Títol: Simulació i modelat d'entorns cloud per a experimentació amb machine learning

Volum: 1

Alumne: Jordi Roldán Llinàs

Director/Ponent: Josep Lluís Berral García / Ricard Gavaldà Mestre

Departament: Llenguatges i Sistemes Informàtics

Data: 29/06/10

DADES DEL PROJECTE

Títol del Projecte: **Simulació i modelat d'entorns cloud per a experimentació amb machine learning**

Nom de l'estudiant: Jordi Roldán Llinàs
Titulació: Enginyeria en Informàtica
Crèdits: 37,5
Director/Ponent: Josep Lluís Berral García / Ricard Gavaldà Mestre
Departament: Llenguatges i Sistemes Informàtics

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: Jorge Castro Rabal

Vocal: Gemma Sese Castel

Secretari: Ricard Gavaldà Mestre

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data:

Abstract

Nowadays Cloud computing has emerged as one of the most promising computer paradigms. The idea of selling software as a service has promoted IT enterprises to bet for this new paradigm. Cloud computing aims to power the next generation data centers not only offering software as a service but virtual services like hardware, data storage capacity or application logic. The increasing use of Cloud-based applications will also increase the power dedicated to the data centers that support this Clouds.

Research in Cloud computing requires solutions that have to be tested in real environments. It is difficult and expensive to set up suitable test-beds for large scale cluster applications. Simulation can fulfill the needs that we find in Cloud computing experimentation. A large data center simulator can save lots of time and effort in Cloud investigation.

This project presents the design and development process of some extensions to an existing virtualized data center simulator for Cloud computing research. It is able to reproduce the behaviour of a real Cloud framework and the information that it offers of the execution makes it suitable for testing and investigation purposes.

The final idea of this project was to extend the heterogeneity of the tests that can be run by the simulator to use it as a test-bed for machine learning experimentation.

Acknowledgements

I would like to express my gratitude to all the people who has made this project possible. I would like to give my sincerely thanks to my advisors Dr. Jordi Torres and Dr. Ricard Gavaldà and specially to my director Josep Lluís Berral García for their time, effort and advices. It has been a great experience I will not forget all the lessons I learned with you.

Many thanks to Iñigo Goiri and Ferran Julià, this project would not have been possible without your help and unconditional support. Thank you for always being there.

Finally, I would like to devote this work to all of my family and friends. Specially to my parents for their unconditional support and patience. Also for the moon which shared with me long nights. You really own a large part of this project.

Contents

List of Figures

List of Tables

1. Introduction

This project is the result of one year investigating and working with a team compound by some teachers and students of post-graduate courses. As it is an extension of some work developed by other students it is important to understand the origin and background that precedes this work.

In this section we describe the origin and background of the project and the facts that motivate the conception and purpose of this work. In the next sections we will survey the work that precedes that project in order to specify our starting point. The investments of some research groups of the university became a simulator which is the essence of that project. The simulator that they developed is our starting point, and all the work we have done is extending its functionalities.

1.1 Basic knowledge for understanding this project

There are some important concepts to understand before reading this document. In this section we will explain some topics that are related with some specific disciplines of computer science.

1.1.1 About Machine Learning and Data Mining

Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to learn patterns and behaviors based on empirical data. Machine learning techniques consist, generally, in collecting a training data set from the system with data composed by labeled example instances, and creating a model through some technique (as induction, deduction, reinforcement,..) able to explain these examples, expecting that new data will fit on it. Machine Learning techniques are divided into supervised learning (like classification and regression), unsupervised learning (discovering the relationship between the input data), clustering (finding clusters of examples), reinforcement learning (selecting the best decision from the past experiences feedback).

A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data. That is the reason why it is really related with Data mining, which is the process of extracting patterns from data.

1.1.2 About Grid and Cloud computing

The grid computing refers to the act of sharing tasks over multiple computers. Grid computing (or the use of a computational grid) is the union of the resources of many computers in a network, usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data.

Clouds are vast resource pools with on-demand resource allocation. Cloud computing overlaps some of the concepts of distributed, grid and utility computing, however it does have its own meaning. In fact, they are both used to economize computing by maximizing existing resources.

Cloud computing is accessing resources and services needed to perform functions with dynamically changing needs.

However, the difference between the two lies in the way the tasks are computed in each respective environment. In a grid, one large job is divided (if possible) into many small portions and executed on multiple machines. This characteristic is fundamental to a grid; not so in a cloud. While grid computing also offers a similar facility for computing power, cloud computing is not restricted to just that. A cloud can offer many different services, from web hosting, right down to word processing. In fact, a computing cloud can combine services to present a user with a homogeneous optimized result.

1.1.3 About Virtualization and Virtual Machines (VM's)

One of the key technologies in the cloud systems is virtualization, which is a technology that has enabled cost reduction and easier resource management for this type of distributed systems. Virtualization provides isolation between different applications that are sharing the same physical resources. That was the context in which the term Virtual Machine (VM) was established. Virtualization provide the sharing of the underlying physical machine resources using different virtual machines, each running its own operating system. So it offers the image of a dedicated and customized machine to each user, decoupling them from the system software of the underlying resource. The software layer providing the virtualization is called a virtual machine monitor or hypervisor.

1.1.4 Service level Agreement (SLA)

Cloud computing is inevitably linked to the Quality of Service Concept. Emerging Cloud applications such as social networking, gaming portals, business applications, content delivery, ... have different Quality of Service (QoS) requirements depending on interaction patterns and time, the way that the service provider offers a given QoS is sealed with a contract: Service Level Agreement (SLA). This contract specifies thinks like the amount of guaranteed memory, CPU or response time among others, and also establishes rewards and penalties for accomplishing or violating such agreements.

SLA creation and negotiation is a very complex process and it is out of the scope of this project, but it is important for us to take it into account in our simulator, for this reason we suppose that the contract negotiation is done before the job arrives at the system. As a consequence of the negotiation we obtain some parameters that are supposed to be estimated from the negotiation and that give us an idea of which are the requirements of the job, allowing us to have a more accurate resource reservation.

1.2 Origin and Background

This degree project is developed within a grant from Ministerio de Educación y Ciencia for collaborating with the LSI department in the FIB at UPC. The collaboration consists on helping

Josep Lluís Berral during the current development of his doctoral (PhD) thesis [1] .This thesis is a multidisciplinar work joining two different research areas, autonomic computing and machine learning, directed by two professors of the Technical University of Catalonia, Jordi Torres from the *Departament d'Arquitectura de Computadors* and Barcelona Supercomputing Center, and Ricard Gavaldà from the *Departament de Llenguatges i Sistemes Informàic*s. Apart from that, the master thesis developed by Ferran Julià [2] became an important reference because it is in this context where the simulator was developed at first. Given that the project is multidisciplinary, a broad survey is necessarily.

The main goal of Josep Lluís' thesis is to let a complex distributed cloud system manage itself, using intelligent and adaptive techniques though machine learning and data mining. The agent responsible of including this intelligence into the cloud will be an autonomous controller component, capable of obtaining the required knowledge model, predict the system status and apply a set of policies guided by utility functions, in order to improve the performance, stability, adaptability and robustness of the cloud system. For instance, if it predicts that there will be a SLA violation or some system failure it should act in consequence and try to avoid it if possible.

