC10b] [MAC11c].

### 2.1.1 Approach to Consider the Explicit Flexibility of Receptors

Several approaches have been used to consider the explicit flexibility of receptors (reviewed by Alonso et al. [ALO06], Cozzini et al. [COZ08], Teodoro et al. [TEO03] and Totrov et al. [TOT08]). The employment of many receptor structures [COZ08] is becoming common place in the simulation of the natural, *in vivo* and *in vitro*, behaviour of flexible receptors. In this approach an ensemble of receptor conformations or snapshots derived from a MD simulation trajectory (reviewed by [ALO06]) are used to incorporate the explicit flexibility of receptors in the molecular docking simulations.

According to Alonso et al. [ALO06] MD simulations are extremely important to understand the dynamic behaviour of proteins at different timescales, from fast internal motions to slow conformational changes or even protein folding processes. The result of a MD simulation is a series of instant conformations of the protein receptor along the simulation time scale. These conformations are also often called snapshots. Figure 2.2 illustrates the flexibility of a receptor derived from a MD simulations trajectory.

Figure 2.2: Flexibility of the InhA enzyme from *Mycobacterium tuberculosis* [PDB ID: 1P45]. Superposition of different InhA conformations, represented as ribbons, along a MD simulation. The initial conformation of the simulation is the experimental crystal structure [PDB ID: 1P45] and is coloured in black. Two other conformations or snapshots were taken from the MD simulation at 1,000 ps (grey) and 3,000 ps (light grey). The dashed rectangle highlights the most flexible regions of this receptor.

The execution of MD simulations is computationally expensive. However, because of the high level of accuracy in the modelling process [TEO03], it is the best method for identifying from crystal structures of proteins alternative binding forms otherwise not apparent from the rigid picture [ALO06]. Furthermore, Cozzini et al. [COZ08] states that among all the available approaches to treat the explicit flexibility of a receptor, the MD technique is the most affordable and accessible method to produce many protein conformations at reasonable cost.

This study models the explicit flexibility of a receptor by using a set of conformations derived from its MD simulation. Such a receptor has been named a Fully-Flexible Receptor (FFR) model [MAC10b] [MAC11c]. For each conformation in the FFR model, a docking simulation is executed and analysed [MAC10b].

The dimensionality of a FFR model is determined by the length of the simulation and how often snapshots are saved during the simulation. New techniques have been developed by Machado et al. [MAC10b] and Hübler [HUB10] to simplify or reduce the size of the FFR model. This new representation of the receptor is called the Reduced Fully-Flexible Receptor (RFFR) model.

# 3. PARALLEL COMPUTERS

This chapter describes basic concepts of parallel computers used to execute the molecular docking simulations in this study. It starts with a summary of computing hardware based on multi-core and multi-processor machines. After that, an overview about cluster environment as a popular architecture formed by commodity computers will be given. Finally, a brief description about the new trend in virtual HPC cluster using Amazon's Elastic Computing Cloud (EC2) will be presented.

## 3.1 Architecture and Taxonomy of Parallel Computers

Parallel computing has become widely used to execute simultaneously instructions that require significant resources. Parallel computational structures must provide hardware and software support to use multiple processors and work on the same task with different data. Initially it is important to know how the processors and memory are organized, how machines are interconnected and then determinate which distributed system should be used to manage these hardware resources through software concepts [TAN10]. Such resources include sharing of CPUs, memories, peripheral devices, networks and data.

There are several different classifications of parallel programming models. The most common representation is based on a number of data and instruction streams or operations [FLY72]. Still according to Flynn [FLY72] there are four different categories to classify the computer architecture and their taxonomy.

1. SISD (Single Instruction Single Data). This model defines the traditional von Neumann computer where a processor executes only a data stream.
2. SIMD (Single Instruction Multiple Data). In this representation each processor executes the same program.
3. MISD (Multiple Instruction Single Data). This model defines systems where multiple programs operate on the same data.
4. MIMD (Multiple Instruction Multiple Data). This model executes multiple programs on multiple data. This classification involves parallel computer architectures which typically are known as clusters.

MIMD identify the supercomputers which combine high processing capacity with intensive calculation tasks. It is classified in two groups: loosely-coupled and tightly-coupled machines [TAN02]. The loosely-coupled machines are heterogeneous multicomputer known as computational grid [BUY00] [FOR06]. The tightly-coupled machines are homogeneous multicomputer known as cluster which is the machines used to perform the molecular docking simulations using the middleware proposed in this work.

In the early years, MIMD supercomputers executed only one process per computer. Nowadays, in the Symmetric Multiprocessing (SMP) machines, each computer contains multiple processors or multi-cores. SMP machines treat the cores as separate processors improving significantly the performance of massive computational workload [TAN02].

In this study, the used HPC environments are clusters that consist of two or more SMP machines or nodes linked by an interconnect network. The major infrastructure and operations of this kind of machines will be discussed in the next section.

### 3.2 HPC Cluster Environments

The first cluster computing model was developed in the 1960s by IBM as an alternative for connecting large mainframes [BUY99]. Only in the 1980s new cluster trends started to emerge as conventional parallel and distributed platforms. In the beginning these machines were called high-performance microprocessors, followed by high-speed networks, most recently supercomputers and at present they are dubbed High Performance Computers (HPC) [BUY99][BUY00]. Currently, clusters have been widely used for research and development of science, engineering, bioinformatics, commerce and industry applications that demand high performance computing. To meet the requirements of these areas a greater number of supercomputers have been emerged with power and advance architectures, such as Fujitsu at RIKEN Advanced Institute for Computational Science at Japan, Jaguar at the Ridge National Laboratory and Pleiades at the NASA/Ames Research Center. These computers are listed among the top 10 fastest supercomputers in the world (TOP 500 supercomputer sites [TOP11]).

The high availability of commodity high-performance microprocessors and high-speed networks, combined with scalability to perform the shared functions as a single system, are making clusters an attractive platform for parallel processing leading to low-cost commodity supercomputing [BUY99]. Clusters can aggregate processing capacity based on a huge number of computers called workstations, nodes or hosts. For example, Figure 3.1 shows a typical architecture of a cluster with four workstations and an interconnection network to support communication between them. Each node provides an assisted system to manage the operations and communications between the workstations used. Such systems provide services as core/thread coordination, inter-process communication, and device handling. According to Buyya [BUY99] the main features of a cluster operation system are:

- Manageability to administrate the local and remote resources.
- Stability to support failures with system recovery.
- Performance to guarantee efficiency for all types of operations.
- Extensibility to provide integration of cluster-specific extensions.
- Scalability to scale the resources without impact on performance.

- Support between user and system administrator.
- Heterogeneity over multiple architectures that have heterogeneous hardware components.

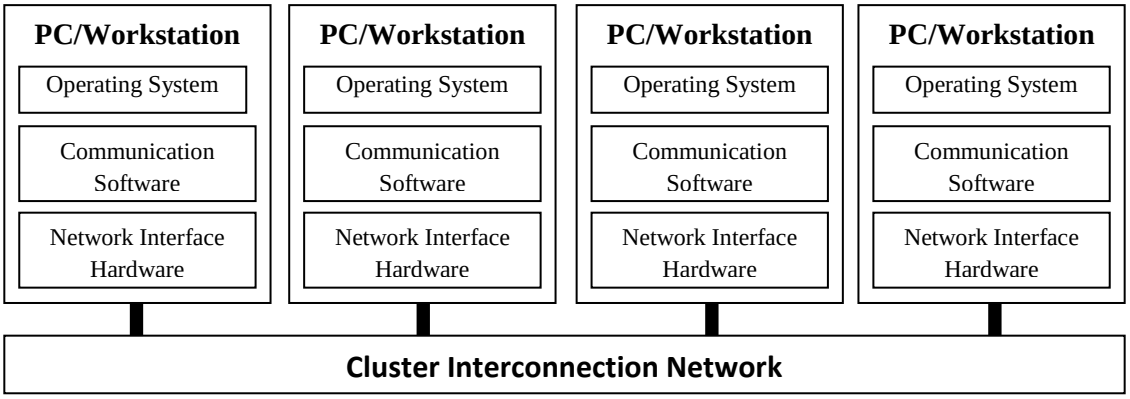


Figure 3.1: Cluster architecture. Cluster Interconnection Network links the software and hardware communication among the workstations (Adapted from [BUY99]).

The main advantages of using clusters are their low cost and high performance. This naturally means that their benefits and capacity are greater if compared to the isolated components because each node has inferior configuration; otherwise, the union of these nodes allow a significant increase in performance. Furthermore, clusters include nodes with low-latency interconnections aiming at gains in performance. Other advantage refers to high accessibility; it means that when a machine of the pool fails tasks can be reallocated to other nodes without losing performance. Thus, the loss of a machine does not affect the cluster operation.

There are several metrics to measure the cluster performance during an execution. The metrics take into account factors such as hardware availability and software to support operational system, file system, communication protocol and network interface. In summary, there are two key features used to measure the capabilities and potential performance of clusters: network and application performance. Analysis of network is based on bandwidth[1] and latency[2] in the interconnection. Analysis of an application relies on the speedup and efficiency. The former is an estimate of how faster is the parallel version of the application compared to its sequential version. The latter identifies if the nodes were well-used during execution of an application.

In order to achieve a desired level of performance with the metrics mentioned above, parallel software must follow some parallel programming models to avoid overhead and idle processors, in other words, carry on synchronisation among processors towards improving the computational load balance and reducing the communication time [TRE00]. Hence, parallel programming allows dividing a computational workload into several separate processes which are concurrently executed by different processors to solve a common task. This approach requires setting the granularity and

[1] Bandwidth: is the amount of data that can be passed along a network in a given period of time.

[2] Latency: time to prepare and transmit data from a source node to a destination node.

communication of the HPC environment [TRE00]. Granularity is the relative size of the units of computation that execute in parallel (coarseness or fineness of task division). Communication is the way that separate units of computation exchange data and synchronise their activity.

The cluster approach is used in two types of HPC environments employed to execute the molecular docking simulations in this study. The first one is a cluster of commodity computers connected by a fast network. The second one is a virtual cluster built on Amazon EC2. The specifications and configurations of each HPC environment will be provided in Chapter 4, Sections 4.1.2 and 4.1.3.

However, because the Amazon EC2 entails a new concept of HPC environment on cloud computing, the next section gives a description of its main features.

### 3.3 Amazon's Elastic Computing Cloud – EC2

Before starting to explain the AWS, the commercial product used in this study to execute the molecular docking simulations on a Cloud, it is very important to present some overall concepts to understanding how cloud computing works.

Cloud computing are been defined in several manners by different authors. However, each definition is modelled according to particular aspects present in the hardware and software services which are largely provided over the Internet by cloud computing applications.

Foster et al. [FOS05] claim that cloud computing is a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the internet. Buyya et al. [BUY09] point out that clouds are designed to provide services to external users; providers need to be compensated for sharing their resources and capabilities. Armbrust et al. [ARM10] states, in terms of trade, that cloud computing is a long-held dream of computing as a utility, because it has the potential to transform a large part of the IT industry, making software even more attractive as a service and shaping the way IT hardware is designed and purchased.

All the definitions converge to the same goal which is to provide computational resources when and where you need them, offering accessibility to use multiples clouds, scalability to launch large number of instances by virtual machine (VM) support, flash memory and schedule VMs [ARM10]. Users or brokers acting on their behalf submit service requests from suitable businesses and negotiate with them to achieve ideal service contracts [BUY09]. Amazon, Google, Salesfores, IBM, Microsoft and Sun Microsystems are examples of these business companies which offer data center for hosting cloud computing applications. These industries describe their products as an infrastructure and platform services. According to Armbrust et al. [ARM10], software as a service is used to identify the application delivered over the internet. Thus, each cloud platform provides its technologies and

infrastructure for supplying software as a service for consumers and enterprises to access on demand regardless of time and location [BUY09].

Amazon Web Services is a suite of web services made available by Amazon that allow developers to access and build on the company's technology platform [MUR08]. It shares the work burden between multiple components as a service pool in the World Wide Web by means of Infrastructure as a Service (IaaS). IaaS is the term typified by the AWS cloud to identify the on-demand access to its compute infrastructure [MUR08]. Application administrators request as many servers as necessary to meet the scalability needs of their application.

AWS is a public cloud which offers set of web services to anyone on the internet by means of pay-as-you-go tax [MUR08]. The most important services it provides are [AWS12]:

- Amazon Simple Storage Service (S3). This service provides to the users the ability to store large amounts of data reliably and with high availability. Also it allows building, maintaining and backing-up the storage system.
- Amazon Elastic Compute Cloud (EC2). This service is being extensively used to run Virtual Machines (VM) multiple on demands [MUR08]. It provide as many computers as you need to process your data in a large number of physical computers.
- Amazon Simple Queue Service (SQS). This service provides a reliable, scalable queuing service between EC2 instances.
- Amazon Simple DB. This service provides many of facilities of relational databases, and provides a web services interface to the system.

AWS provides several services which are useful for scientific applications; however in this study only Amazon's EC2 and S3 services have been extensively used and explored. For this reason, the features of both services will be presented in detail. Further information about the other services can be found in [MUR08] and Amazon Web Services web site [AWS12].

Amazon's Simple Storage Service (S3) is a data model which allows users to store unlimited amounts of data by means of two kinds of storage resources: objects and buckets. More precisely, data and metadata are stored in objects while buckets are containers that can hold an infinite number of objects [MUR08]. Furthermore, with the purpose of manipulating the storage resources S3 uses protocols such as SOAP[3], REST [FIE00] and Bit Torrent P2P [POU05] which permit the users to read and write data on Amazon S3. It also supplies access control mechanisms to keep the information private or public. The public information gives accessibility to anyone on the internet via normal web browsers by standard Universal Resource Identifiers (URIs). For example, the alternative domain name used to download the objects is http://s3.amazonaws.com/bucket-name/object-name

[3] http://www.w3.org/TR/soap.

Although the online storage service is priced according to the geographical location, S3 account holders are billed monthly for their usage of service considering three key aspects. First, the volume of data transferred to or from S3, second the storage space consumed and finally, the number of Application Programming Interface (API) request operations that have been performed on each account. According to Amazon Web Services [AWS12] the storage price on Amazon S3 infrastructure ranges between $0.12 and $0.15 per GB per month whereas the data transfer is $ 0.12 per month. However, to calculate the price for data storage in S3 it is necessary to know the location and also the amount of data used during the executions due to various policies to determine the charges.

Amazon's Elastic Compute Cloud (EC2) provides a virtual computing environment based on demand to run applications [BUY09] [MUR08]. The virtual computing environment holds one or more virtual machines which allow installing and configuring the pool of servers for handling computing tasks as a root user on Linux machines. Each virtual machine can be launched from prepared servers created by third parties or set up an EC2 server to work as the user's want, i.e. the users install their own software and configure their own environment. Thus, the user can start as many virtual servers as necessary to perform a task, increase or decrease the number of servers as demand rises and falls, and stop them all when the tasks is finished [MUR08]. Furthermore, the user pay only for the computing service you use.

The EC2 service comprises three key components [MUR08]:

- Instances. The instances are the virtual machine which run in the EC2 environment and perform computing tasks that would typically be done by physical servers.
- Environment. The instances run in the EC2 environment, which provides contextual data, configurable access control, and other information that the instances need to do their work.
- Amazon Machine Images (AMIs). The AMI is used to launch a machine image as the boot disk for the instances. They are files that capture a full snapshot of an EC2 instance at a point in time, containing its applications, libraries, and even its data associated configuration settings or select from a library of globally available AMIs [BUY09].

Each EC2 virtual machine instance launched is based on Xen virtualisation technology [BAR03]. This engine allows one physical computer to be shared by several virtual computers each of which hosts different operating systems. Hence, the EC2 Compute Unit is used to provide a baseline guide to the computing capacity expected from an EC2 instance. As a reference point, an instance is a rating of 1 EC2 compute unit to provide the same CPU capacity as a physical machine with a 1.0 to 1.2 GHz 2007 Opteron or 2007 Xeon processor [MUR08]. Thus, the use of VMs gives rise to further challenges such as the intelligent allocation of physical resources for managing competing resource demands of the users [BUY09].

Amazon standardises the services available into five different types of instances with different levels of performance and resourcing [MUR08]. The configuration and tier of each instance are showed in Table 3.1. Where are illustrated the variations of virtual cores, the amount of RAM, whether it is a 32-bit or 64-bit architecture, how much storage is available, and the prices, which are charges on an hourly basis. The prices are based on Amazon Web Services web site [AWS12] to the region used to execute the simulations in this study, i.e., US East Region.