Thee Emotive framework [20] is developed by some research groups of the UPC and the Barcelona Supercomputing Center (BSC-CNS). EMOTIVE abstracts a Cloud architecture using different layers and provides users with basic primitives for supporting the execution of services (features for resource allocation and monitoring, data management, live migration, and checkpointing, etc.). This software will be very useful to meet the need of mine for data of the Josep Lluís controller. Unfortunately we do not have a 5000 hosts grid to play with. Furthermore, one of the pretensions of the Josep Lluís controller is to be tested in heterogeneous cloud systems which would not be solved by having just one grid. We will give more information related with the simulation and our pretensions by using a simulator in Section 2, 'Introducing simulation'.

Ramón Nou, as part of his doctoral thesis [3] studied the relation between the performance and energy consumption of one machine used as a test bed [4]. After he did that study he implemented a simulator called EEFSim (we will call it EEFSimV1) which could simulate the energy consumption of a specified cloud. One of the main goals of emotive is to manage one cloud system minimizing its energy consumption so the directors of the Josep Lluís' thesis decided that it could be possible to use this simulator as a testbed if some extensions were developed, as supporting Virtualization. That solved the infrastructure problem.

The responsible of developing this new version of the simulator (called EEFSimV2) was Ferran Julià. The main goal of the Ferran Julià master thesis [2] was to understand the low level behaviors that rule a virtualized data center in which the cloud runs over, and to use all that experience in developing a simulator that permits the improve of the resource scheduling techniques introducing innovative green policies. The architecture of EEFSimV2 permits the usage of an external entity, a scheduler, to manage its behavior. This architecture has three different parts (see Figure 1.1). The workload generator, the simulator and the scheduler. The workload generator module sends the jobs to scheduler's core module at the specified arrival time. The scheduler schedule the jobs according to some policy. Any scheduler can be plugged in with the appropriate interface. The simulator simulates the behavior of the specified cloud system according to the schedule decisions and collects all the execution information and prints out the execution's power consumption, scheduling and resource usages statistics.

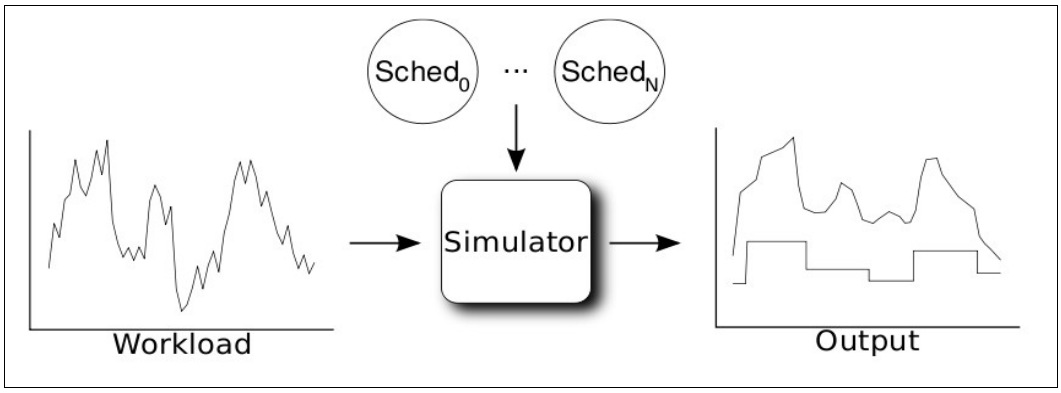


Figure 1.1 - Architecture of the simulator[1]

As it can be observed, at first appearance it fits the needs of a testbed for Josep Lluís' thesis but as he progressed in his work he noticed that the capabilities of this simulator were too limited for his objectives. The main problem is that one of the major ambitions of Josep Lluís' thesis is to achieve good scheduling of heterogeneous workloads but EEFSimV2 was designed for other purposes so it does not provide much heterogeneity respect its workloads and cloud system specifications. At this moment it is enough to take into account that there is the need of endow EEFSimV2 with more heterogeneity with respect to the tests that it could previously simulate. That is the frame in which my degree project was conceived.

Before we get in detail into the simulator structure, let us take a look at which are the main artifacts that interact in the proposed solution:

- The simulator: It is responsible of performing all the actions demanded by the scheduler. It has all the entities that compose the cloud system. Its function is to simulate the behavior of the cloud system if the actions demanded by the scheduler are performed. Apart from that, it is responsible of providing information relative to the cloud system architecture to the scheduler and giving the data related with the execution such as energy consumption, occupation of the machines during the simulation,etc.

- The scheduler: It is responsible of giving orders to the simulator for scheduling its workload in the different hosts that compose the cloud system. Its scheduling policy can change over time. Another important fact is that it will use the data provided by machine learning elements of the autonomous controller developed by Josep Lluís Berral.

- The autonomous controller: Its responsibility is obtaining the required knowledge, model and predict the system status, and apply a set of policies guided by utility functions.

1 Image obtained from [2]

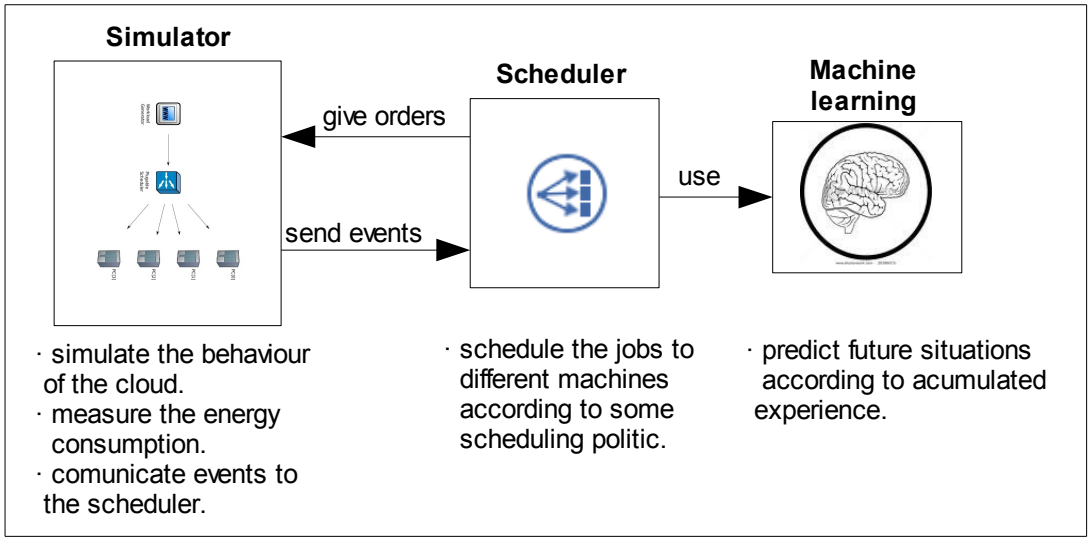


Figure 1.2 - Main artifacts of the simulation context

This artifacts are mapped into the emotive architecture as follows.