Table 3.1: Specification of five instances types on Amazon EC2 [AWS12].

| Type | #cores | RAM | Bits | Storage (hard disk) | Price (per hour) |
|---|---|---|---|---|---|
| m1.small | 1 | 1.7 GB | 32 | 160 GB | $0.085 |
| m1.large | 2 | 7.5 GB | 64 | 850 GB | $0.34 |
| m1.xlarge | 4 | 15 GB | 64 | 1690 GB | $0.68 |
| c1.medium | 2 | 1.7 GB | 32 | 350 GB | $0.17 |
| c1.xlarge | 8 | 7 GB | 64 | 1690 GB | $0.68 |

The Amazon EC2 instance type is chosen when an AMI is launched. The AMI captures the root file system of an instance in a series of files [MUR08]. It means that when an instance is launched it starts the boot from the software, the configuration settings and the data that were stored in the AMI. Each AMI is stored on Amazon's S3 service and needs to be registered with EC2 to give an AMI ID that is used to start a server instance at any time. Basically an AMI is a snapshot of the instance at the point of creation. Thus, each time a new instance of the created AMI is started it will have the same state from which it was created [MUR08]. Furthermore, after the AMI is launched it gives a DNS address, which can be accessed by the users using SSH command to manage the virtual server, i.e. running the applications, install the needed software, configuring the environment or even creating a new AMI.

Amazon EC2 also contains a secondary storage volume which is associated in its instances. This storage space exists only while the instance is active and is called Elastic Block Store (EBS). Amazon EBS volumes are off-instance storage which persists independently from the life of an instance; likewise, when an instance is terminated the EBS volume terminates too [HAZ08] [AWS12]. An EBS is provided as a block device which the user may format with an appropriate file system. For example, in this study, all the input and output files resulting from the FReMI execution are stored in a shared file directory in a block device of the EBS data volume inside the EC2 instance.

According to Amazon Web Services web site [AWS12] the costs to use the EBS volume storage is charged by the amount of allocation until it is released. Thus, a rate of $ 0.10 per allocated GB per month and $ 0.10 per 1 million I/O requests is charged by each EBS volume active in the instance.

The key objective to make use of cloud computing in the FReMI execution is to compare the performance, the usability and the costs take in a local cluster infrastructure and a virtual cluster environment on Amazon EC2. Thus, from a comparative study of this metrics verifies the benefits

undertaken in each cluster environment through execution in parallel of molecular docking simulations of conformations subsets of the FFR model on FReMI.

# 4. MATERIALS AND METHODS

This chapter presents all the materials and methods used to develop this study. It is divided into three sections. The first section describes the software used to execute the molecular docking simulations and the HPC environments used in the parallel execution of FReMI's tasks. The following section presents an overview of the approach used to generate the different groups of snapshots which has given rise to all the data used in this dissertation and the data pattern applied within W-FReDoW to reduce the dimension of the FFR model. The last section explains the implementation of FReMI. It includes the parallel programming paradigm to execute simultaneously the molecular docking simulations on clusters and the communication protocols used to give communication between FReMI and W-FReDoW.

## 4.1 Hardware and Software

The AutoDock4.2 software used to perform molecular docking simulations of the FFR model and the MPI high performance environments used to execute the parallel docking jobs are presented in this section.

### 4.1.1 AutoDock4.2

AutoDock, a non-commercial open-source software, has been widely used to perform virtual screening of a huge database of potential ligands against a variety of receptors. It has been successfully applied to design new inhibitors or bioactive compounds and, consequently, to improve the RDD efforts.

AutoDock4.2 employs a stochastic search algorithm that generates random conformations of receptor and ligand [MOR10]. It has an empirical force-field-based scoring function that incorporates a set of atom types and charges to estimate free energies of binding (FEB) [MOR09]. The main steps to perform molecular docking simulations between a target molecule and a ligand with AutoDock4.2 are illustrated in Figure 4.1 and summarized below.

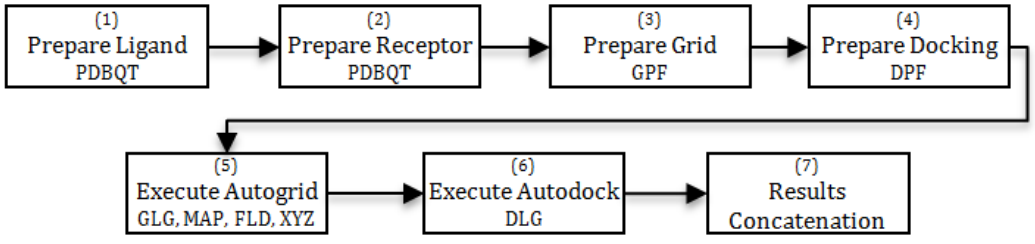


Figure 4.1: Steps in the AutoDock4.2 sequential execution. The function name and the output files are represented in each step to execute a single molecular docking simulation with AutoDock4.2.

1. Preparation of ligand and receptor files (steps 1 and 2 in Figure 4.1). In this step, from a coordinate files, generally in PDB format, it is possible to insert the polar hydrogen atoms, partial charges and atom types [MOR10]. Furthermore, the torsions degree can be selected to limit the flexibility of the ligand. After all configured, a PDBQT file is created for both receptor, and ligand.
2. Configuration grid and docking files (steps 3 and 4 in Figure 4.1). After ligand and receptor files created, the grid and docking files are prepared. Each file contains parameters that are specified in this stage to execute the third and fourth steps. A GPF file is generated to execute *autogrid* and a DPF file is generated to execute *autodock*.
3. Autogrid execution (step 5 in Figure 4.1). For each atom type present in the ligand that is being docked a grid maps are calculated by *autogrid* [MOR10]. This step is essential because, besides this pre-calculation to be the *autodock* input parameters, it helps to make the docking computations fast. To run *autogrid* it is necessary a grid parameter file (GPF extension) which specifies some parameters, for instance the grid point spacing, the grid centre, and the name of output files written during the grid calculation [MOR10]. The grid output files are a log file with grid calculation, and other information about the coordinates and specifications to create a grid box. The amount of grid map files depends on the number of atom types in each small molecule or ligand.
4. Autodock execution (step 6 in Figure 4.1). Finally, *autodock* is executed by one of the search methods. The docked conformations and FEB results are independently generated by the search algorithm employed to perform a number of receptor and ligand interactions. The algorithms used by Audodock4 are Lamarckian Genetic Algorithm (LGA) [MOR98], Genetic Algorithm [MOR98] and Simulated Annealing [GOO96]. These search algorithms have a parameter for determining the amount of runs (number of evaluations) that will be used to identify ligands by ranking the relative binding energy of small molecules [MOR10]. There are several parameters to improve the docking performance. However, different values can be selected depending upon the search approach chosen. Besides the input parameters to specify the docking calculation, the grid maps and a ligand files are also specified in input file (DPF extension) to execute docking simulations. At the end of the execution, an output file (DLG extension) is produced with final docked coordinates, final binding energies, such as RMSD and FEB, and other values which belong to each evaluation executed.

The third and fourth steps above are a limitation of AutoDock4.2 because of the high computational cost required to perform several docked conformations by means of a stochastic search function. Furthermore, AutoDock4.2 is originally designed to execute in a single core machine. The total time spent to execute *autogrid* and *autodock* of only one snapshot of the FFR model and one ligand in a CPU with 2.13 GHz and 2GB RAM is around 3 minutes. However, the number of

conformations a FFR model can have may reach thousands to hundreds of thousands to millions of conformations. The high computational cost involved in performing practical virtual screening of a FFR model against a library of thousands to millions of thousands of ligands, the ZINC Database [IRW05] is an example, may be it impractical. For these reasons, this study constitutes an attempt to make docking simulation of a FFR model against a ligand, and in the future a library of ligands, a viable step in the RDD process. For that, we make intense use of parallel architectures on MPI cluster environments which simultaneously control and execute molecular docking.

### 4.1.2 Atlântica Cluster

Atlântica cluster is one of the two HPC architectures used to execute massively parallel molecular docking simulations of a FFR model. It consists of 10 nodes connected by a fast network system. Each node contains two CPUs Intel Xeon Quad-Core E5520 2.27GHZ with Hyper-Threading and 16GB of RAM, aggregating 16 cores per node (16 hardware threads running on 8 physical cores, 2 threads per core) and 160 cores in total. The cluster is connected by two gigabit Ethernet network; one for communication between nodes and another for management. The Atlântica cluster is hosted in the "Laboratório de Alto Desempenho da PUCRS" (LAD)[4]. LAD supplies high performance computational resources for the academic community and research groups at PUCRS.

The cluster provides several software and hardware resources necessary to execute parallel programs in a secure and usable manner. It uses Torque[5] to manage the allocation and control of the access to the nodes through queues monitored by job-scheduling policies. To allocate one or more nodes with this control it is necessary to provide information about the type of parallel execution to be processed, e.g.; the number of nodes allocated, allocation time, access mode (exclusive or not exclusive) and amount of cores/threads.

Figure 4.2 shows the Atlântica cluster network architecture where the nodes are represented as Atlântica machines from 1 to 10. The server machine called Marfim gives local access to the cluster. Marfim is also located in LAD and runs as an image to the hosts of the Atlântica cluster. It provides all the needed support for preparing and controlling executions in the cluster, such as the file system operations, the libraries to compile and execute the parallel applications.

[4] http://www.pucrs.br/ideia/lad.
[5] http://www.adaptivecomputing.com/products/torque.php.

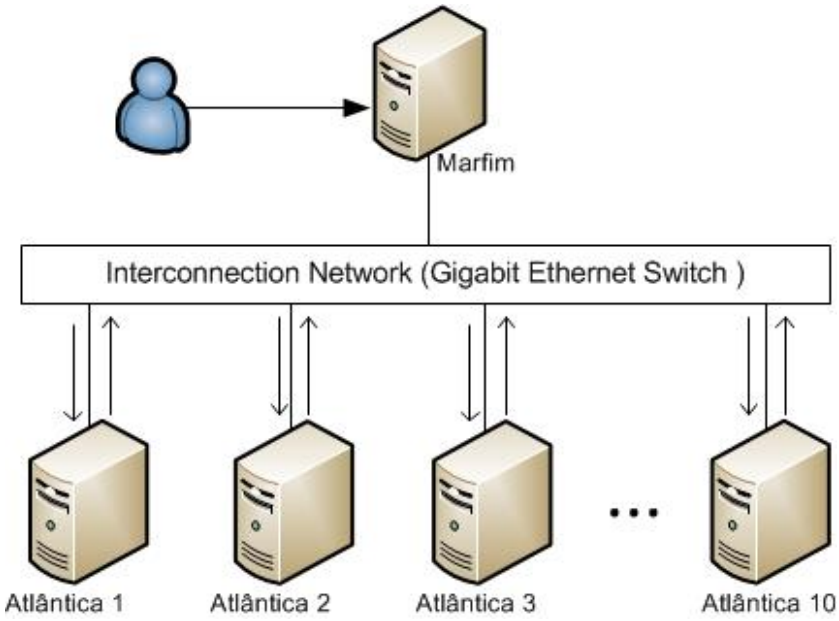


Figure 4.2: Atlântica cluster's network architecture. The users access the Marfim server that provides access up to 10 machines from the Atlântica cluster.

The Atlântica cluster includes a multi-core processor architecture to leverage the performance of large-scale executions and optimize applications by means of distribution of workloads. Taking advantage of the parallel processing power of a cluster may likely increase the execution of molecular docking simulations of FFR models by FReMI.

### 4.1.3 HPC on Amazon EC2 Instances

Another HPC architecture used in this study to execute FReMI's tasks in parallel is a virtual cluster built with Amazon EC2 instances. The AMI used as base is ami-da0cf8b3[6] with Ubuntu 10.04 Server 64-bit and 6 different instance types with different features. Only a High-CPU extra large instance (c1.xlarge) was prepared and configured to create a new instance which was used to execute FReMI. Due to the sharing of instances by more than one user, Amazon EC2 provides a basic measure of processing to identify different types of physical machines. A rating of one EC2 compute unit is a unit of CPU performance where the CPU capacity corresponds to 1.0 - 1.2 GHz 2007 Opteron or 2007 Xeon processor [MUT08]. Amazon's EC2 c1.xlarge instance has 8 virtual cores with 2.5 EC2 compute units each, 7 GB of RAM and 1,690 GB of local instance storage.

After launching the instance the internal and external dynamic network addresses are assigned by the EC2 environment [MUT08] to allow access using SSH in each instance. Furthermore, the virtual machines can be configured by the user to install and run software as root user. Afterwards, users can create a new image from the AMI base and launch instances of their own AMI over the

[6] http://aws.amazon.com/amis/4348.

internet and interact with them. This is the approach used in this study to create an MPI cluster pool on Amazon EC2 and subsequently to execute FReMI. The steps necessary to construct this MPI cluster environment are described below.

1. Launch an instance from base image (ami-da0cf8b3).
2. Prepare the base instance. In this step all essential software to run the FReMI middleware are installed. The data files for the scientific experiment, as well as the program files used to execute FReMI, are stored. The software installed in this machine were: GCC 4.6.2 [STA10], MPICH2 [MPI09], libxml [XML03] and libcurl [CUR11] libraries, AutoDock4.2 [MOR09] package and network configuration tools.
3. Create a new image from step 1. At this time, the new AMI is created with the same software and data of the original AMI, but with a different AMI identification.
4. Launch instances from image created on Step 3. In this step it is started as many instances as necessary to execute tasks in parallel.
5. Prepare MPI Cluster Environment. An MPI cluster pool must be configured with the launched instances. In this regard, a ring of multiprocessor daemons are activated to provide communication among the instances. Additionally, a Network File System (NFS) is created to share the same file directory system amongst all virtual machines.
6. Finally, execute the FReMI middleware on Amazon instances.

The data files are stored on Amazon Elastic Block Store (EBS). EBS provides block level storage volumes for use with Amazon EC2 instances. Amazon EBS volumes are off-instance storage that persists independently from the life of an instance [AWS12]. This means that if the instance is rebooted EBS will keep the stored data. However, if the instance is terminated the EBS storage volume terminates too. Figure 4.3 shows the cluster pool created on Amazon EC2's instances where the same files directory is shared by NFS among the instances to store all input and output files used during run time of FReMI. In this pool, all data are stored on EBS of the master machine and all the instances have permission to read and write in this shared directory. All data is kept stored in the shared directory even if a slave instance terminates. However, if the master instance terminates all data are lost because the master instance EBS volume terminates at the same time. Thus, the S3cmd[7] source code and package is used to replicate the most important information from Amazon EC2 to Amazon S3 bucket[8].

[7] S3cmd is an open source project available under GNU Public License v2 and free for commercial and private use. It is a command line tool for uploading, retrieving and managing data in Amazon's S3. S3cmd is available at http://s3tools.org/s3cmd.

[8] Bucket is the space to store data on Amazon S3. Each bucket is identified with a unique bucket name.