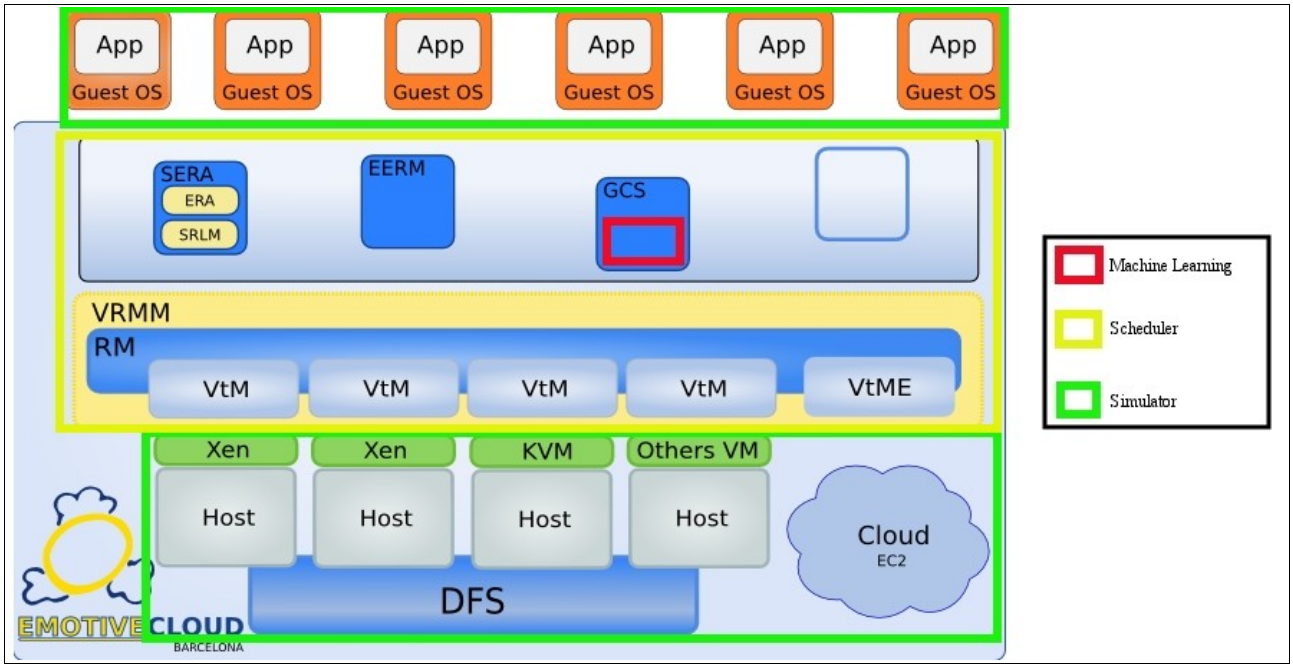


Figure 1.3 - Main artifacts of the simulation context mapped into the emotive architecture

1.3 Motivation

The development of a degree project consist on developing some system demonstrating the knowledge and skills that you have acquired along the degree. I have gone one step beyond. Due to the fact that I have worked in a research environment I have not tried to just develop something based on my previous knowledge. Instead of that, I have tried to keep learning and improving my skills and as a result I have developed some extensions to a simulator which is really avant-garde and unique in the world.

This degree project becomes an opportunity to enforce my communicative and comprehensive skills. As I just said in the introduction I am working with a group of people from interdisciplinary areas of the academy. Furthermore, the main topics of the students with whom I am working are at the very state of the art: cloud computing, virtual machines management, data mining, machine learning and much more. Apart from that, the opportunity of working in a university department has given me the opportunity of seeing how does research work.

Thats the reason why I think that this project has given me experience in one of the fields that university studies does not usually treat (research). It is therefore at the same time a perfect ending for my degree and a perfect beginning to get used to work with a group of people in the research area.

1.4 Goals of the project

The main goals of this project are:

- To extend EEFSimV2 for increasing the heterogeneity of not only the workload executed by the cloud but also the declaration of the specifications of the machines that form the cloud. Special effort should be directed to provide a good testbed for the Josep Lluís' doctoral thesis.

- To develop new modules of the simulator with a cohesive and extensible structure. This simulator is to be used by several research projects within the group so we can assume that it will be modified in the future. It is important to design every new module of the extensions providing the possibility of extending it in the future and with coherence. The maintenance and scalability of the simulator depends on these key features.

- To redefine the format of some of the documents used for specifying some of the modules of the simulator to acquire a extensible and semantic model of specification.

1.5 Functional requirements of the project

To succeed with the goals of the project we agree that the simulator should be able to perform the next requirements:

1. **Improve the representation of the CPU.**

   In order to improve the cohesion and scalability of the project we thing that is important to implement a new class to represent the CPU of the hosts that execute the workload of the cloud system. Apart from that we will improve the accuracy of the execution time of one application considering the characteristics related with the CPU of the machine where it is being executed.

2. **Take into account the memory usage of one task.**

   This requirement aims to improve the heterogeneity of the workload executed during the simulation. Furthermore, taking into account the memory usage of one application will improve the accuracy of the simulator in respect to the way the resources of the machines are used and the implied time to attend these needs.

3. **Take into account the usage of external devices of one task.**

   One of the bottlenecks during the execution time of one application is its usage of external devices as hard disks. The performance of one task could vary depending on its needs respect to that kind of resources. Apart from that, this kind of resources are limited so if more than one application attend to use it at the same time there must be a penalization at its execution time due to that they access sequentially to the device. That requirement will improve the heterogeneity of the workload as well.

4. **Improve the network representation.**

   One of the final objectives of that simulator is that it could be used in a commercial environment. The heterogeneity of the different networks and equipments that compound a cloud system is huge. This is the reason why one of the improvements that we need to perform is to bring the possibility of declaring heterogeneous networks. Apart from that we should use its characteristics for calculating the time that is needed to transmit data over it. That will provide more realism at the simulation results.

5. **Improve the way that some specifications are provided to the simulator.**

   In order to improve the maintenance tasks of that project and facilitate its scalability is very important to use an appropriate format to specify the configuration files that are used to specify some of the characteristics that are used during the execution of the simulation.

1.6 Document organization

The document is structured as follows: Chapter 2 summarizes the relevant features of simulation and describes some of relevant types of simulation that exist. Chapter 3 exposes the work related with the technology choice for develop the simulator. In Chapter 4 we give all the relevant information related with the selected technology: Omnet++. Chapter 4 summarizes the state of the simulator before performing the extensions presented in this degree project. Chapters 6, 7, 8, and 9 expose the work developed for each of the extensions. Chapter 10 give all the information related with the final validation of the simulator, after developing the extensions. In Chapter 11 we describe the schedule and the cost of the project comparing them with the estimation that we did after we began with the development of this project. Finally Chapter 12 collects all my conclusions.

2. Introducing simulation

In this section we will justify the decision of using a simulator instead of executing the controller in a real cloud system. First of all we will briefly explain what does the simulation consist on and explain why it is appropriate for use it as a testbed. At the end of this section we will describe the different kinds of existing simulations and their main characteristics.

Simulation in general is to pretend that one deals with a real thing while really working with something that behaves like it. From now on, when the term 'simulation' is used, it refers to a computer simulation, which means an attempt to model a real-life or hypothetical situation on a computer so that it can be studied to see how the system works. There is a huge variety of scenarios that have been studied by simulation. Simulation has become a useful part of modeling many natural systems in physics, chemistry and biology, and human systems in economics and social science (the computational sociology) as well as in engineering to gain insight into the operation of those systems. A good example of the usefulness of using computers for simulation can be found in the field of network traffic simulation. In such simulations, the model behavior will change each simulation according to the set of initial parameters assumed for the environment.

Any real-life system studied by simulation techniques is viewed as a system. The following features of a system are of interest:

1. Environment: Each system can be seen as a subsystem of a broader system.
2. Interdependency: No activity takes place in total isolation.
3. Sub-systems: Each system can be broken down to sub-systems.
4. Organization: Virtually all systems consist of highly organized elements or components, which interact in order to carry out the function of the system.
5. Change: The present condition or state of the system usually varies over a long period of time.

Some of the variables of the system that are of paramount importance are those used to define the status of the system. Such variables are known as status variables. These variables form the backbone of any simulation model. At any instance, during a simulation run, one should be able to determine how things stand in the system using these variables. Obviously, the selection of these variables is affected by what kind of information regarding the system one wants to maintain.

There are a lot of reasons that can justify using a simulation: security, testing, training, teaching, etc. In general, there are two main reasons to use a simulator. The first one is to predict something before it happens, for instance, weather prediction uses current weather conditions as input into mathematical models of the atmosphere to predict the weather. Although the first efforts to accomplish this were done in the 1920s, it was not until the advent of fast computers and computer simulations that it was feasible to did this with advantage. The second reason is to save money: flying with a simulator is safer and cheaper than building the real airplane. Often it is very costly, dangerous and often impossible to make experiments with real systems. Provided that models are adequate descriptions of reality (they are valid), experimenting with them can save money, suffering, and even time.

The controller that is being designed in Josep Lluís' thesis must interact with a cloud system, that means a cluster of hundreds of different machines. Furthermore the behavior of the controller should be tested in different kinds of clouds. Besides the fact that it is very difficult to find such amount of hardware for research and development, most of times it will not be acceptable to have so many machines, with all the maintenance work and the energy consumption that it involves, only for testing. This kind of infrastructures are usually only used in production systems and using them has an important economic cost.

We though that a simulator would be a suitable and effective solution for our problems, giving us the possibility to reproduce our solutions in real-like data-centers and validate our solutions. In a simulation we could use the data that we modeled from the real system avoiding all the complexity of the real cloud. For instance if we want to know how many time is being executed some application in the cloud system but there is no need of reproducing its output we can just keep it in the system without executing any code. A well designed simulator can provide results accurate enough for our proposals. Furthermore a simulator usually can run on a simple machine and often reduce the test time several times, so that would save lots of energy and time.

2.1 Types of simulation

Computer models can be classified according to several independent pairs of attributes, in this section we will describe the different types of simulations that exist considering those attributes. Some of the definitions in this chapter are taken from [25].

Continuous vs discrete simulation

In some simulations the status of the system changes each time an event occurs. During the time that elapses between two successive events, the system status remains unchanged. In view of this, it suffices to monitor the changes in the system status. Jumping in the time line from one event to the next one optimizes the execution time of the simulation. This kind of simulation is called discrete event simulation.

The opposite kind of simulation is the continuous simulation. Continuous simulation concerns the modeling over time of a system by a representation in which state variables change continuously with respect to time. For example, the water level in a reservoir with given in and outflows may change all the time. In such cases "continuous simulation" is more appropriate, although discrete event simulation can serve as an approximation.

Stochastic vs deterministic simulation

Deterministic models are models which do not contain the element of probability. Examples are: linear programming, non-linear programming and dynamic programming. Stochastic models are models which contain the element of probability. Examples are: queueing theory, stochastic processes, reliability, and simulation techniques.

In simulation, random numbers are used in order to introduce randomness in the model. For instance, if we are able to observe the operational times of a machine over a reasonably long period, we will find that they are typically characterized by a theoretical or an empirical probability

distribution. Therefore, in order to make the simulation model more realistic, one should be able to randomly numbers that follow a given theoretical or empirical distribution.

Simulation techniques rely heavily on the element of randomness. However, deterministic simulation techniques in which there is a no randomness, are not uncommon. Simulation techniques are easy to learn and are applicable to a wide range of problems.

Transient state vs. steady-state simulation

In general, a simulation model can be used to estimate a parameter of interest during the transient state or the steady state.

Let us assume that one is interested in obtaining statistics pertaining to the number of broken down machines of one system. The simulation starts by assuming that the system at time zero is at a given state. This is known as the initial condition. Evidently, the behavior of the system will be affected by the particular initial condition. However, if we let the simulation run for a long period, its statistical behavior will eventually become independent of the particular initial condition. In general, the initial condition will affect the behavior of the system for an initial period of time, say T. Thereafter, the simulation will behave statistically in the same way whatever the initial condition. During this initial period T, the simulated system is said to be in a transient state. After period T is over, the simulated system is said to be in a steady state.

One may be interested in studying the behavior of a system during its transient state. In this case, one is mostly interested in analyzing problems associated with a specific initial starting condition. This arises, for example, if we want to study the initial operation of a new plant. Also, one may be forced to study the transient state of a system, if this system does not have a steady state. Such a case may arise when the system under study is constantly changing (dynamic system).

Typically, a simulation model is used to study the steady-state behavior of a system. In this case, the simulation model has to run long enough so that to get away from the transient state. That means that a steady-state simulation is one in which the measures of performance are defined as limits as the length of the simulation goes to infinity. There is no natural event E to terminate the simulation, so the length of the simulation is made large enough to get “good” estimates of the quantities of interest.

2.2 Related concepts

Emulation

In the context of software, an emulator reproduces the behavior of one system on another. It executes, or strives to execute, the same programs as the "original" system, and produces the same results for the same input. It is important that the user of an emulator is not supposed to care *how*. Although a simulation model and an emulation model may look to all intents and purposes the same, and may be built largely with the same building blocks, there are significant differences in usage and operation.

Simulation models are used to test and develop different solutions in order to arrive at a best solution, based on an accepted set of pre-defined metrics. Simulation often provides the impartial judge between experience and new ideas, and allows the user to demonstrate functionality and results in a cost-effective and flexible environment. Simulation results help define the physical layout of a system, its operating limits and its control system. Models are used as a basis for extensive experimentation, often using automatic procedures to determine optimal or robust solutions.

Emulation models are used in a much more precisely defined way; in order to test the operation of the control system under different system loading conditions, and as a risk-free means of training system operators and maintenance staff. Emulation models are not used for experimentation in the same way that simulation models are; they are unsuited to this function as they often execute only in real time.

The emulation model reflects more precisely the system that will be implemented, and as such, can be used to carry out a constrained series of verification procedures to ensure the performance or reaction of the control system. Systems exist to automate the execution of these tests, and to run them in parallel in order to carry out more tests than would be practical on the real system.

Virtualization

A common mistake that people not familiar with virtualization make is that they think virtualization is similar to emulation or simulation. The fact of the matter is virtualization is neither emulation or simulation . In Section 1.1.3, 'About Virtualization and Virtual Machines (VM's)', we provide a detailed description of what does virtualization consist on.

3. Technology choice

There are several features that make an ideal simulation package. Some are properties of the package, such as support, reactivity to bug notification, interface, etc. Some properties come from the user, such as their needs, their level of expertise, etc. For these reasons asking which package is the best one has no use. The first question to be asked is for what purpose you need the software. In this case is for an student-project or research so the used simulator must be robust and enough customizable to permit the implementation of all the capabilities of the simulation purpose. Apart from this, it is also important what it provides: easy input, allow some programming, good framework for creating customized modules, message exchange and supports a discrete, stochastic and transient simulation.