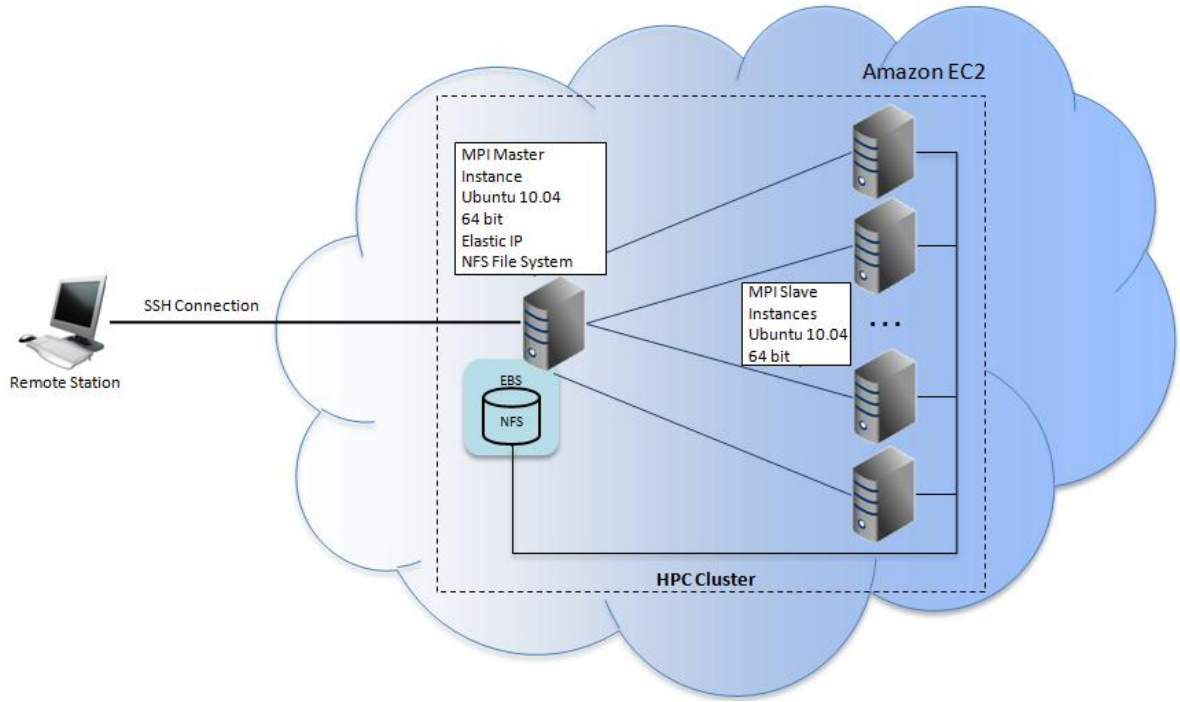


Figure 4.3 MPI cluster environment created to execute FReMI on Amazon EC2. The remote station represents the machine outside Amazon EC2 used to connect the MPI Master Instance by an SSH connection. The MPI Master Instance is the machine that manages the MPI slaves during the FReMI execution. It also holds the FReMI source code and the I/O files stored on the Amazon Elastic Block Store (EBS). All instances may access EBS through NFS.

Although Amazon provides a range of command-line utilities for bundling and controlling EC2 images and instances, there are several tools and software packages that have been developed to aid the use of AWS. This study used the EC2 command line tools[9] to manage the instances, such as manipulating security groups, launching and terminating instances. In addition, Elastic Fox [AEF11] and S3Fox [ASF11] extensions are employed to control EC2 and S3 using a GUI instead of command line. Both are extensions of Mozilla Firefox web browser and can provide access through this tool. Figure 4.4 shows Elastic Fox at work. It allows the user to control AMIs, launch, monitor and terminate instances. Figure 4.5 illustrates the use of S3Fox which, besides transferring data between S3 and the local machine, it also allows creating buckets, deleting files, and setting permissions.

[9] http://developer.amazonwebservices.com/connect/entry.jspa?externalID=351.

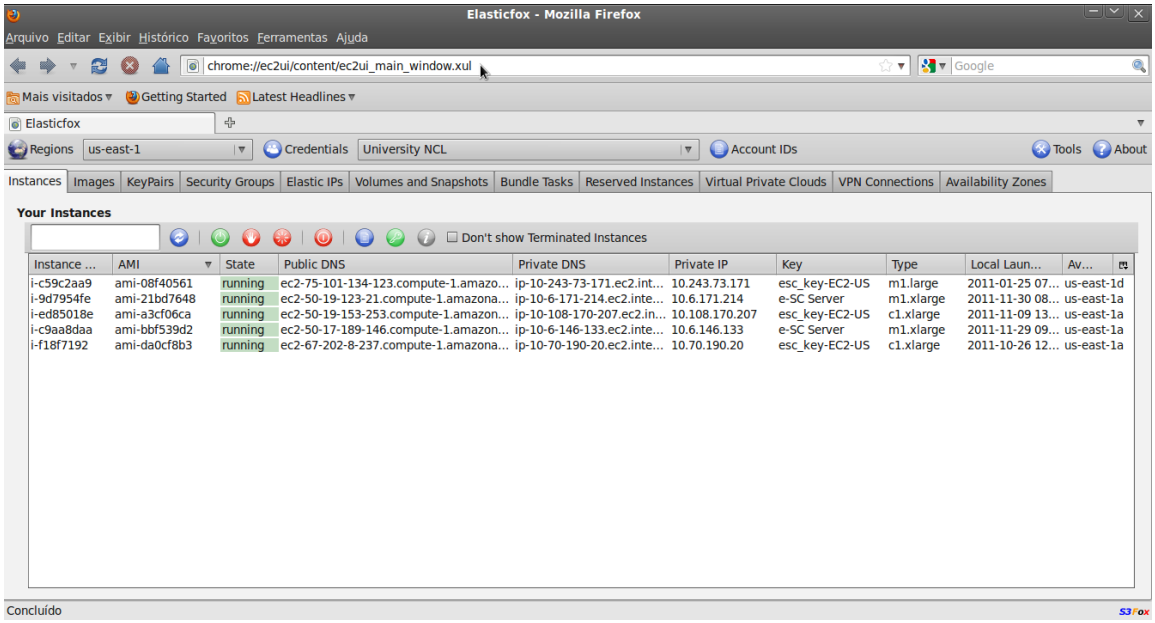


Figure 4.4: Elastic Fox interface showing five running virtual machines and their features. This interface also allows launching, rebooting and terminating the running images.

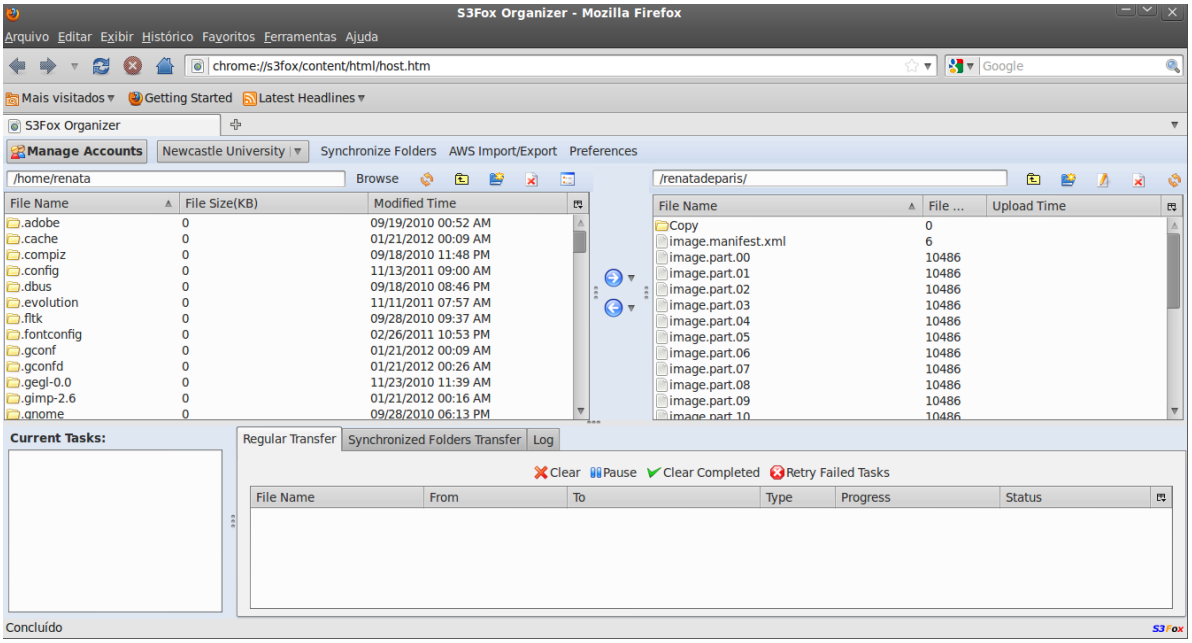


Figure 4.5: S3 Fox interface. The right box shows the files inside the S3 bucket. The left box shows the file on the local machine. This interface allows transferring files to and from Amazon's S3.

## 4.2 Methodology Applied to Build the RFFR Model

The snapshots of the FFR model used in this study (see Section 2.1.1) are derived from a MD simulation trajectory of the receptor. Even though this approach is considered the best to mimic the natural behavior of ligands and receptors [ALO06], its dimension or size may become a limiting step. Moreover, the high computing cost involved could also turn it unfeasible to perform practical virtual

screening of such a receptor model. For these reasons, new methods have been developed to assist in the simplification or reduction of a FFR model to a RFFR model. The primary rationale of this approach is to eliminate redundancy in the FFR model through clustering of its constituent conformations [MAC10b]. This is followed by the generation of subgroups with the most promising conformations via the P-SaMI algorithm [HUB10]. FReMI was developed to aid in this task and its results are expected to support faster execution of molecular docking simulations in more realistic virtual screening experiments. To achieve this FReMI relies on data derived from the following approaches:

1. Data mining techniques to achieve the groups of snapshots [MAC10b], [MAC11a] and [MAC11c].
2. The Self-adapting Multiple Instances Partner (P-SaMI) to classify and prioritize the different groups of snapshots [HUB10].
3. The W-FReDoW web environment which uses P-SaMI and prepare the docking input files [DEP11].

### 4.2.1 Snapshots Clustering of a FFR Model

The group of snapshots used in this study was generated using clustering algorithms developed by Machado [MAC11a]. In this approach the author executes 10 clustering algorithms with different similarity functions over the FFR model to find patterns that define groups of similar conformations [MAC10b]. The process of knowledge discovery in database is used to analyze the FFR molecular docking data stored on FReDD (Flexible Receptor Docking Database) [WIN09]. FReDD database stores receptors and docking experiments results to discovery interesting information under a FFR model. It includes, but is not limited to, the best FEB and RMSD of each snapshot calculated with AutoDock3.0.5 [GOO96].

Machado [MAC11a] used, as a rigid receptor, the crystal structure of the InhA enzyme from *Mycobacterium tuberculosis* [DES95] (PDB ID: 1ENY). The FFR model of InhA was derived from its MD simulation trajectory [COH11] [SCH05]. Four different ligands were employed in her docking experiments, specifically: nicotinamide adenine dinucleotide (NADH) [DES95], triclosan (TCL) [KUO03], pentacyano(isoniazid)ferrate(II) (PIF) [OLI04], and ethionamide (ETH) [WAN07]. The following rule emerged as a result of this study: if a snapshot is associated with a docking with significantly negative FEB and small RMSD values, for a unique ligand, it is possible that this snapshot will interact favorably with structurally similar ligands [MAC11a].

According to Machado et al. [MAC11c] the groups of snapshots, which were related to different classes of FEB values, are useful to identify the most promising receptors conformations, and also the ones that can be discarded for docking simulations, considering ligands with similar

characteristics. As a consequence of this approach, the groups of snapshots are post processing using the P-SaMI data pattern to select the receptor conformations and reduce the complexity of the FFR model [HUB10] [MAC11c]. The next section presents a detailed description of the P-SaMI data pattern and its application in the processing of a FFR model to a RFFR model by FReMI.

### 4.2.2 Prioritization of the Subgroups of Snapshots – P-SaMI

The P-SaMI data pattern was employed in the identification of the most promising conformations in the different groups of snapshot of a FFR model identified by Machado [MAC11a]. P-SaMI is the acronym for Pattern-Self-adaptive Multiple Instances – a data pattern for scientific workflows developed by Hübler [HUB10]. The purpose of this approach is to define a pattern which is able to dynamically perform the selection of the FFR receptor's conformations with the aim of producing a RFFR model. As a consequence of this reduction, one can eliminate the exhaustive execution of dockings simulations of a FFR model without affecting its quality [HUB10][MAC10b].

The preliminary step of P-SaMI [HUB10] is to capture the clustering of snapshots independently of the similarity function used [MAC11a]. Next, it tags the data as follows: the snapshot identifier, the group identifier to which the snapshot belongs, the subgroup identifier, and its status and processing priority. A subgroup is a percentage sample of a group. The subdivision of the groups in to subgroups is performed before and during the execution of docking simulations for the status of the subgroups can vary while their snapshots are being processed. As defined by Hübler [HUB10] the status can be active (A), finalized (F), discarded (D) and changed priority (P). Only snapshots belonging to the subgroups with status "A" are processed. P-SaMI also uses the minimum quantity of conformations to be processed and the percentage of the sampling that make each subgroup, which are defined by user.

The docking simulation execution of each snapshot begins after splitting the conformations within each group into subgroups (Figure 4.6). The outcome is the "Docking Result". These results contain the best FEB value for each snapshot. Afterwards, P-SaMI uses some evaluation criteria, e.g., FEB's averages and standard deviations, to analyse the results and to determine the status and priority of the snapshots waiting to be processed. The status and priority are attributed to each subgroup. Thus, if the "Docking Results" of a subgroup present an acceptable FEB value (the more negative the better) then that subgroup is credited with a high priority. Conversely, the subgroup has its priority reduced or its status changed to "D". In the latter case, the snapshots waiting to be processed are discarded; unless all the snapshots of that subgroup have already been processed (status "F"). The status information is paramount to decide whether a snapshot will be processed or not.

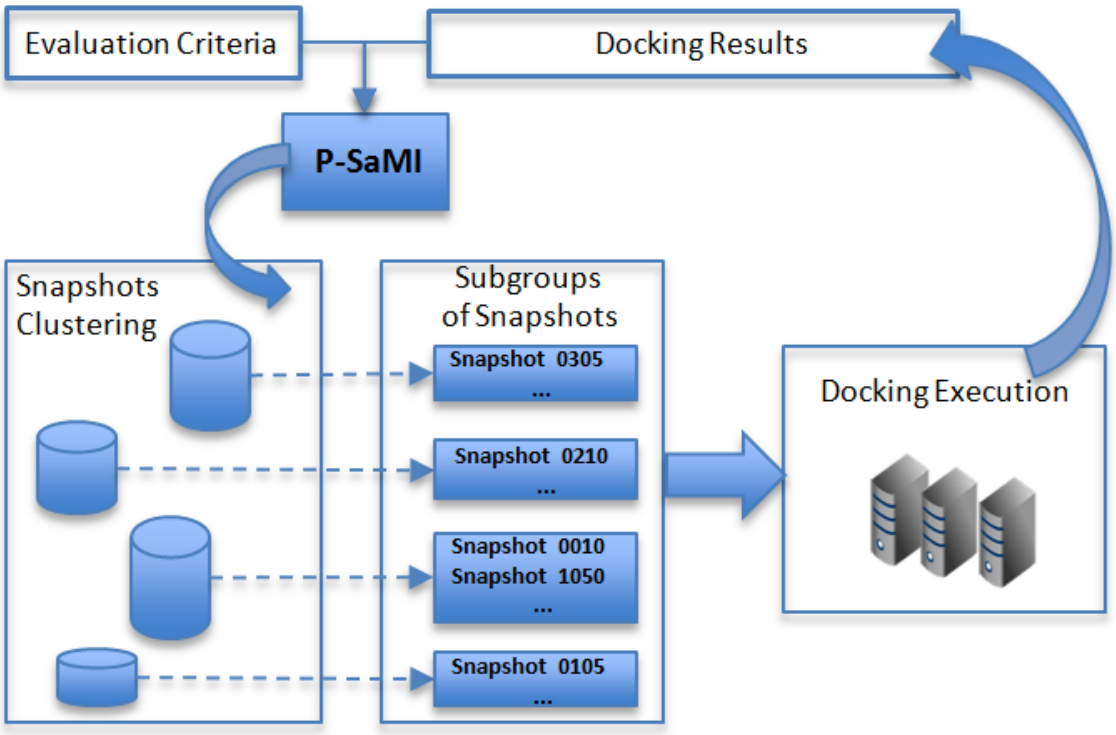


Figure 4.6: Model of P-SaMI data pattern execution. Clustered snapshots [MAC11a] are divided into subgroups using the P-SaMI data pattern in W-FReDoW. Molecular docking simulations are executed on these subgroups. P-SaMI analyses the docking results, based on some evaluation criteria, to select promising conformations from subgroup of snapshots.

Hübler [HUB11] organized the subgroups' priorities in three levels: high (3), intermediate (2) and low (1). The subgroups of snapshots with high priority are processed before that ones have low priority. Thus, the snapshots that gradually present high-quality results during docking execution are considered by P-SaMI as promising conformations. Consequently, its subgroups receive high priority and more processors to execute docking in parallel. On the other hand, the subgroups of unpromising snapshots receive low priority and fewer processors.

From P-SaMI it is possible to make use of groups of conformations [MAC11a] and classify the clustered snapshots in different conformation levels (most or less promising). In this sense, different snapshots – within the same group – with highly similarity can be classified under a classification criterion most stable and deterministic if they are processed by molecular docking simulations [HUB10]. For example, let consider two different subgroups: subgroup 01 (SG01) and subgroup 02 (SG02). If 20% of processed snapshots of the SG01 presents a constancy of poor results in molecular docking simulations, this group may be discarded (status D) or has its priority reduced (low [1] or intermediate [2]). On the other hand, if the SG02 with the same percentage of snapshots processed presents a gradual increase on the results, it is possible to suggest that this group contains promising conformations, and consequently it will have increased priority and the percentage of snapshots that will be processed. Furthermore, if the SG01 has not shown good results, the last group may use the processes which had been reserved for G01 before changing its level. Hence, selection of the most

promising conformations can reduce the total execution time in the docking experiments, and, consequently, accelerate the RDD efforts [HUB10] [MAC10b].