As we said in the introduction, we began our work from a existing simulator (EEFSimV2) which used one simulation package called Omnet++. If we had decided to change the simulation environment it would involve some time for rewriting EEFSimV2 to the new one, but it does not mean that we did not have to contemplate this alternative. In fact, if we had found some simulation package that fits better with our purposes it would involve saving a lot of time in the future.

In this section we describe the alternatives that we considered for developing this project before choosing that Omnet++ is the better one. There are many simulators designed for simulating a network and its behavior. In this section we will describe the main simulators that can be found currently and some of their advantages and disadvantages.

3.1 COTSon

COTSon [19] is a simulator framework jointly developed by HP Labs and AMD. The goal of COTSon is to provide fast and accurate evaluation of current and future computing systems, covering the full software stack and complete hardware models. It targets cluster-level systems composed of hundreds of commodity multicore nodes and their associated devices connected through a standard communication network. COTSon adopts a functional-directed philosophy, where fast functional emulators and timing models cooperate to improve the simulation accuracy at a speed sufficient to simulate the full stack of applications, middleware and Oss.

One of the advantatges of COTSon's functional simulator is that it uses AMD's SimNowTM simulator. The SimNow simulator functionally models most of the existing hardware that can be found on an AMD system. Other companies and partners, including HP, developed their own functional devices for the SimNow platform. Full-system simulation means that all aspects of the system can be analyzed by the simulator. So it can provide a complete accuracy for the devices that it provides.

Another important fact is the methodology that it uses to balance the performance and the accuracy of its simulations. COTSon makes its functional simulator run for a dynamically set interval. This produces a stream of events which are sent to the respective CPU timing models. At the end of the interval, using the metrics from the CPU models (which include the whole memory hierarchy) a new IPC (instructions per cycle) is fed back to the functional simulator. By selecting different

interval sizes, the user can turn the accuracy vs speed relation. Moreover, COTSon's approach couples very well with sampling, enabling the user to select just those intervals which are considered representative or "interesting."

But COTSon has an important drawback for our purposes: in order to simulate a whole cluster, COTSon instantiates several COTSon node simulators, potentially in different host machines. Each of them is a stand-alone application which communicates with the rest via a network mediator. The network mediator acts as a functional network switch, directing network packets to the appropriate COTSon node destination. So it does require a heavy infrastructure, one of the problems we want to avoid.

Despite that, it can be useful when we need some reference, for tuning some of the characteristics of the different devices that we are going to use, such as memory latency or time delay of migrating a virtual machine.

3.2 PARSEC

PARSEC [10] (for PARallel Simulation Environment for Complex systems) is a C-based discrete-event simulation language. It adopts the process interaction approach to discrete-event simulation. An object (also referred to as a physical process) or set of objects in the physical system is represented by a logical process. Interactions among physical processes (events) are modeled by timestamped message exchanges among the corresponding logical processes.

One of the important distinguishing features of PARSEC is its ability to execute a discrete-event simulation model using several different asynchronous parallel simulation protocols on a variety of parallel architectures. Although that it is not important from our point of view. The purpose of the simulator is to execute the behavior of a cloud network and collect the information that will be used to schedule the jobs minimizing the energy consumption of the devices without penalizing the performance so much.

Furthermore, it is reported in [2] that there is not much documentation about how to use it. Aparently it is not very popular and not supported by any big enterprise or community.

3.3 NS2

NS2 [11] is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

It is an object oriented simulator, written in C++, with an OTcl interpreter as a frontend. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy), and a similar class hierarchy within the OTcl interpreter (also called the interpreted hierarchy). The two hierarchies are closely related to each other; from the user's perspective, there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy. The root of this hierarchy is the class TclObject. Users create new simulator objects through the interpreter; these objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in

the compiled hierarchy.

The interpreted class hierarchy is automatically established through methods defined in the class TclClass. user instantiated objects are mirrored through methods defined in the class TclObject. There are other hierarchies in the C++ code and Otcl scripts; these other hierarchies are not mirrored in the manner of TclObject.

In this case the official website [11] gives more documentation but the purpose of that simulator is more focused in the study the network behavior instead of the hole system.

3.4 VnetUML

VNUML (Virtual Network User Mode Linux) is an open-source general purpose virtualization tool designed to quickly define and test complex network simulation scenarios based on the User Mode Linux (UML) virtualization software. VNUML is a useful tool that can be used to simulate general Linux based network scenarios. It is aimed to help in testing network applications and services over complex testbeds made of several nodes (even tenths) and networks inside one Linux machine, without involving the investment and management complexity needed to create them using real equipment.

VNUML tool is made of two main components: the VNUML language used for describing simulations in XML; and the interpreter of the language (vnuml command), that builds and manages the scenario hiding all UML complex details to the user.

A typical working cycle using VNUML is made up of the following phases:

- **Design phase**. First of all, the user must design a simulation scenario. For that, several aspects have to be considered in advance: the number of virtual machines, the topology (network interfaces in each machine and how they are connected), what processes each virtual machine will execute, etc. Note that the whole simulation runs in the same physical machine (named host).
- **Implementation phase**. Once the simulation scenario has been designed, the user must write the source VNUML file describing it. This is an XML file. There are three kinds of tags in VNUML: structural tags, with few or no semantics; topology tags, used to describe the topology; and simulation tags, used to describe simulation parameters and commands.
- **Execution phase**. Once the user has written the VNUML file, he must run vnumlparser.pl against it in order to build and manage the simulation scenario.

As it can be observed it has some aspects in common with the simulator to be developed: configuration of the devices and topology in XML file and only needs one host. But it does not support much customization, and while it offers several tags they are not enough enough for the purpose of our simulator. Furthermore it only supports Linux and this could penalize the heterogeneity of the cloud system to simulate.

3.5 Netsim

NetSim is a versatile tool to simulate and analyze computer networks. With comprehensive

modeling facility, detailed performance reports and enhanced protocol analytics, NetSim offers unmatched power and flexibility. It is very graphic and has some libraries that simulates the behaiviour of some protocols used in the networks communications as Gigabit Ethernet and Wireless LAN 802.11 a and g.

Although as it happened with VnetUML, it does not support much customization and the purpose of this simulation is more in a academic line, which means that it is used for giving a practical and graphical point of view about the behavior and metrics of a different network configurations. Furthermore it is proprietary software which means that it would increase the cost of this project and its code is private. It is important to highlight that we did not discard it because it is proprietary software. We discarded it because it does not support much customization.

3.6 Omnet++

OMNeT++ [12] is an object-oriented modular discrete event network simulation framework. OMNeT++ itself is not a simulator of anything concrete, but it rather provides infrastructure and tools for writing simulations. One of the fundamental ingredients of this infrastructure is a component architecture for simulation models. Models are assembled from reusable components termed modules. Well-written modules are truly reusable, and can be combined in various ways. Modules communicate through message passing.

It has all the features we were looking for: easy input, allow some programming, good framework for creating customized modules, message exchange and supports a discrete, stochastic and transient simulation. As it is the selected framework we will give more information about its features and how does it work in Section 4, 'Omnet++'.

3.7 Conclusion

Omnet++ is clearly the best of these simulators for our purposes. Omnet++ offer a really customizable environment because it is implemented in C++. Furthermore in the official website [12] we can find a lot of documentation and interesting samples about how to use it.

Other simulation tools like NS2, NetSim and PARSEC are centered in other purpose than creating a customized simulator. NS2 and NetSim are designed for showing the behavior of a network.

VnetUML and COTSON are limited but they can be useful if need to take some measures about the time latency of some of the devices.

4 Omnet++

As Omnet++ is the selected framework to implement the simulator we should describe its behavior and functionalities. In this section we make emphasis on those aspects of the simulator that will be needed to follow the implementation details of the extensions that we developed during the realization of that project.

4.1 General description

OMNeT++ [12] is an extensible, modular, component-based C++ simulation library and framework, with an Eclipse-based integrated development environment (IDE), with which we are familiarized, and a graphical runtime environment. It is a discrete event simulation environment. Its primary application area is the simulation of communication networks, but because of its generic and flexible architecture, is successfully used in other areas like the simulation of complex IT systems, queueing networks or hardware architectures as well.

OMNeT++ provides a component architecture for models. Components (modules) are programmed in C++, then assembled into larger components and models using a high-level language (NED). Reusability of models comes for free. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into your applications.

Although OMNeT++ is not a network simulator itself, it is currently gaining widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community.

The screenshots showed in Figure 4.1 and Figure 4.2 are an example of the two views of a NED component. The first one shows how the modules and submodules can be showed into a graphical and intuitive mode. The second one shows the code behind the module.

OMNeT++ is free for academic and non-profit use, and it is rapidly becoming a preferred simulation platform in the scientific community worldwide.

This framework offer all the features that the simulator needs. The fact that the components are programmed in C++ makes easy to create and parse the xml documents that are used for specifying the workload and the topology of the network. Furthermore high-level language (NED) make it easier to describe the modules. Another important fact is that it had Eclipse-based IDE which is very popular and intuitive. In addition it have two ways of execute the simulation, the first one is with a graphical environment which shows all the messages exchange between the different modules and all the events that are produced during the execution. This one is the best one to debug the behavior of the network. The second one is executed in a shell. This one is faster and is the best alternative to run an execution when you want to collect all the data produced on it.

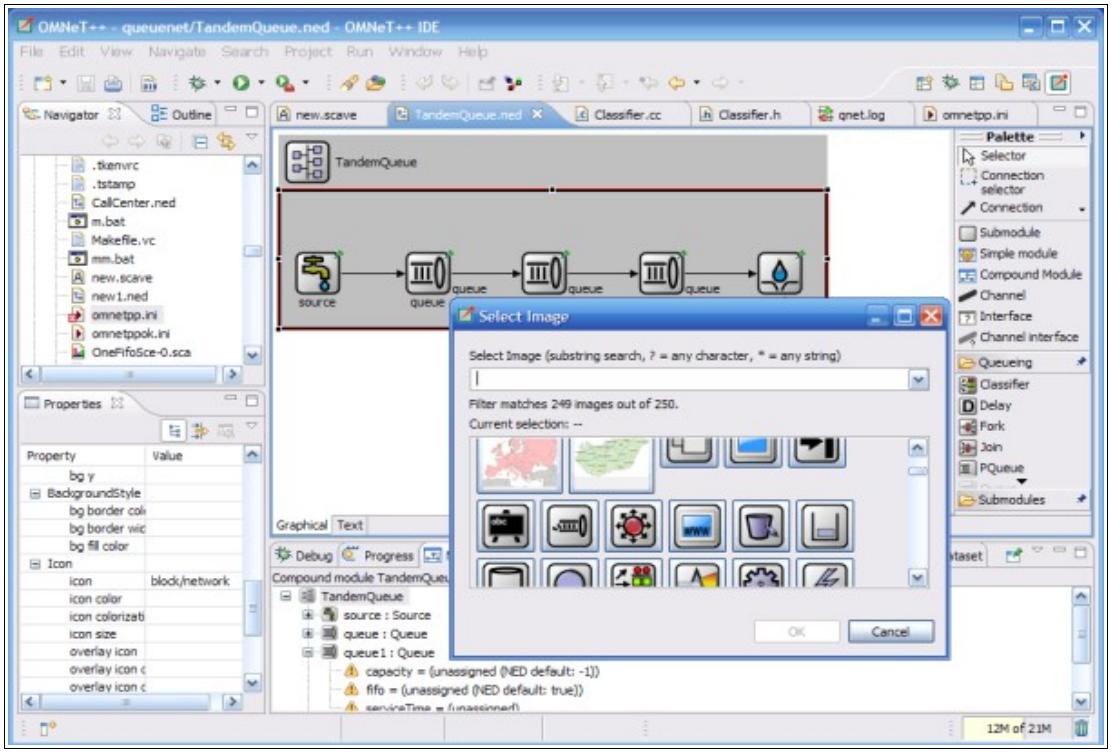


Figure 4.1 - Graphical NED editor[2]

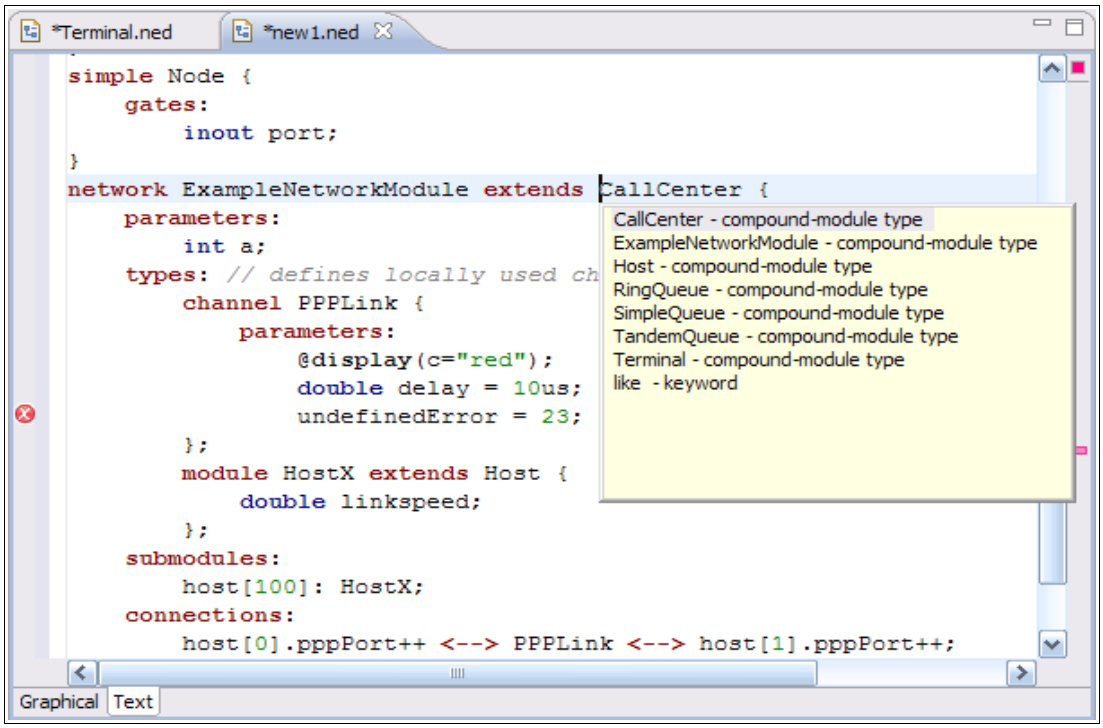


Figure 4.2 - NED source editor[3]

2 Image obtained from http://www.omnetpp.org/home/screenshots.

3 Image obtained from http://www.omnetpp.org/home/screenshots.

4.2 How it works

An OMNeT++ model consists in the following parts:

- NED language topology description(s) (.ned files) which describes the module structure with parameters, gates etc. NED files can be written using any text editor, but the OMNeT++ IDE provides excellent support for two-way graphical and text editing.
- Message definitions (.msg files). You can define various message kinds and add data fields to them. OMNeT++ will translate message definitions into full-fledged C++ classes.
- Simple modules sources. They are C++ files, with .h /.cc suffix.