The goal for using P-SaMI [HUB10] in this study is to make full use of its pattern data to help FReMI improves the performance in the molecular dockings simulations using a FFR model. The main suggested future works by Hübler [HUB10] are: develop a scientific workflow component which manipulates snapshots data to identify promising snapshots and; make use of a HPC environment to execute the molecular docking experiments. Both suggestions are being developed in a parallel and cooperative way. The workflow, developed as a web server environment, is the research theme of another M.Sc. student in the same working group. The middleware to handle molecular docking simulations in HPC environments is presented in this dissertation. The web server is called W-FReDoW and its main functions to work with FReMI will be explained in the next section.

### 4.2.3 W-FReDoW: A Web Server to Prepare Files for Molecular Docking Simulations and to Analyse Docking Results of a FFR Model

Even though FReMI is capable to work on its own, it also includes functionalities that allow interaction with W-FReDoW.

Figure 4.7 summarizes the W-FReDoW workflow environment. W-FReDoW is a web server built in parallel with FReMI to aid in the reduction of the execution time of molecular docking simulations of FFR models. Its central role is to select promising snapshots subgroups from a FFR model by means of the P-SaMI data pattern. In addition to that W-FReDoW includes two components and a database that stores the docking results and gives provenance about the snapshots during execution time.

- The Client layer represents the web interface that integrates W-FReDoW and FReMI executions. It performs three activities: 1) configuration of molecular docking, P-SaMI and FReMI parameters; 2) initialize the W-FReDoW and FReMI execution and; 3) analyse the docking results.
- The Molecular Docking component executes the pre-docking steps required for AutoDock4.2 (steps 1 to 4 in Figure 4.1). Its activities must be executed before or during the execution time.
- The P-SaMI component implements the P-SaMI data pattern. It will be detailed in Chapter 5.
- The FReDD Extension represents the database used to provide provenance about data generated by the components described above. This database is an extension of FReDD [WIN09].

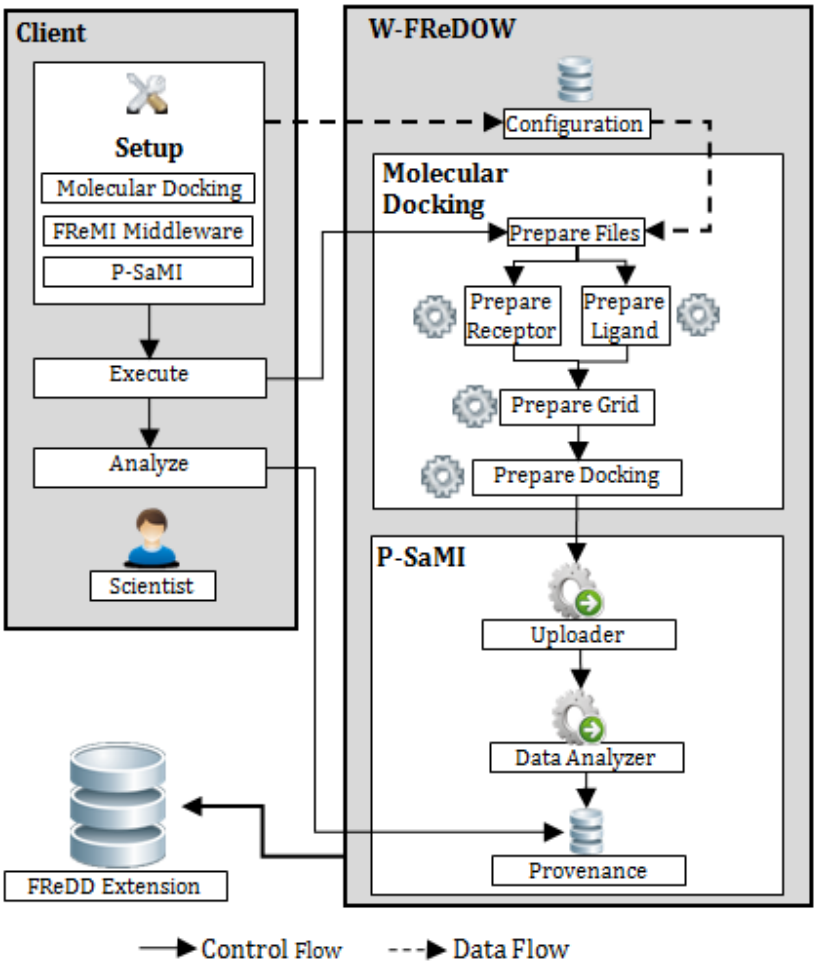


Figure 4.7: Client and W-FReDoW conceptual architecture. The client layer represents the steps performed by scientist before and after the execution. Molecular docking holds the functions prepare the snapshots to be executed on FReMI. P-SaMI represents the functions to send, analyse and store the docking files. FReDD Extension is the database to keep the W-FReDoW provenance.

## 4.3 Methods Used to Develop FReMI

The C programming language was used to develop the FReMI middleware due to the low-level access to memory, the flexibility to support a number of different functions from the central library and portability. This section presents the C libraries used to develop FReMI. It outlines the MPI and OpenMP functions for the parallel tasks and the functions to establish the communication with W-FReDoW via internet.

### 4.3.1 The MPI Parallel Program Model

Message-Parsing Interface (MPI) is a parallel programming paradigm used to move data from the address space of one process to that of another process through cooperative operations on each process [MPI09]. MPI provides an API to establish communication characteristics between nodes by

specification of operations through message transfer. Furthermore it can be accessed in C, C++, Fortran-77 and Fortran-95. The MPI used in this work is MPICH2 version 1.2.1.

The main advantages for using an MPI model are portability, efficiency and ease of use. A MPI programming model is the correct paradigm to use for all levels of parallelism available in the application and that the application topology can be mapped efficiently according to the hardware topology [RAB09]. For instance, exploit the master-slave paradigm under HPC machines. It also can be used by MIMD programs providing features that improve performance on scalable parallel computers with inter-processor communication hardware [MPI09]. MPI is the appropriate parallel programming model used in different cluster environments for establishing safe communication and for supplying high portability. For example, it is used in virtual machines launched by EC2 instances and the dedicated machines in the cluster Atlântica.

### 4.3.2 The OpenMP Parallel Program Model

OpenMP Multi-Processing (or only OpenMP) is an API to exploit the parallelism on shared memory architectures though parallel programming models which simultaneously execute multiple threads. It is based on library routines, environment variables and a set of compiler directives which expand the C, C++ and FORTRAN support languages [OMP11] [SMI01]. There are several compilers that support the OpenMP API, and as MPI, these compilers include their command line option which allows interpretation of all OpenMP directives.

In contrast to MPI, which only sends a message to two or more machines and wait for a process to receive it and to execute the tasks, OpenMP offers a more efficient parallelisation strategy within a node through fork-join model of parallel execution [OMP11] [SMI01]. Although this API only runs on shared memory machines, the communication is implicit and it is relatively easy to implement OpenMP applications [SMI01]. The key reason for using OpenMP in this study is to take advantage of the SMP nodes that make the HPC environments. Making use of inter-node multi-threads on Cluster OpenMP is important to enhance the performance of automated molecular docking simulations with FReMI.

### 4.3.3 The Hybrid MPI-OpenMP Programming Model

Hybrid MPI-OpenMP programming exploits the explicit inter-node communication through message passing on distributed systems and the high performance of SMP’s shared memory among threads by an implicit synchronization state. According to Rabenseifner et al. [RAB09] the hybrid MPI-OpenMP enforces the domain decomposition to be two-level algorithm. On MPI level, a coarse-gained domain decomposition is performed. Parallelization on OpenMP level implies a second level domain decomposition. This model closely maps to the architecture of an SMP cluster, the

parallelisation occurring between the SMP nodes and the OpenMP parallelisation within the nodes. [SMI01]

The most important advantage of using a hybrid MPI-OpenMP programming model is that introducing MPI into OpenMP not only is able to reduce the amount of data to be communicated and the total number of MPI calls, but also improve the load balancing while maintaining a high level of parallelism. Further, this model has the potential to exploit the scheduling of the effective parallelism, with distributed memory programming for the coarser grain parallelism and shared memory programming for the finer-grained [SMI01].

The master-slave paradigm [BAN04] is used to allocate a large number of independent, equal-size tasks to a HPC environment. In this concept a specific node, referred to as the *master*, holds a large collection of independent, identical tasks to be allocated on the cluster. The *master* node needs to decide which tasks to perform itself, and how many tasks to forward to each of its neighbours [BAN04]. These neighbours are named *slaves* and, in contrast to the *master*, which needs to execute and handles all the tasks, it only executes the jobs received from a *master* node. For example, Figure 4.8 shows the communication and the workload among the nodes used in a master-slave programming concept. Bidirectional arrows from and to the *master* node send the workload to the *slaves* and receive a return when they are idle or complete their works through message transfer (MPI). Unidirectional arrows indicate the amount of cores, only four in this example, per node for distributing the tasks for parallel execution using OpenMP programming scheme.

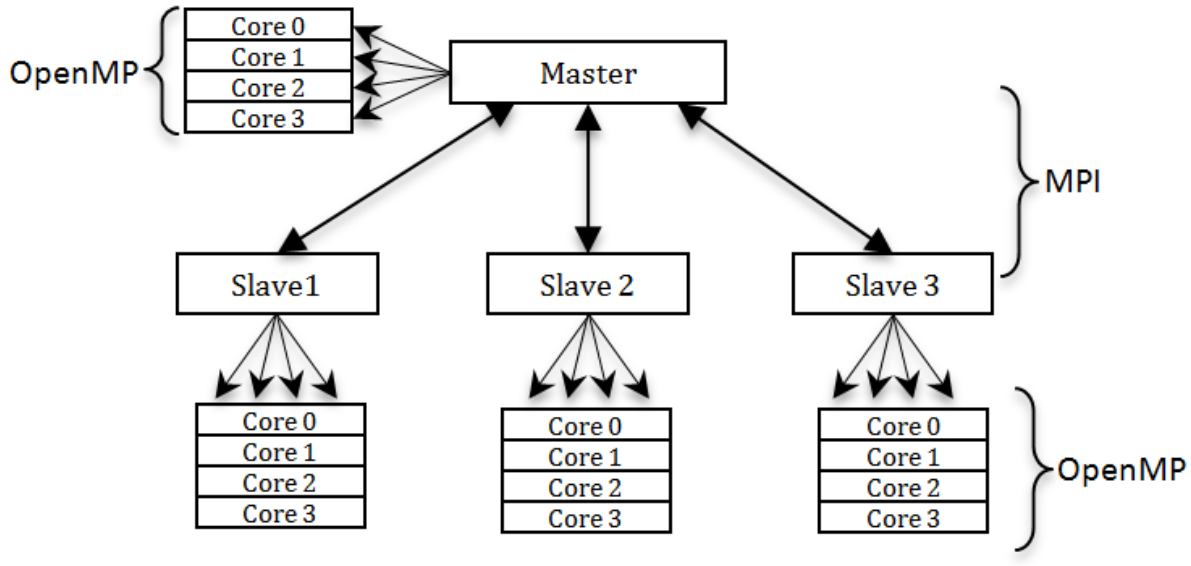


Figure 4.8: Schematic representation of a hierarchical hybrid MPI-OpenMP programming model. Bidirectional arrows show the MPI programming scheme using the master-slave paradigm. Unidirectional arrows illustrate the number of OpenMP cores to be used in the parallel execution of a workload on each SMP node.

According to Rabenseifner [RAB09] a hybrid MPI-OpenMP programming model employs multi-threaded MPI processes to distribute the tasks in a standard dynamic load balancing. Thus, the implementation of the hybrid parallel programming model in this study aims to reach a high level of workload balancing, using a tasks' distributing algorithm based on the master-slave paradigm of HPC environments. To achieve this objective the following steps are performed. First, MPI functions are used to send a message from the *master* to the *slave* nodes in the cluster pool. Second, the *slaves* receive the message and the information about the amount of tasks that need to be executed using thread parallelism inside each SMP node, including the *master* node. Third, inside each SMP node, the allocating of the tasks is performed by OpenMP and one task is executed for each processor. Finally, if all processors from a specific SMP node finish the executing of its tasks, it sends a message to *master* node notifying that it is idle and more work must be sent. In this stage, if there are more tasks to be executed the process starts again, else a "MPI end message" is sent to communicate to every other node to stop execution.

### 4.3.4 XML Files

The dataset's logical structure of the different groups of snapshots of a FFR model is specified in a physical file system via a subset schema in Extensible Markup Language (XML). XML is a markup language that allows the creation of documents with data arranged hierarchically in a tagged layout. Element tags render XML documents inherently usable for storing and sharing textual information with structural/semantic annotations [XML03]. Furthermore, it is able to integrate with other languages, interconnect with different databases and communicate with clients and servers on internet and intranet networks. Independently of the computational infrastructure or operational system that the FReMI is located, the XML file is able to store, manipulate and recognize the information representing a FFR model.

Libxml2 [XML03] is the library supported by the C programming language and used in this study to handle XML files. This library includes functions that allow creating, reading and writing a metalanguage to design markup languages. FReMI uses two XML files for handling the molecular docking executions. One file identifies the groups of snapshots. The other states the priority and status of the groups of snapshots which are updated by P-SaMI through the w-FReDoW web server. The structure and specifications of such files are presented in more details in Sections 5.1.2 and 5.1.3.

### 4.3.5 Communication Protocols

FReMI uses the Hypertext Transfer Protocol (HTTP) to set up communication with w-FReDoW (as shown in Section 4.2.3). The Representational State Transfer (REST) [FIE00], similar to that adopted by sites, e.g., Facebook and Google, is used to sign REST requests and communicate

representations of resources. REST is the web architecture style used in software engineering to specify the communication model using HTTP calls between distributed software.

According to Fielding [FIE00] one of the mains goals of REST is to support the introduction of versioning requirements and rules for extending each of the HTTP protocols. POST is one of the protocols supported by HTTP and is also used in this work. HTTP POST sends the processed docking results from FReMI to the W-FReDoW web server. The functions of libcurl library [LIB11] were used to call the HTTP POST protocol through C programming language. Libcurl is free and is part of the curl package which has command line tool for transferring data with URL syntax. HTTP POST function was implemented with version 7.23.0.

# 5 RESULTS 1 – FReMI CONCEPTUAL ARCHITECTURE

This chapter describes the first result of this dissertation. It presents the conceptual architecture of the FReMI middleware used to obtain the experimental results (Chapter 6).

As mentioned in the Introduction, the foremost objective of this study is to develop a middleware to assist in the high performance massively parallel execution of molecular docking simulations of subgroups of FFR models' conformations. To achieve this goal, a middleware called FReMI was developed. FReMI is able to distribute, control and monitor the execution of subgroups of snapshots of FFR models in two different HPC environments. It provides resources to interact with a web server through internet communication protocols. These protocols allow FReMI to receive and send messages in run time using the REST services and the FTP network protocol (outlined in Section 4.3.5). Thus, beyond handling tasks to decrease the time in the molecular docking executions in HPC environments, it also simplifies or reduces the FFR model dimensionality by generating a RFFR model.

FReMI uses the Many-Task Computing (MTC) [RAI10] paradigm to address the problem of executing multiple parallel tasks in multiple processors. MTC is a traditional technique used by the scientific community to denote a model of loosely coupled computations in which large volumes of data are exchanged among tasks via files, databases or XML documents, or by a combination of these [RAI10]. Moreover, MTC is an efficient approach to share multiple tasks and manage the scalability and granularity on different computing paradigms as in HPC environments. Figure 5.1 details the FReMI conceptual architecture and its interaction with the W-FReDoW web server.

A preliminary conceptual architecture of FReMI and W-FReDoW was published earlier as an LNBI-LNCS extended abstract on the 2011Brazilian Symposium on Bioinformatics [DEP11]. The purpose of the MTC environment remains unchanged, however, the architecture has changed somehow in order to better accommodate the proper execution and analyses of molecular docking simulations of FFR model in HPC environments.

FReMI handles large volumes of data and controls the distribution of tasks in the parallel execution mode by HPC environments. Its five components were designed to deal with both large scale task's distribution and data handling. The single components are Start and the HPC Environment. Start begins the execution of FReMI. HPC Environment denotes the two clusters used in this study: Atlântica and the virtual cluster on Amazon EC2 (outlined in Sections 4.1.2 and 4.1.3 respectively).

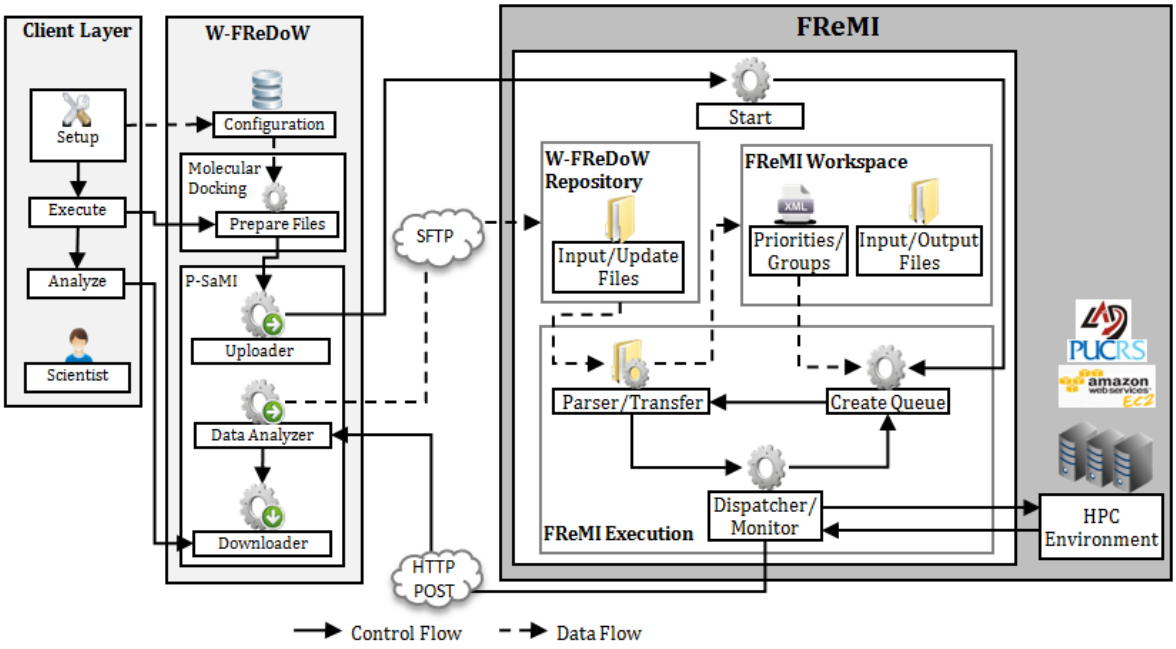


Figure 5.1: Conceptual architecture of FReMI and its interactions. The two left boxes show the tasks to be executed by a user on the web server, which sends and receives messages to and from FReMI (right-hand box) via internet communication protocols. The HPC Environment represents the MPI Clusters on Amazon EC2 and LAD's Atlântica. The W-FReDoW Repository, FReMI Workspace, and FReMI Execution are detailed in the text below.

As shown in Figure 5.1, the remaining components are distributed in three sets: W-FReDoW Repository, FReMI Workspace and FReMI Execution. The first two sets comprise the file system used to store data during FReMI execution. The last one is the most important for it synchronizes all the functions performed by the other FReMI components. The next sections will detail them.

## 5.1 W-FReDoW Repository

The W-FReDoW Repository contains the Input/Update Files. The Repository directory stores the files sent from W-FReDoW through the SFTP network protocol. It consists of: input files to execute the *autogrid4* and *autodock4* applications; one control file to identify the status and priorities of the subgroups and their snapshots; and update files to change the priority and status of the subgroups of snapshots whenever needed. The next subsections describe these files and how they are handled in the Repository directory by FReMI.

### 5.1.1 Input Files

The contents of the input files for *autogrid4* and *autodock4* programs have been outlined separately in Section 4.1.1. These files are created by W-FReDoW in which, for each snapshot of the FFR model, one PDBQT, GPF and DPF files are used to execute one molecular docking simulation

with a single ligand. For example, to execute an experiment with a FFR model, composed of 3,100 snapshots, W-FReDoW creates: 3,100 PDBQT receptor files; 3,100 GPF files; 3,100 DPF files, and one PDBQT file for the ligand. As a result, a total of 9,301 files are received by the Input/Update Files Component for the execution of this experiment.

The input files of a whole experiment, like the one shown above, are sent to FReMI before starting its execution. Otherwise, file transfer during FReMI execution, would turn it very expensive because PDBQT files are usually large, of the order of 300 Kbytes each. For the FFR model example above it would total approximately 1.0 GB. For each snapshot the GPF and DPF files have together only 4.2 KB.

FFR models are becoming ever large [ALO06] [COZ08]. Currently we have FFR models with over 40,000 PDBQT files [COS11]. Thus, to avoid long waiting times to transfer files, the model is placed in the W-FReDoW Repository before running the application.

### 5.1.2 Control File

The different subgroups of snapshots generated by data mining techniques [MAC11a] are stored in the control file called "groupSnap.xml". In this file each tag represents a data parameter that identifies the grouping structure of the snapshots [MAC11a] and their settings according to P-SaMI [HUB10]. Figure 5.2 shows part of a sample file with three root tags described as:

- *experiment*: identifies each molecular docking simulation of a FFR model. The experiment identification (id) is a unique number created and controlled by W-FReDoW for each new simulation. This tag includes different subgroups of snapshots.
- *subgroup*: specify the information of the subgroups. The *idSubgroup* tag identifies the key number of each subgroup. The *stat* tag denotes whether a subgroup of snapshots is active (A), finalized (F) or discarded (D) and the *priority* tag indicates how much promising are the snapshots belonging to that subgroup, in a 1 to 3 priority scale, as defined by PSaMI [HUB10].
- *snapshot*: contains information about the snapshots. The *idSnap* tag represents the key number of each snapshot. The *status* tag denotes whether the snapshot is waiting to be processed (P) or has been processed by the HPC environment (Q).

For each docking simulation, with one FFR model and a single ligand, a new XML control file is created by W-FReDoW and sent to FReMI. This XML file is essential because FReMI Execution components (see Figure 5.1) use it to create the queues of tasks and maintain FReMI with the subgroups of snapshots updated according to P-SaMI parameters.

```xml
<?xml version="1.0"?>
<experiment id="1">
    <Subgroup>
        <idSubgroup>G1L1</idSubgroup>
        <stat>A</stat>
        <priority>2</priority>
        <snapshot>
            <idSnap>001055</idSnap>
            <status>P</status>
        </snapshot>
        <snapshot>
            <idSnap>000100</idSnap>
            <status>P</status>
        </snapshot>
        ...
    </Subgroup>
    ...
</experiment>
```

Figure 5.2: Fragment of a XML file that creates a queue of tasks. This file stores the groups of snapshots generated by data mining techniques [MAC11c] and its parameters according to P-SaMI [HUB10].

### 5.1.3 Update Files

When the status and priority of a subgroup of snapshots change during the execution time, update files are sent from W-FReDoW to FReMI. As shown in the conceptual architecture (see Figure 5.1) the Data Analyzer component represents the functions that create these files by means of the P-SaMI data pattern. It worth remembering that P-SaMI is able to select the most promising snapshots of FFR models from the docking results and some specified evaluation criteria. These analyses are conducted in W-FReDoW and the result is a parameters' set which are used to classify the snapshots to be processed in the HPC environments. The P-SaMI results are XML files sent to FReMI by SFTP. Figure 5.3 illustrates the structure of these files and their physical representations.

```xml
<?xml version="1.0" ?>
<experiment id="1">
    <idSubgroup>G1L1</idSubgroup>
    <priority>1</priority>
</experiment>
```

(a)

```xml
<?xml version="1.0" ?>
<experiment id="1">
    <idSubgroup>G2L2</idSubgroup>
    <stat>D</stat>
</experiment>
```

(b)

Figure 5.3: Fragments of XML files to update the parameters of the subgroups of snapshots. (a) File to update the priority of the subgroups. It contains a tag with the subgroup identification, e.g. G1L1, and a tag with the new priority. (b) File used to identify if a subgroup of snapshot should be discarded or not. As the previous XML file, there are the subgroup identification (G2L2), but instead of the *priority* tag, the *start* tag is used to identify which subgroup should be discarded.

The update files are used by FReMI to handle the queue of tasks. The priority determines how many processors in the cluster are allocated to each subgroup. The status, according to the P-SaMI pattern, indicates which snapshots should be discarded. Therefore, to keep FReMI updated, one XML file is dispatched any time P-SaMI makes a modification on the priority and/or status of subgroups of snapshots. Then, FReMI updates these information in the control file (groupSnaps.xml).

## 5.2 FReMI Workspace

The FReMI Workspace represents the directory structure used to store the huge volume of data manipulated to execute the molecular docking simulations of FFR models. To control the input, output, temp, and control FReMI data files a workspace model was created (Figure 5.4). The job and parameter directories store the input files for the execution of the *autogrid4* and *autodock4*. The result and temp directories store the output files from the executions in the HPC environment. Except the XML file, all other files stored in this workspace are read and generated by *autogrid4* and *autodock4* program. In summary, the workspace contains the following files:

- job: store the snapshot files from FFR models (PDBQT format) and the XML control file (grupoSnap.xml).
- pending: store snapshot files that are waiting to enter the queue of tasks.
- queue: store snapshot files that have been processed by the HPC environment.
- parameter: store the ligand files (PDBQT format), and the input parameter files to execute *autogrid4* (GPF format) and *autodock4* (DPF format).
- result: store output files from *autodock4* (DLG format).
- temp: store temporary files generated during FReMI run time.

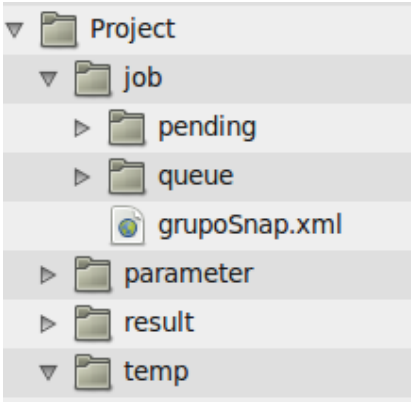


Figure 5.4: Directory structure in FReMI' workspace. Project is the root directory; job, parameter, result and temp are its sub-directories; pending and queue are job's sub-directories.

The input files placed in the W-FReDoW Repository are transferred during FReMI's execution time to its workspace by the Parser/Transfer Component inside the FReMI Execution set of components (Figure 5.1). This component is described next.

## 5.3 FReMI Execution

FReMI Execution constitutes the most important set of components of FReMI. It contains every procedure invoked to run the middleware and controls. The source code of FReMI was written in the C programming language whose libraries are used to run tasks in parallel, read/write XML files, and send the docking results to W-FReDoW.

Figure 5.5 shows in detail the data flow control in the FReMI Execution components. Its main operations are performed by three different components, namely:

- Create Queue. It holds the heuristic function to create queues of balanced tasks. The heuristic function uses the priorities of the different subgroups of snapshots and evaluates which of them must be in or left out of the queues of tasks.
- Parser/Transfer. It transfers the input files from W-FReDoW Repository to FReMI's workspace and updates the XML control files.
- Dispatcher/Monitor. It distributes the parallel execution of tasks in both HPC environments used in this study. Also, it executes the HTTP POST function to send the docking results to W-FReDoW.

### 5.3.1 The Create Queue Component

The three main functions of Create Queue are:

1) Read the "groupSnap.xml" file (Section 5.1.2) to obtain the priorities of the subgroups and their snapshots.
2) Apply the heuristic function to compute the amount of snapshots to be active.
3) Create balanced queues of tasks to be processed in the HPC environment.

The HPC environments employed in this study are multiprocessing clusters for large scale executions. In this context, more than one task may be processed in each node. Each task or job is the sequential execution of the *autogrid4* and *autodock4* programs of only one snapshot of the FFR model with a single ligand.

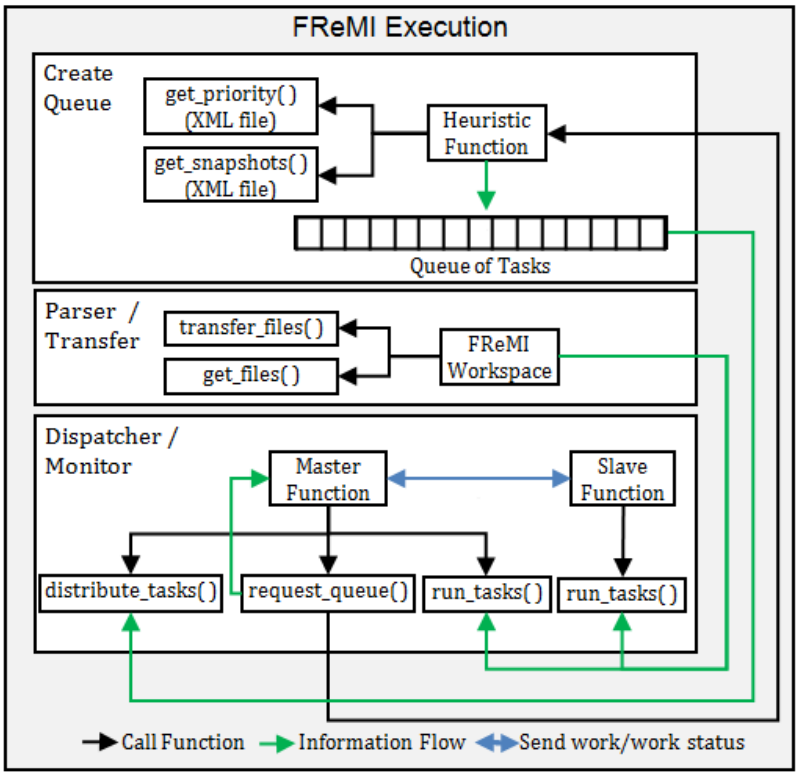


Figure 5.5: Scheme of the FReMI Execution implementation. The Create Queue, Parser/Transfer and Dispatcher Monitor components include the main functions executed by FReMI.

The queue of tasks is built based in the content of the XML control file which contains the heuristic function's parameters to produce the balanced queues during the FReMI execution time. The full execution of a molecular docking simulation of a FFR model, called "an experiment" in this project, requires creating and sending several queues of tasks to the HPC environment.

The heuristic function calculates the maximum number of snapshots than can be supported by each queue. The amount of nodes or machines (N) and the amount of tasks (T) that will be processed per node are used to obtain the queue size (Q) with the following equation:

$$Q = N_{cluster} \times T_{node} \qquad (1)$$

Afterwards, the amount of snapshots per subgroup is calculated in order to achieve the balanced distribution of tasks in every queue created. A balanced queue contains one or more snapshots of an active subgroup. From the subgroups' priorities, it is possible to determine the percentage of snapshots to be included in the queues of tasks. Thus, subgroups with higher priority are queued first than those with lower priority. Equation (2) below is used to calculate the amount of snapshots for a balanced queue.

$$S_g = Q \times \left(\frac{P_g}{\sum_{i=1}^{n} P_n}\right) \qquad (2)$$

$S_g$ is the amount of snapshots of subgroup $g$ placed in the queue. $Q$ is the queue size from Equation (1). $P_g$ is the priority of the subgroup $g$, and $n$ is the total number of subgroups. From Equation (2) one queue of balanced tasks ($Q_{tasks}$) is created with the following equation:

$$Q_{tasks} = \sum_{i=1}^{g} S_i \qquad (3)$$

5.3.2 Parser/Transfer Component

The Parser/Transfer component handles and organizes the files sent from W-FReDoW to the W-FReDoW Repository. These file are further transferred to FReMI Workspace by means of the *transfer_file* function (Figure 5.5). Additionally, this function parses *autogrid4* and *autodock4* input files in order to recognize the files' directory structure. The *get_files* function is always called before creating a new queue and, when necessary, to verify and update the parameters of the subgroups of snapshots.

5.3.3 Dispatcher/Monitor Component

The functions of the Dispatcher/Monitor component are invoked to distribute tasks among the processors/cores of an MPI Cluster environment based on master-slave paradigm. The Slave Function only runs the tasks while the Master Function, aside from running tasks, also performs other two functions: the *distribute_tasks* function which is called when a node/machine asks more work and; the *request_queue* function which is called when the queue of tasks is empty.