Despite that, there is a special file with .ini extension where the parameters of the different modules can be set.

An OMNeT++ model consists in modules that communicate using message passing. The active modules are termed simple modules; they are written in C++, using the simulation class library. Simple modules can be grouped into compound modules and so forth; the number of hierarchy levels is not limited. The whole model, called network in OMNeT++, is itself a compound module.

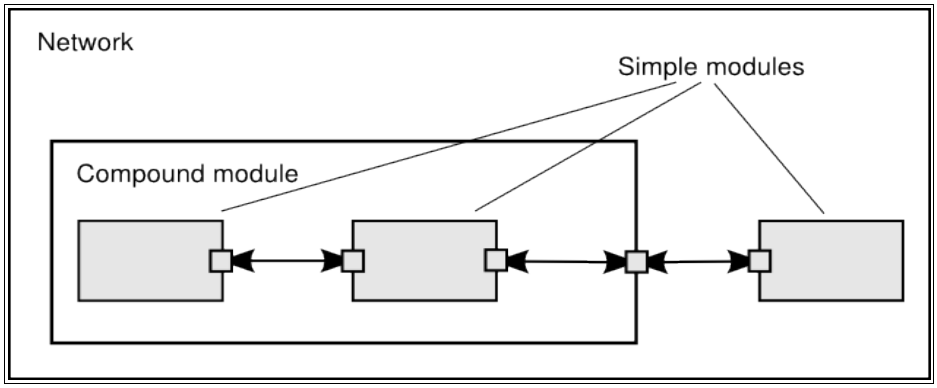


Figure 4.3 - Simple and compound modules[4]

Modules communicate with messages which, in addition to usual attributes such as timestamp, may contain additional data. Simple modules typically send messages via gates, but it is also possible to send them directly to their destination modules.

Modules can have parameters. Parameters can be assigned either in the NED files or the configuration file omnetpp.ini. Parameters may be used to customize simple module behavior, and for parameterizing the model topology.

When the program is started, first of all it reads all NED files containing the model topology and then it reads a configuration file (usually called omnetpp.ini). This file contains settings that control how the simulation is executed, values for model parameters, etc.

4 Image obtained from http://www.omnetpp.org/doc/omnetpp41/manual/usman.html.

5. Starting point of the simulator

As said in the introduction, the simulation presented in this degree project is an extension of the simulator designed in Ferran Julià masther thesis [2], called EEFSimV2. In this section we will explain the main characteristics of EEFSimV2 with a special effort to explain all the underlying technical background concepts. Then we will describe all the modules that compose that first version one by one.

5.1 EEFSimV2

This simulator have three main modules: the workload generator, the pluggable scheduler and the grid of hosts that compound the cloud system.

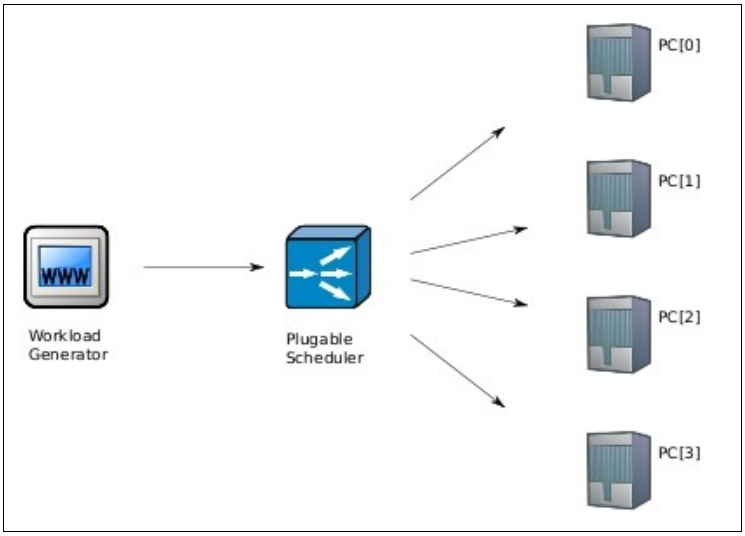


Figure 5.1 – Internal architecture of the simulator[5]

We will describe these modules with detail in Section 5.3, 'Modules', but let us take a first look at the functionalities of each of these modules.

The workload generator is the module that creates the workload that is going to be executed. All the jobs that are going to be executed into the cloud are specified in one file. The workload generator parses that file and generates the corresponding jobs to be dispatched to the pluggable scheduler using messages.

The pluggable scheduler is the module that represents the scheduler which is going to take the decisions about how are scheduled the different VM's that are created for attending each job. It is important to highlight that the scheduler and the pluggable scheduler are not the same. As we say in the introduction, the scheduler is external to the simulator. That module communicates with the scheduler by remote procedure calls, an inter-process communication technology that allows a computer program to call a subroutine or procedure to be executed in another address space (usually

5 Image obtained from [2]

on another computer on a shared network). Basically it performs three kind of calls: one to notify to the scheduler that a host is added to the grid (used when the host modules are being initialized), one for requesting the creation of a new VM to allocate some job, and a third one for requesting which operations are going to be performed by the simulator (adding, migrating, removing or chekpointing some VM).

The `PC` are the unique compound module of the EFFSimV2. It has two modules. The `PowerSupply` which is responsible for collecting the energy consumption of the host and the `HyperScheduler` which represents the hypervisor that manages the VM's that are going to be executed by the host. In the next section we will explain some important aspects related to the virtualization mechanism and its internal behavior.

5.2 Virtualization

In order to understand better the challenges that involve the creation of a virtualized data center simulator, in this section we will introduce some essential internals of virtualization systems. Notice that these mechanisms play a key role in the structure as they determine the way real CPU is assigned to virtual CPUs and have a great influence on the overall behavior.

The finality of EEFSimV2 was to test some parts of a real Cloud management middleware called emotive [6],developed within J.Torres' group. The emotive framework is built on the Xen [7] virtualization system, so the simulator has to reproduce some of the Xen's internal mechanisms and low level behaviors.

Xen uses a virtualization technique called paravirtualization, which requires the partial modification of the guest operating system. As the guest O.S. is modified, some special operations call hypervisor instead of calling the hardware, and it is the hypervisor who manages the real resources and schedule them among the virtual hosts.

There is a special host in the system, called "dom0" (domain 0), that is always running in the host and that have some special operations that permit controlling the hypervisor. This special module in the O.S. is in charge of creation, destruction and migration of virtual machines.