This chapter showed the conceptual architecture that was implemented to execute the molecular docking simulations of FFR models. Chapter 6 shows the results obtained from FReMI execution on two HPC environments: the Atlântica cluster and virtual cluster on Amazon EC2.

# 6 RESULTS 2 – EXPERIMENTAL RESULTS

This chapter presents the second set of results of this dissertation. Two sets of experiments are performed using the FReMI conceptual architecture defined in Chapter 5. Section 6.1 reports the experiments performed with FReMI using a sample of snapshots from a FFR model on two HPC environments; the Atlântica cluster and the virtual cluster using EC2 instances from AWS. Section 6.2 describes the experiments, on the complete data set, performed with FReMI and W-FReDoW shared execution in a MPI cluster environment on Amazon EC2 instances only.

## 6.1 Experiment 1 – FReMI Performance on Atlântica and Amazon EC2 HPC Environment

The primary goal of this set of experiments is to find the best MPI/OpenMP implementation to execute parallel molecular docking simulations using only a small data set composed of a sample of snapshots from a FFR model. This data is called Dataset 1. It has the following attributes:

- Receptor: the first 126 snapshots from the InhA FFR model [MAC11b].
- Ligand: Triclosan (TCL400) [KUO03].

FReMI executes molecular docking simulations with the AutoDock4.2 [MOR09] package. For that, it must prepare input files for each snapshot of the FFR model. All input files were parameterized as follow:

1. Receptor preparation. A PDBQT file for each snapshot from the FFR model was generated using Kollman charges.
2. Ligand preparation 1. The TCL ligand was initially positioned in the region close to its protein binding pocket. TCL contains two rotatable bonds.
3. Ligand preparation 2. The TCL ligand was also prepared but using the coordinates of the experimental structure (PDB ID: 1P45). This is the ideal position and orientation of the ligand that is expected from docking simulations. It is called a reference ligand position.
4. Grid preparation. For each snapshot a grid parameter file (GPF) was created with box dimensions of 100 Å x 60 Å x 60 Å. The other parameters maintained the default values.
5. Docking preparation. The LGA search method and its standard parameters were selected as follow: a population size of 150 individuals, a maximum of 250,000 energy evaluations, 27,000 generations. The number of runs was set to 25 LGA runs. A docking parameter file (DPF) was generated for each snapshot from the FFR model.

In these first set of experiments the clustering of conformations created by [MAC11a] and the P-SaMI data pattern are not used. Instead, these features were simulated. FReMI uses clustering of

snapshots randomly selected (identified by “groupSnap.xml”). The P-SaMI simulation consists in setting priorities to subgroups, however, it does not discard neither changes the priorities of subgroups during the FReMI execution. This is so because there is no updating by P-SaMI in this set of experiments. Hence, to run these set of experiments, the first 126 snapshots of the FFR model were separated into 4 subgroups with 31, 32, 31 and 32 snapshots each. The priorities ranged from 1 to 3 for each subgroup.

According to equation (1) outlined in Section 5.3 the number of tasks executed per node ($T_{node}$) can be set before starting FReMI’s execution. $T_{node}$ is one of the factors that determine the size of the tasks’ queue ($Q$). To process Dataset 1, $T_{node}$ was set to either 8 or 16. Speedup and efficiency are the metrics used to investigate the performance on the HPC environments used by FReMI. Speedup is $S(n) = T1/T(n)$ and efficiency is $E(n) = S(n)/n$, where $T(n)$ is the time to run with $n$ cores. T1 is the run time for the sequential execution of the experiments (in a single core).

The experiments are composed of three distinct simulations.

- Simulation 1: FReMI execution of Dataset 1 on Atlântica and Amazon EC2 clusters. $T_{node}$ was 8 and the number of processors in each node was 8. Table 6.1 shows the scalability and performance results for this simulation.
- Simulation 2: FReMI execution on Dataset 1 on Atlântica and Amazon EC2 clusters. $T_{node}$ was 16 and the number of processors in each node was 8. Table 6.2 shows the scalability and performance results for this simulation.
- Simulation 3: FReMI execution on Dataset 1 on Atlântica cluster only. $T_{node}$ was 16 and the number of processors in each node was 16. Table 6.3 shows the scalability results for this simulation.

As shown in Table 6.1 FReMI scales well, in both HPC environments, when compared to the sequential execution. However, in most cases, the efficiency and speedup on the Atlântica cluster are better. This result can be explained by the fact that Atlântica’s machines are connected by interconnection network and they are positioned in the same physical place. Conversely, Amazon EC2 machines are connected by virtual private network connection and its virtual machines are located in multiple locations in Northern Virginia (US East Cost) [AWS12]. This may explain the decline of performance (Figure 6.1) in Amazon EC2 since FReMI requires high-throughput of message exchange. This message exchange is intrinsic to the master-slave paradigm of MPI.

Table 6.1: Scalability of Simulation 1 for Dataset 1 on Atlântica and EC2 clusters. In both clusters every node, using 8 cores in each, receives 8 tasks for parallel execution.

| | | Atlântica Cluster | | | Amazon EC2 Cluster | | |
|---|---|---|---|---|---|---|---|
| Cores | Nodes | Time (min.) | S(*n*) | E(*n*) | Time (min.) | S(*n*) | E(*n*) |
| 1 | 1 | 287.24 | 1.00 | 1.00 | 220.63 | 1.00 | 1.00 |
| 8 | 1 | 37.77 | 7.61 | 1.09 | 33.37 | 6.61 | 0.94 |
| 16 | 2 | 21.76 | 13.20 | 0.88 | 20.27 | 10.89 | 0.73 |
| 24 | 3 | 16.78 | 17.11 | 0.74 | 16.82 | 13.12 | 0.57 |
| 32 | 4 | 16.62 | 17.28 | 0.56 | 14.43 | 15.29 | 0.49 |
| 40 | 5 | 11.85 | 24.24 | 0.62 | 12.78 | 17.26 | 0.44 |
| 48 | 6 | 11.67 | 24.62 | 0.52 | 12.32 | 17.91 | 0.38 |
| 56 | 7 | 11.33 | 25.34 | 0.53 | 12.18 | 18.12 | 0.33 |
| 64 | 8 | - | - | - | 10.21 | 21.61 | 0.34 |

Simulation 2 differs from Simulation 1 only in the number of tasks executed, but its scalability (Table 6.2) and performance (Figure 6.1) are, in most cases, inferior in the Atlântica cluster. This performance loss can be understood in the following way: in Simulation 1 the nodes on Atlântica were used in the exclusive mode, i.e., despite that only 8 cores were used to execute the tasks, all 16 cores were allocated per node. On the other hand, in Simulation 2 the nodes were used in the shared mode, in which only 8 cores were allocated per node. Hence, it is possible that the remaining nodes were busy.

Table 6.2: Scalability of Simulation 2 for Dataset 1 on Atlântica and EC2 clusters. In both clusters every node, using 8 cores in each, receives 16 tasks for parallel execution.

| | | Atlântica Cluster | | | Amazon EC2 Cluster | | |
|---|---|---|---|---|---|---|---|
| Cores | Nodes | Time (min.) | S(*n*) | E(*n*) | Time (min.) | S(*n*) | E(*n*) |
| 1 | 1 | 287.24 | 1.00 | 1.00 | 220.63 | 1.00 | 1.00 |
| 8 | 1 | 38.67 | 7.43 | 1.06 | 32.93 | 6.70 | 0.96 |
| 16 | 2 | 24.74 | 11.61 | 0.77 | 18.98 | 11.62 | 0.77 |
| 24 | 3 | 17.89 | 16.06 | 0.70 | 17.66 | 12.50 | 0.54 |
| 32 | 4 | 18.17 | 15.81 | 0.51 | 12.83 | 17.19 | 0.55 |
| 40 | 5 | 13.91 | 20.66 | 0.53 | 12.17 | 18.13 | 0.46 |
| 48 | 6 | 14.94 | 19.22 | 0.41 | 8.78 | 25.14 | 0.53 |
| 56 | 7 | 10.59 | 27.13 | 0.57 | 4.52 | 48.80 | 0.89 |
| 64 | 8 | - | - | - | 4.33 | 50.92 | 0.81 |

On the Amazon EC2, especially with more than 40 cores, the gain in performance is substantial with increasing number of tasks ($T_{node}$). For example, with 48 processors executing 8 parallel tasks per machine, EC2 takes 8.78 minutes to execute Dataset 1 while Atlântica takes 14.94 minutes (Table 6.2). For this case study Amazon EC2 outperforms Atlântica (Figure 6.1). It is worth remembering that the data set used in these simulations is only a sample (126 snapshots) of the much larger, complete data set, which describes a FFR model. Therefore, the EC2 configuration bestows itself as a very attractive HPC solution for executing molecular docking simulations on the complete data set.

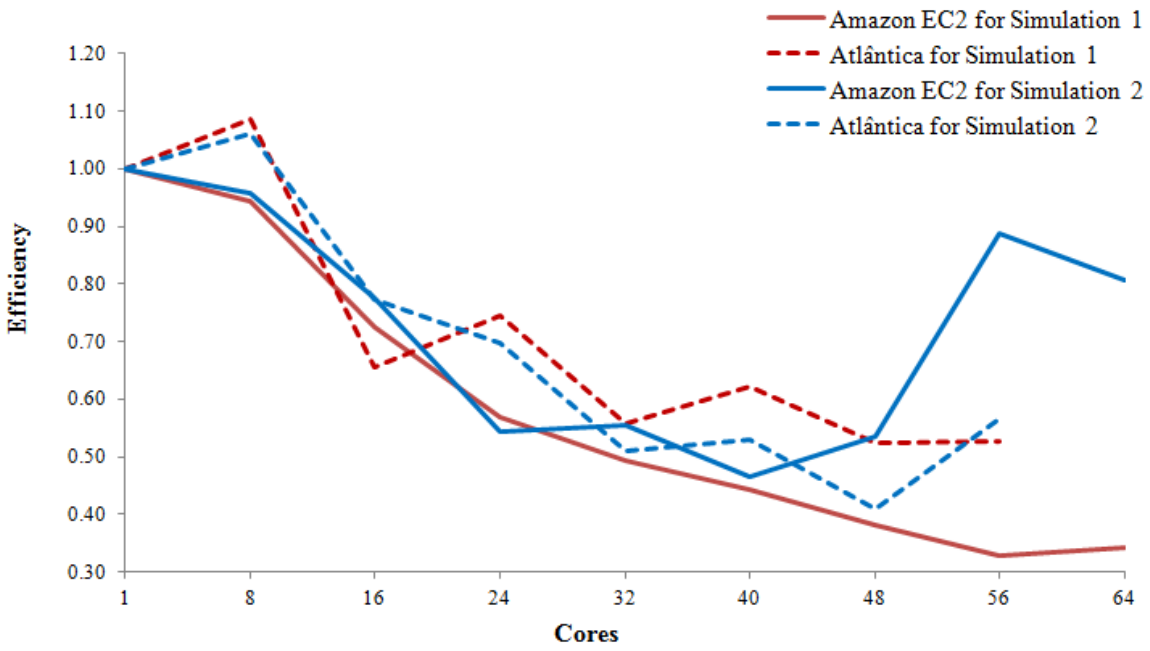


Figure 6.1: Comparative performance of the Atlântica and Amazon EC2 clusters for Simulations 1 and 2, respectively. The *x* axis is the number of MPI slave processes used; the *y* axis shows the efficiency of parallelization.

Overall, Simulation 3 shows motivating results for the total time execution. However, the speedup and the efficiency are unappealing. Specially, if compared to the others simulations.

Comparing Simulation 3 with the others, Atlântica does not scale or gain much performance when all the processors of each node are used (Table 6.3). This is surprising because it is expected that, by doubling the number of parallel tasks per node, the performance should increase. For instance, when 4 nodes were used, the efficiency obtained was 56% and 53% for the Simulations 1 and 2, respectively, and only 40% for Simulation 3. The performance reduction happens because the Intel Hyper-threading technology, which allows one core on the processor to appear like 2 cores to the operation system, was fully exploited inside the Altântica's nodes for Simulation 3. Hence, 2 tasks threads running on a single core do not have to be threads of the same process. Consequently, increasing the number of parallel tasks per node in Altântica does not provide improvements to the FReMI's performance.

Table 6.3: Scalability of Simulation 3 for Dataset 1 on Atlântica cluster. Each node receives 16 tasks that run in parallel in 16 cores.

| Cores | Nodes | Time (min) | S(*n*) | E(*n*) |
|---|---|---|---|---|
| 1 | 1 | 287.24 | 1.00 | 1.00 |
| 16 | 1 | 30.57 | 9.40 | 0.63 |
| 32 | 2 | 17.58 | 16.34 | 0.53 |
| 48 | 3 | 11.67 | 24.62 | 0.52 |
| 64 | 4 | 11.33 | 25.34 | 0.40 |
| 80 | 5 | 7.53 | 38.13 | 0.48 |
| 96 | 6 | 7.67 | 37.47 | 0.39 |

### 6.1.1 Final Considerations about Experiment 1

The main goal of this section was to find the best MPI/OpenMP implementation to execute FReMI. Three different configurations for every simulation were investigated. Simulations 1 and 2 used two HPC environments to compare their results. Simulation 3 (Table 6.3) demonstrates that gains in performance on the Atlântica cluster are obtained only if 8 tasks are executed in parallel per node. From this result, the Simulation 2 results (Table 6.2) evidenced that increasing the number of tasks per node ($T_{node}$) and using 8 cores per node is a promising approach to reduce the total execution time and enhance the performance in both HPC environments. Amazon EC2 cluster boosted the performance (4.52 minutes and 89% of efficiency) using 7 instances with 8 cores each (Figure 6.1 and Table 6.2).

It is clear from the results above that Amazon EC2 is the best choice for executing molecular docking simulations of a FFR model using FReMI. The only disadvantage is the cost of the operations on AWS. For example, Simulation 1, with only 126 snapshots, cost around US$20.00. For the complete FFR model against one ligand the cost was US$ 251.60 (See Experiment 2 below). For ten similar ligands the cost would be US$ 2,516.00. If one intends to employ this model in virtual screening of large libraries of drug-like molecules, such as the ZINC database [IRW05], with more than 23 million of molecules, the cost would be prohibitive.

In fact, the cost- effectiveness of the EC2 and S3 depends on overall usage patterns [BUY09] between the providers and consumers. To take advantage of AWS, it is necessary to control the supply and demand of cloud resources to achieve a trade-off between cost and computer power [MUR08].

For all these reasons, the best way to integrate the FReMI and W-FReDoW execution was to make use of a virtual cluster on Amazon EC2 instances. This service is able to support the storage and the internet communication protocols to keep the controlled sharing of both applications without restrictions or performance loss. As a result, an MPI environment within the Amazon EC2 instances was built (see Section 4.1.3) to perform the second set of experiments, namely Experiment 2, with the FReMI and W-FReDoW shared execution. The results obtained are described in the next section.

## 6.2 Experiment 2 – Integration of W-FReDoW with FReMI Execution on Amazon EC2 MPI Cluster

The main goal of this set of experiments is to reduce the dimensionality of the FFR model, transforming it into a RFFR model, by discarding the non-promising conformations generated by W-FReDoW. Thus, the integration of W-FReDoW with FReMI execution on Amazon EC2 MPI cluster may allow practical use of totally fully-flexible receptor models in virtual screening.

In this experiment Dataset 2 was used. It has the following attributes:

- Receptor: the first 3,100 snapshots from the InhA FFR model [MAC11b].
- Ligand: Triclosan (TCL400) [KUO03].

Unless stated otherwise, all Autodock4.2's parameters are identical to those of Dataset 1.

Dataset 2 was processed by FReMI and W-FReDoW using three different types of clustering of conformations of the FFR model [MAC11a]. Hence, there are three different XML control files (groupSnap.xml) sent by W-FReDoW to FReMI. Following the P-SaMI data pattern, for every cluster of snapshots, four docking simulations are executed. For each simulation the quality of the receptor model is evaluated only after 30%, 40%, 50%, and 70 % of the snapshots, in each subgroup of the clusters, have been docked. Then, Experiment 2 involves a total of 12 simulations.

Before starting Experiment 2 a study to validate the proper scalability to obtain a satisfactory performance under FReMI and W-FReDoW shared execution was conducted. Only FReMI with Dataset 2 was executed on Amazon EC2 MPI cluster using 56 cores (this number of processors gave the best scalability in Experiment 1). In this test $T_{node}$ = 32. This set up surprisingly resulted in only 28% efficiency against the 89% efficiency found in Experiment 1. Other scalabilities, ranging from 32 to 64 cores, were tested. Then the best efficiency, of 41 %, was found for 40 cores.

FReMI took 5 hours and 40 minutes to execute Dataset 2 on Amazon EC2. Hence, based on these analyses, the best configuration to run the Dataset 2 was:

- 40 cores = 5 c1.xlarge EC2 Amazon instances with 8 cores each and;
- 32 tasks per instance ($T_{node}$ = 32, $Q$ = 160).

Table 6.4 summarizes the results of Experiment 2. According to Hübler [HUB10] the processing of the smallest subgroups of snapshots produces the maximum performance gain because the analysis is performed with minimum amount of snapshots. In addition, she claims in her conclusions that the earlier the analyses start, the larger is the quantity of unpromising snapshots that should be recognized and discarded. The results of this experiment corroborates Hubler's hypothesis (see boldface numbers in column 5).

The total execution time spent in the serial molecular docking simulations using Dataset 2 was around 4 days. As a mentioned above, the FReMI only execution of Dataset 2, using the same EC2 configuration of Experiment 2, reduced the execution time to 5 hours and 40 minutes. When FReMI and W-FReDoW are integrated in a shared execution, the total execution time was further reduced by 30%, i.e., down to 3 hours and 55 minutes for all subgroups that began the analysis with 30% of the processed snapshots (Table 6.4). Overall the execution time is reduced about 10-30% (Figure 6.2).

Table 6.4: Results of the simulations executed in Experiment 2 using three different types of clustering of conformations of the FFR model [MAC11a]. Column 1 identifies the three different types of clustering. Column 2 specifies the percentage of processed snapshots after which P-SaMI analysis [HUB10] of the model quality starts. Column 3 displays the total execution time for each simulation. Columns 4 and 5 display the amount of snapshots docked and discarded, respectively.

| Clustering | Start Analysis of Model Quality (%) | Total Time | Docked Snapshots | Discarded Snapshots |
|---|---|---|---|---|
| **01** | **30.00** | **03:59:30** | **2,249** | **851** |
| | 40.00 | 04:13:00 | 2,407 | 693 |
| | 50.00 | 04:28:00 | 2,495 | 605 |
| | 70.00 | 04:59:40 | 2,891 | 281 |
| **02** | **30.00** | **03:54.30** | **2,210** | **890** |
| | 40.00 | 04:16:00 | 2,423 | 677 |
| | 50.00 | 04:40:00 | 2,512 | 586 |
| | 70.00 | 04:57:40 | 2,868 | 232 |
| **03** | **30.00** | **03:56:30** | **2,264** | **836** |
| | 40.00 | 04:13:00 | 2,377 | 723 |
| | 50.00 | 04:38:00 | 2,537 | 563 |
| | 70.00 | 04:58:40 | 2,818 | 282 |

Column 4 contains more than the docked snapshots in each row. The docked snapshots are actually examples of a RFFR model obtained from the integrated FReMI and W-FReDoW execution of a FFR model in an HPC environment.

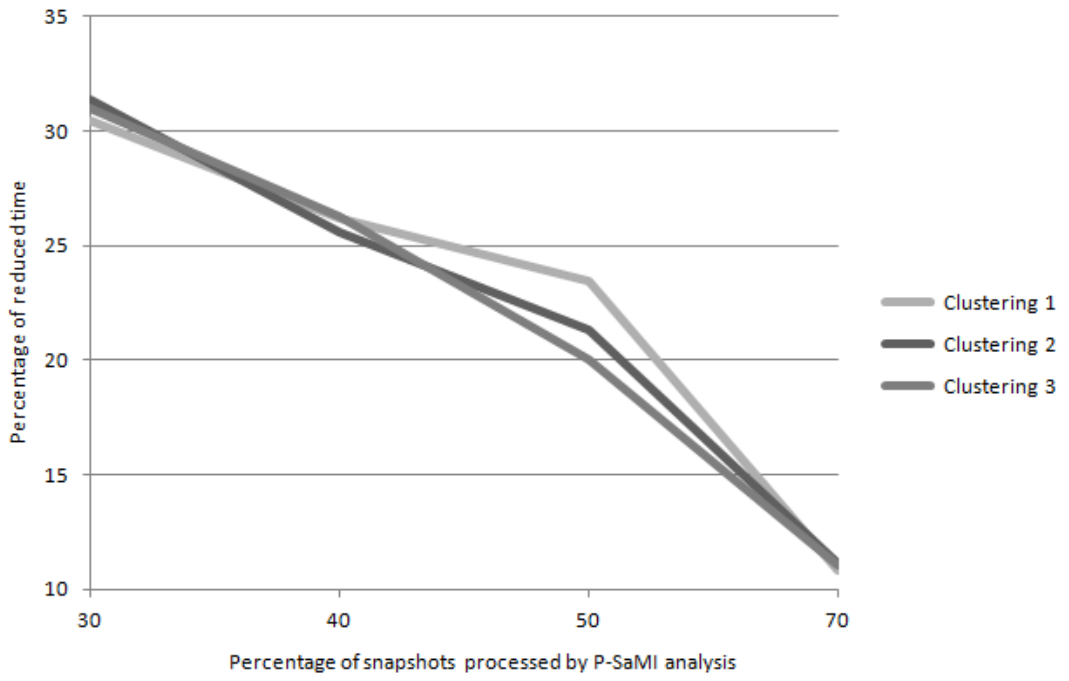


Figure 6.2: Performance gain versus P-SaMI analysis using three clustering of snapshots on FReMI and W-FReDoW shared execution. Percentage of reduced time is calculated from the FReMI only execution. Percentage of snapshots processed by P-SaMI analysis is the amount of docking results that P-SaMI utilizes to start the analysis on the subgroup of snapshots in each cluster.

Finally, Experiment 2 with W-FReDoW reduced the total execution time to between 10-30% of that of FReMI´s only execution, which, in turn, decreased near 94 % with respect to the serial execution time. Clearly, the integration of the FReMI middleware, based on the MTC paradigm, with the W-FReDoW webApp is a promising strategy to tackle the problem of molecular docking simulation that employs FFR models. It reduces the dimension of such models allowing the reduction of the overall execution time from months to days and possibly to hours. This is especially valuable for routine virtual screening of libraries with over millions of ligands.

### 6.2.1 Discussion of Experiment 2

The most significant advantage of shared resources is the guaranteed access time of the resources wherever you are and whenever you need. There is no competition or restrictions for access to the machines. However, it is necessary to pay for as many computing nodes as needed, which are charged at an hourly rate. The rate is calculated for what and when the resources are being used, e.g. if you do not need compute time, you do not need to pay [HAZ08].

The only drawback is that the hard disks associated with EC2 instances, called EBS by Amazon, do not have an existence beyond the instance's life-time. This means that the user data must be copied back and forth between S3 and EC2. Although this take a little work to get going, it is easy to do with open source tools. Also, the fast interconnect between S3 and EC2 allows efficient file transfer.

The only practical problem experienced during the integrated execution of FReMI and W-FReDoW was with the internet connection, which sometimes became very unstable. However, this is not an issue since W-FReDoW is capable of restarting an execution from where it stopped. For instance, during the Experiment 2 simulations the connection dropped only twice.

The total bill for the work done on S3 and EC2 to conclude the Experiment 2 results was approximately US$ 251.60 calculated from 370 hours of use of 5 c1.xlarge instances, i.e. 74 hours for each instance. No charge is applied to the S3 services since the total use was within the monthly global free tier. The cost and time taken to learn about cloud computing and run the first test simulations are not being computed.

# 7. RELATED WORKS

There are some methods in the literature that propose to perform virtual screening of small molecules using molecular docking on dedicated HPC clusters. However, three features distinguish FReMI from the other works, namely:

1. The utilization of a FFR model rather than a rigid receptor model.
2. The methodology applied to achieve the RFFR model with the FReMI and W-FReDoW shared execution.
3. The creation of a virtual cluster environment to execute FReMI on the Amazon Cloud Computing using the hybrid MPI-OpenMP programming model.

In addition, FReMI includes more than one computational technique that works in cooperation to achieve the RFFR model, such as HPC environments, cloud computing, internet communication protocol and also the methodology employed to handle data sets before and during molecular docking simulations in parallel (see Chapter 5). For these reasons, the middleware presented in this study has few similarities with the current state of the art.

FReDoWS [MAC10a] is the only tool found in the literature that executes molecular docking simulations including the explicit receptor flexibility and its snapshots generated by MD simulations trajectories of the receptor. It automates molecular docking simulations of a FFR model using AutoDock3.0.5 and a scientific workflow. Additionally, FReDoWS intends to accelerate virtual screening of ligands with a snapshot selection function which reduces the dimension of FFR models.

There are a number of software that performs virtual screening of small molecules against rigid receptors on local HPC environments using AutoDock4.2. Most of them use the number of ligands to distribute the tasks along the processors. For instance, DOVIS 2.0 [JIA08] uses a dedicated HPC Linux cluster to execute virtual screening where ligands are uniformly distributed on each CPU. The pre-docking steps required for docking with AutoDock4.2 are performed in Linux systems by a Graphical User Interface (GUI). VSDocker 2.0 [PRA10] and Mola [ABR10] are other examples of such systems. The main difference between them and DOVIS 2.0 is that VSDocker 2.0 works on multiprocessor computing clusters and multiprocessor workstations operated by a Windows HPC Server; it also has a console application for Microsoft Windows platforms. Mola uses AutoDock4.2 [MOR09] and AutoDock Vina to execute the virtual screening of small molecules on non-dedicated compute clusters.

Another approach used to enhance the performance of docking simulations of rigid receptors is the reduction of network I/O traffic files during the loading of grid maps at the beginning of each docking simulation. This methodology was used in DOVIS 2.0 [JIA08], Autodock4.lga.MPI [COL11] and mpAD4 [NOR11]. They observed one potential decrease in I/O traffic files when keeping the grid maps in memory, i.e., for each node of the computer cluster only one grid map is loaded by the master

server. Autodock4.lag.MPI only uses the MPI library to distribute the jobs, while mpAD4 use a multilevel parallelization where MPI is used to distribute the jobs across the nodes and the OpenMP parallelization to execute the LGA inside of each node. Nevertheless, this approach can not to be used in FReMI, due to the explicit receptor flexibility, in which, for each snapshot, a new atom coordinate is assumed. It means that for each snapshot from FFR models *autogrid4* and *autodock4* must be executed.

Hydra [COU10] is a middleware which provides the linking between Scientific Workflow Management Systems (SWfMS) and the HPC environment for distributing tasks. It explores the data parallelism on three fragmentation scenarios to find regions of similarity between biological sequences using the BLAST software. Hydra holds a set of components to be included on the workflow specification of any SWfMS to control parallelism of activities following the MTC paradigm [COU10]. Using Hydra, the MTC parallelism strategy can be registered, reused, and provenance may be uniformly gathered during execution of workflows. MTC is the concept borrowed by FReMI to distribute the tasks along the HPC processors. Finally, while Hydra was built to perform large-scale sequence comparison, FReMI is focused on treating the problem of molecular docking simulation of FFR models.

# 8. FINAL CONSIDERATIONS

*In silico* molecular docking simulations is an integral part current drug design efforts. The amount of these docking executions has been increased due to the several parameters and input data that have been used with the purpose of mimic the natural behaviour of ligands and receptors, especially those considering the explicit plasticity and flexibility of FFR models and flexible ligands [MAC10b]. Furthermore, these simulations are computer intensive and their sequential execution is an unfeasible task. For this reason, this dissertation presented a study on the use of parallel programming techniques, focussing on MPI cluster environments, applied to the optimization of CPU time of molecular docking simulations of FFR models. Based on this research, "FReMI: a middleware to handle many tasks of FFR models in HPC environments" has been built and tested. The results are very encouraging.

Chapter 2 described the theoretical principles for the understanding the problem at study. It provided explanations of molecular docking as one step of RDD, focussing on receptors of flexibility types and the approach used in this dissertation to consider its plasticity and elasticity. Chapter 3 and 4 show the materials and methods used for the development this study. Chapter 3 presented a revision about parallel programming, HPC environments and cloud computing on Amazon EC2. It purposes is to provide a better comprehension of the resources used and methodology explained in Chapter4 and used to build FReMI.

Chapter 5 and 6 present the results obtained during the development of this dissertation. The conceptual architecture, showed in Chapter 5, was created to enrichment the understanding between FReMI and its bridges the gap with the others applications. From FReMI conceptual architecture implementation the experimental results were generated and presented on Chapter 6. At the beginning a sample of snapshots from FFR model in study, called Dataset 1, was executed on FReMI and both HPC environments; Atlântica cluster and Amazon EC2 instances. The performance results show few differences between them; however, by means of an evaluation at the end of this section was identified several benefit using Amazon EC2 instead, Atlântica cluster. Based on this conclusion, Chapter 6 presents the FReMI and W-FReDoW shared execution with a MPI cluster on Amazon EC2. This chapter showed that using the "pay-as-you-go" AWS facilities [MUR08] the premium to complete various experiments in shorter elapsed time is a good option when time becomes more limit than money. Therefore, the foremost contribution on chapter 6 was the time reduction in the molecular docking simulations execution FReMI and W-FReDoW with the Dataset 2 (FFR model with 3,100 snapshots) on Amazon EC2 MPI cluster. The results show that the overall execution time reduced from days to hours with the FReMI execution, after than with the FReMI and W-FReDoW mutual execution it was achieve, from the FReMI total execution time, further 30% performance gain.

Chapter 7 showed the state of art associated with this study; highlighting the key contributions this dissertation compared with the currently scientific community works.

## 8.1 Main Contributions

The main contributions of this Dissertation to the addressed problems are:

- The heuristic function developed to handle the tasks executed in HPC environments. This function sorts the placing of a proportional fraction of snapshots in each queue of tasks according to the priorities of their subgroups during the execution time.
- The parallel algorithm developed to distribute tasks in large scale, using the master-slave programming paradigm and improve the scalability among the nodes with MPI library. The hybrid MPI-OpenMP programming model to obtain the high level of parallelism outside and inside of every node used for the HPC environments.
- The FReMI conceptual architecture proposed to identify the FReMI functions and its interoperability with the others application servers. It was built with a set of components which represents the different stages with their activities and several arrows that supplies the operations direction inside and outside FReMI. Furthermore, the conceptual architecture extends its components to represent the main functions of the W-FReDoW that support a cooperation execution with FReMI and the arrows that evidence the communication internet protocols employed between these both applications.
- The total time reduction shown in the experimental results. FReMI could scale to a large number of cores in order to process large amounts of computational that would otherwise not have been possible by a compute inside of laboratory. Reducing the molecular docking simulations run-time of the FFR model from 4 days to 5 hours and 40 minutes is an attractive solution. Especially, because it allows the use of news trajectories of FFR models even greater and the exploration of large libraries of ligands than would otherwise not have been possible.
- The reduction of the FFR model dimensionality to achieve the RFFR model. FReMI and W-FReDoW shared execution can perform the gradual elimination of unpromising conformations to assist in the high performance of massively parallel execution of docking simulations of flexible receptors. Where FReMI will not use all the conformations that make up a FFR model, but instead, only those which are significantly more promising.

## 8.2 Future Works

FReMI was tested with only a ligand and a FFR model containing 3,100 conformations generated by a MD simulation. MD simulations are now running on the tens to hundreds of nanoseconds. This could produce FFR models with up to 200.000 snapshots! FReMI should be tested

with such models. Additionally, it would be interesting to make use of others ligands by means of exploration of virtual libraries of compounds such as ZINC [IRW05].

More research is needed to improve FReMI's performance even further on MPI EC2 instances. It was observed that, for Dataset 2 execution in 48 processors, there was a significant raise in total execution time possible due to some core becoming idle. A future study should concentrate on investigating the hybrid MPI-OpenMP programming model and the master-slave paradigm in order to limit or eliminate this idleness. This would make possible future FReMI experiments with larger FFR models with different ligands.

Finally, FReMI and W-FReDoW communication protocols still need improvement. A significant delay in updating subgroups' status was observed when FReMI received the updating files. While some snapshots were still being processed by FReMI, W-FReDoW had already discarded them.

# REFERENCES


[ABR10] Abreu, R. M. V.; Froufe, H. J. C.; Queiroz, M. J. R. P.; Ferreira, I. C.F.R. "MOLA: a bootable, self-configuring system for virtual screening using AutoDock4/Vina on computer clusters". Journal of Cheminformatics, vol. 2, 2010, pp. 3-6.

[AEF11] Elasticfox Getting Started Guide. "Amazon Elastic compute Cloud Elasticfox: Getting started guide". Available at http://aws.amazon.com/articles/1797. Accessed at December 2011.

[ASF11] S3Fox Organizer. "Firefox Organizer for Amazon S3 and Amazon CloudFront (S3Fox)". Available at http://aws.amazon.com/developertools/771.

[ALO06] Alonso, H.; Bliznyuk, A. A.; Gready, J. E. "Combining docking and molecular dynamic simulations in drug design". Medicinal Research Reviews, vol. 26, 2006, pp. 531-568.

[AWS12] Amazon Web Services. "A suite of web services made available by Amazon. Amazon.com company". http:// aws.amazon.com. Accessed at January 2012.

[ARM10] Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A. D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; Zaharia, M. "A View of Cloud Computing". Communications of the ACM, vol. 53, 2010, pp. 50-58.

[BAR03] Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield, A. "Xen and the art of virtualization". Proceedings of the 19th ACM Symposium on Operating System Principles, vol. 37, 2003, pp. 164-177.

[BER00] Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. "The Protein Data Bank". Nucleic Acids Research, vol. 28, 2000, pp. 235-242.

[BUY09] Buyya, R.; Yeo, C. S.; Venugopal, S.; Broberg, J.; Bradic, I. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility". Future Generation Computer Systems, vol. 25, 2009, pp. 599-616.

[BUY99] Buyya, R. "High Performance Cluster Computing: Architectures and Systems, Volume 1". Australia : Prentice Hall, 1999, 881p.

[CLA01] Clauβen, H.; Christian, B.; Rarey, M.; Lengauer, T. "FlexE: Efficient Molecular Docking Considering Protein Structure Variations". Journal of Molecular Biology, vol. 308, 2001, pp. 377-395.

[COH11] Cohen, E. M. L.; Machado, K. S.; Cohen, M.; Norberto de Souza, O. "Effect of the explicit flexibility of the InhA enzyme from Mycobacterium tuberculosis in molecular docking simulations". BMC Genomics, vol. 12, 2011, pp. S7.

[COL11] Collignon, B.; Schulz, R.; Smith, J. C.; Baudry, J. "Task-Parallel Message Passing Interface Implementation of Autdock4 for Docking of Very Large Databases of Compounds Using High-Performance Super-Computers". Journal of Computational Chemistry, vol. 32, 2011, pp. 1202-1209.

[COS11] Costa, A. L. P.; Pauli, I.; Dorn, M.; Schroeder, E. K.; Zhan, C. –G; Norberto de Souza, O. "Conformational changes in 2-trans-enoyl-ACP (CoA) Reductase (InhA) from M. tuberculosis induced by na inorganic complex: a molecular dynamics simulation study". Journal of Molecular Modeling, vol. 17, 2011, PP. 10-23.

[COZ08] Cozzini, P; Kellogg G. E.; Spyrakis, F.; Abraham, D. J.; Costantino, G.; Emerson, A.; FAnelli, F.; Gohlke, H.; Kuhn, L. A.; Morriz, G. M.; Orozco, M.; Pertinhez, T. A.; Rizzi, M.; Sotriffer, C. A. "Target flexibility: Na emerging consideration in drug discovery and design". Journal of Medicinal Chemistry, vol. 51, 2008, pp. 6237-6255.

[COU10] Coutinho, F.; Ogasawara, E.; de Oliveira, D.; Braganholo, V.; Lima, A. A. B.; Dávila, A. M. R.; Mattoso, M. "Data Parallelism in Bioinformatics Workflows Using Hydra". In: 19th ACM International Symposium on High Performance Distributed Computing, pp. 507-515, 2010.

[CUR11] Libcurl Tutorial. "Libcurl programming tutorial". Accessible at http://curl.haxx.se/libcurl/c/libcurl-tutorial.html. Accessed at July 2011.

[DEP11] De Paris, R.; Frantz, F. A.; Norberto de Souza, O.; Ruiz, D. D. A. "A Conceptual Many Tasks Computing Architecture to Execute Molecular Docking Simulations of a Fully-Flexible Receptor Model". Lecture Notes in Computer Science, vol. 6832, 2011, pp. 75-78.

[DES95] Dessen, A.; Quemard, A.; Blanchard, J. S.; Jacobs, W. J.; Sacchettini, J. C. "Crystal Structure and Function of the Isoniazid Target of Mycobacterium tuberculosis". Science, vol. 267, 1995, pp. 1638-1641.

[FIE00] Fielding, R. T. "Architectural Styles and the Design of Network-based Software Architectures". Doctoral Dissertation, University of California, Irvine, California, 2000, 162p. Available at http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. Accessed at December 2011.

[FLY72] Flynn, M. "Some computer organizations and their effectiveness". IEEE Transactions on Computer, vol. 21, 1972, pp. 948-960

[FOS05] Foster, I. "Globus Toolkit Version 4: Software for Service-Oriented Systems". Lecture Notes in Computer Science, v. 3779, 2005, pp.2-13.

[GOO96] Goodsell, D.; Morris, G.; Olson, A. "Automated Docking of Flexible Ligands: Applications of AutoDock". Journal of Molecular Recognition, vol. 9, 1196, pp. 1-5.

[HAZ08] Hazelhurst, S. "Scientific computing using high-performance computing: a case study using the Amazon elastic computing cloud". In: ACM Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology, v. 338, 2008, pp. 94-103.

[HUA07] Huang, S. Y.; Zou, X. "Ensemble docking of multiple protein structures: considering protein structural variations in molecular docking". Proteins, vol. 66, 2007, pp. 339-421.

[HUB10] Hübler, P. N. "P-MIA: Padrão Múltiplas Instâncias Autoadaptáveis - Um Padrão de Dados para Workflows Científicos". Tese de Doutorado, Programa de Pós-graduação em Ciências da Computação, PUCRS, Porto Alegre, RS, Brasil, 2010, 179p.

[JIA08] Jiang, X.; Kumar, K.; Hu, X.; Wallqvist, A.; Reifman, J. "DOVIS 2.0: an efficient and easy to use parallel virtual screening tool based on AutoDock 4.0". Chemistry Central Journal, vol. 2, 2008, pp. 12-18.

[JON97] Jones, G.; Willett, P.; Glen, R. C.; Leach, A. R.; Taylor, R. "Development and Validation of a Genetic Algorithm for Flexible Docking". Journal of Molecular Biology, vol. 267, 1997, pp. 727-748.

[IRW05] Irwin, J. J.; Shoichet, B. K. "Zinc-a free database of commercially available compounds for virtual screening". Journal of Chemical. Information and Modeling, vol. 45, 2005, pp. 177-182.

[KAP08] Kapetanovic, I.M. "Computer-aided drug discovery and development (CADDD): *In silico*-chemico-biological approach". Chemico-biological Interactions, vol. 17, 2008, pp. 165-176.

[KUN92] Kuntz, I. "Structure-based strategies for drug design and discovery". Science, vol. 257, 1992, pp. 1078-1082.

[KUO03] Kuo, M. R.; Morbidoni, H. R.; Alland, D.; Sneddon, S. F.; Gourlie, B. B.; Staveski, M. M.; Leonard, M.; Gregory, J. S.; Janjigian, A. D.; Yee, C.; Musser, J. M.; Kreiswirth, B.; Iwamoto, H.; Perozzo, R.; Jacobs, W. R.; Sacchettini, J. C.; Fidock, D. A. "Targeting tuberculosis and malaria through Inhibition of Enoyl reductase: compound activity and structural data". Journal of Biological Chemistry, vol. 278, 2003, pp. 20851-20859.

[LAN09] Lang, P. T.; Brozell, S. R.; Mukherjee, S.; Pettersen, E. F.; Meng, E. C.; Thomas, V.; Rizzo, R. C.; Case, D. A.; James, T. L.; Kuntz, I. D. "DOCK 6: Combining techniques to model RNA-small molecule complexes". RNA Journal, vol. 15, 2009, pp. 1219-1230.

[LUS01] Luscombe, N. M; Greenbaum, D.; Gerstein, M. "What is Bioinformatics? A Proposed Definition and Overview of the Field". Methods of Information in Medicine, vol. 4, 2001, pp. 346-358.

[MAC10a] Machado, K. S.; Winck, A. T.; Ruiz, D. D.; Norberto de Souza, O. "Discretization of flexible-receptor docking data". LNBI-LNCS Advances in Bioinformatics and Computational Biology, vol. 6268, 2010, pp. 75-79.

[MAC10b] Machado, K. S.; Winck, A. T.; Ruiz, D. D. A.; Norberto de Souza, O. "Mining flexible-receptor docking experiments to select promising protein receptor snapshots". BMC Genomics, vol. 11, 2010, pp. 1-10.

[MAC11a] Machado, K. S. "Efficient Selection of Flexible Receptor Snapshots Applied in Molecular Docking Simulations". Tese de Doutorado, Programa de Pós-graduação em Ciências da Computação, PUCRS, Porto Alegre, RS, Brasil, 2011, 180p.

[MAC11b] Machado, K. S.; Schroeder, E.; Ruiz, D. D.; Cohen, E. M. L.; Norberto de Souza, O. "FReDoWS: a method to automate molecular docking simulations with explicit receptor flexibility and snapshots selection". BMC Genomics, vol. 12, 2011, pp. 2-13.

[MAC11c] Machado, K. S.; Wick, A. T.; Ruiz, D. D.; Norberto de Souza, O. "Mining flexible-receptor molecular docking data". WIREs Data Mining and Knowledge Discovery, vol. 1, 2011, pp. 532-541.

[MAN09] Mandal, S.; Moudgil, M.; Mandal, S. K. "Rational drug design". European Journal of Pharmacology, vol. 625, 2009, pp. 90-100.

[MOR98] Morris, G. M.; Goodsell, D. S.; Halliday, R. S.; Huey, R.; Hart, W. E.; Belew, R. K.; Olson, A. J. "Automated docking using a Lamarckian genetic algorithm and empirical binding free energy function". Journal of Computational Chemistry, vol. 19, 1998, pp. 1639-1662.

[MOR09] Morris, G. M.; Huey, R.; Lindstrom, W.; Sanner, M. F.; Belew, R. K.; Goodsell, D. S.; Olson, A. J. "AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility". Journal of Computational Chemistry, vol. 30, 2009, pp. 2785-2791.

[MOR10] Morris, G. M.; Goodsel, D. S.; Pique, M. E.; Lindstrom, W. L.; Huey, R.; Forli, S.; Hart, W. E.; Halliday, S.; Belew, R.; Olson, A. J. "AutoDock User's Guide – AutoDock: User Guide: Automated Docking of Flexible Ligands to Flexible Receptors. Version 4.2". Department of Molecular Biology, the Scripps Research Institute, La Jolla, USA, 2010, 49p.

[MPI09] Message Passing Interface Forum. "MPI: A Message-Passing Interface Standard – Version 2.2". University of Tennessee, 2009. Accessible at http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf. Accessed at December 2011.

[MUR08] Murty, J. "Programming Amazon Web Services: S3, EC2, SQS, FPQ, and Simple DB". United States of America : O'Reilly, 2008, 581p.

[NOR11] Norgan, A. P.; Coffman, P. K.; Kocher, J. A.; Katzmann, K. J.; Sosa, C. P. "Multilevel Parallelization of AutoDock 4.2". Journal of Cheminformatics, vol. 3, 2011, pp. 1-7.

[OLI04] Oliveira, J.; Souza, E.; Basso, L; Palaci, M.; Dietze, R.; Santos, D.; Moreira, I. "Na inorganic iron complex that inhibits wild-type and na isoniazid-resistant mutant 2-trans-enoyl-ACP (CoA) reductase from Mycobacterium tuberculosis". Chemical Communications, vol. 3, 2004, pp. 312-313.

[OMP11] OpenMP Specifications. "OpenMP Application Program Interface – Version 3.1". OpenMP Architecture Review Board, 2011. Accessible at http://www.openmp.org/mp-documents/OpenMP3.1.pdf. Accessed at December 2011.

[PLE11] Plewczynski, D.; Lazniewski, M.; Augustyniak, R.; Ginalski, K. "Can We Trust Docking Results? Evaluation of Seven Commonly Used Programs on PDBbind Database". Journal of Computational Chemistry, vol. 32, 2011, pp. 742-755.

[POU05] Pouwelse, J.; Garbacki, P.; Epema, D.; Sips, H. "The Bittorent P2P File-Sharing System: Measurements and Analysis". Lecture Notes in Computer Science, vol. 3640, 2005, pp. 205-216.

[PRA10] Prakhov, N. D.; Chernorudskiy, A. L.; Gainullin, M. R. "VSDocker: a tool for parallel high-throughput virtual screening using AutoDock on Windows-based computer clusters". Bioinformatics, vol. 26, 2010, pp. 1374-1375.

[RAB09] Rabenseifner, R.; Hager, G.; Jost, G. "Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes". In: 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, 2009, pp.427-736.

[RAI10] Raicu, I.; Foster, I.; Wilde, M.; Zhang, Z.; Iskra, K.; Beckman, P.; Zhao, Y.; Szalay, A.; Choudhary, A.; Little, P.; Moretti, C.; Chaudhary, A.; Thain, D. "Middleware support for many-task computing". Cluster Computing, vol. 13, 2010, pp. 291-314.

[SCH05] Schroeder, E.; Basso, L.; Santos, D.; Norberto de Souza, O. "Molecular dynamics simulation studies of the wild-type, I21V, and I16T mutants of isoniazid-resistant Mycobacterium tuberculosis enoyl reductase (InhA) in complex with NADH: toward the understanding of NADH-InhA different affinities". Biophys Journal, vol. 89, 2005, pp. 876-884.

[SMI01] Smith, L.; Bull, M. "Development of mixed mode MPI / OpenMP applications". Journal Scientifica Programming, vol. 9, 2001, pp. 83-98.

[STA10] Stallman, R. M. "Using the GNU Compiler Collection – For GCC version 4.6.2". GCC Developer Community, 2010. Accessible at http://gcc.gnu.org/onlinedocs/gcc-4.6.2/gcc.pdf. Accessed at December 2011.

[STO93] Stoddard, B.; Koshland, D. "Molecular recognition analyzed by docking simulations: the aspartate receptor and isocitrate dehydrogenase from Escherichia coli". Proceedings of the National Academy of Sciences of the United States of America, vol. 90, 1993, pp. 1146–1153.

[TAN02] Tanenbaum, A. S.; van Steen, M. "Distributed Systems: Principles and Paradigms". United States of America : Prentice Hall, 2002, 803p.

[TEO03] Teodoro, M. L.; Kavraki, L. E. "Conformational Flexibility Models for the Receptor in Structure Based Drug Design". Current Pharmaceutical Design, vol. 9, 2003, pp. 1419-1431.

[TOT08] Totrov, M.; Abagyan, R. "Flexible ligand docking to multiple receptor conformations: a practical alternative". Current Opinion in Structural Biology, vol. 18, 2008, pp. 178-184.

[TRE00] Trelles, O. "On the parallelization of bioinformatics applications". Briefings in Bioinformatics, vol. 2, 2000, pp. 181-194.

[WAN07] Wang, F.; Langley, R.; Gulten, G.; Dover, L. G.; Besra, G. S.; Jacobs Jr., W. R.; Sacchettini, J. C. "Mechanism of thioamide drug action against tuberculosis and

leprosy". The Journal of Experimental Medicine, vol. 204, 2007, pp. 73-78.

[WEI04] Wei, B.; Weaver, L.; Ferrari, A.; Matthews, B.; Shoichet, B. "Testing a flexible-receptor docking algorithm in a model binding site". Journal Molecular Biological, vol. 337, 2004, pp. 1161-1182.

[WIN09] Winck, A. T; Machado, K. S.; Norberto de Souza, O.; Ruiz, D.D. "FReDD: Supporting mining strategies through a flexible-receptor docking database". LNBI-LNCS Advances in Bioinformatics and Computational Biology, vol. 5676, 2009, pp. 143-146.

[WON08] Wong, C. F. "Flexible ligand-flexible protein docking in protein kinase systems. Biochi". Biochimica et Biophysica Acta , vol. 1784, 2007, pp. 244-251.

[XML03] The XML C parser and toolkit of Gnome. Message Passing Interface Forum. "Libxml Tutorial". Open Source CMS Services, 2003. Accessible at http://xmlsoft.org/index.html. Accessed at January 2012.