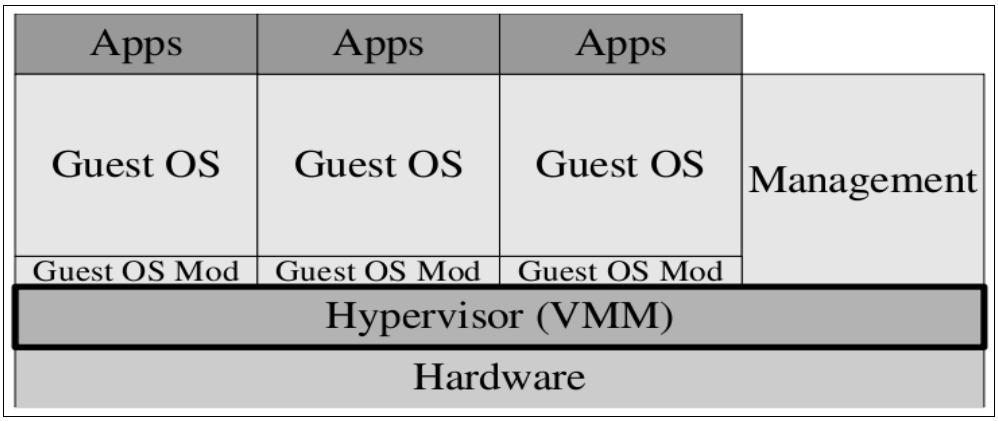


Figure 5.2 – Xen architecture[6]

6 Image obtained from [2].

The CPU scheduler of EEFSimV2 is based on the Credit Scheduler [13]. The most important to notice from the credit scheduler is the Capabilities and Weight operation modes. With these parameters we can tune the scheduler behavior giving more ore less execution time or priority to the different virtual machines.

The scheduler assigns real CPU to virtual CPUs maintaining the same proportionality than have the ”weight“ parameter for the virtual machines. Each parameter is set at virtual machine level, so all virtual CPUs of the same virtual machine have the same weight parameter as it happens with the capability parameter. For example, if we have 2 virtual machines with 1 virtual CPU for each one running at the same host, if both have the same weight the real CPU will be shared at 50% for both, but if one have a weight value of 20 and the other of 80, the real CPU will be 20% for the former and 80% for the second. Notice that what is important here is the relation among the two weight values, nor the real value of them, instead of 20 and 80 values we could have 40 and 160 and the result would be the same 20%-80%. The ”Capability“ parameter fixes the amount of real CPU that one virtual machine can use, so despite of what we calculated from the weight parameter, if the real amount of CPU that is allocated for the virtual machine is grater than the capability the virtual machine will not have more real CPU than the capability. With these two parameters we can adjust dynamically the real CPU that we allocate for each virtual machine thus control the behavior of our system.

5.3 Modules

The modules are the backbone on the simulation in omnet++. In this section we will describe the modules that compose EEFSimV2. We highlight that the general behavior and the main modules of that compose that simulator had been described in Section 5.1, 'EEFSimV2'. This section can be skipped on a first reading.

5.3.1 WorkloadGenerator

This module is the one that generates all the workload that will be executed by the network. For doing it in the most practical and extensible way it is generated from a text file. This allows having different kinds of workload in different files. To define the path of the file that contains the specific workload that is going to be used in the execution, this module has a parameter called `workload`.

The format of the file that contains the workload is as follows:

```
#startTime  name     duration   userCpu    userMem
  26399     app22     2217       200         128
  26463     app200    514        200         128
...
```

As can be observed, for each job it defines start time, the name of the application configuration file which should be specified with the same name with a .load extension (for instance, app22.load), duration (in seconds), and user cpu and memory that it requires. The application configuration file must be defined too. It has the following format:

```
#startTime  cpuUsage    memUsage    SLA
0           200         128         1.2
...
```

The start time represents the time (during the execution of the job) in which the corresponding cpu, memory usage and SLA are valid. So, with this format, for each job, there can be intervals with different cpu and memory usage and SLA.

The `WorkloadGenerator` loads all the information described in those files and generates entities objects of type `WorkloadComponent` that are sorted in a queue called `vwc` (vector workload component). For this moment its enough to understand that a `WorkloadComponent` represents a job. For more information about this entity refer to Section 5.5.1, 'WorkloadComponent'.

5.3.2 PlugableScheduler

This module is the one that manages communication with the external scheduler which gives the directives to follow in the simulation. For this reason it has some functions that serves as a feed back between the state of both schedulers (the external one and this one).

In this module is where the configuration of the network is performed. It uses two parameters to set the configuration of the computers of the network: the `hostConfig` and the `specsDir`. The first one specifies the path of the file that contains the specification of the type of each computer. The format of this file is as follows:

```
#type descriptor
pctinet
...
```

There is one of these descriptors for each PC. This descriptor links to another file. This file can be found in the path setted in the `specsDir` attribute. The format of this file is:

```
#CPUs freq Memory(GB) boot migrationIn migrationOut IdleBattage bootPower
 4    3000 4          20   7           3            230         275
```

As it can be observed, for each PC we have the following descriptors: its number of CPU's, the frequency of its CPU's, the total amount of RAM memory, the penalty of booting up the machine and migrating in or out some virtual machine (all of them in seconds), and its energy consumption when it is in a idle or booted state.

For the moment, this module is connected to all the PC's that from the cloud which in this version is a grid of computers. This means that there is no possibility of describing different network topologies. Across these connections, this module establishes communication with all the PC's and schedule the workload generated by the `WorkloadGenerator`.

5.3.3 PC

This is a composite model. So its behavior is described more precisely in the next sections, when the submodules that compose it are described. As a first approximation we can analyze the submodules: a `PowerSupply` and `HyperSched`. The first one is the responsible of collecting data referred to the energy consumption of the `PC`. The second one is the responsible of manage and schedule all the virtual machines that are being executed in the `PC`.

The HyperSched is connected to the `PowerSupply`. Furthermore, the input and output of the `PC` are connected to the HyperSched.

5.3.4 HyperScheduler

This model represents the hypervisor of the physical machine. It is the responsible of scheduling the virtual machines that are being executed in the `PC`. Every virtual machine consumes some of the resources (like CPU, memory,...) of the computer. This module computes the proportion of real CPU that is assigned on each of its virtual machines; in the last paragraph of Section 5.2, 'Virtualization', there is a detailed description about how is it done. Every time quantum, `HyperSched` calculates the amount of CPU that is consumed and sends it to `PowerSupply` for taking it into account when computing the energy consumption of the device.

As a virtual machine manager, it is responsible of creating, booting, migrating and destroying its virtual machines. The module that represents a virtual machine is `LocalScheduler`. `LocalScheduler`'s are created dynamically during the simulation so they are not initialized with parameters specified in the '.ini' file, unlike `PCs`. Of course, there is a connection between the HyperSched and all the virtual machines that it is executing.

5.3.5 PowerSupply

As it was anticipated in Section 5.3.3, 'PC', this is one of the submodules of the composed module `PC` and it serves to calculate the consumption of the `PC`. It is computed having two parameters into account, the first one is the number of CPU's that are being used and the second one is the global CPU usage. This amount of consumed power is calculated every quantum of `HyperScheduler` so it is not an estimation, it is an exact figure. This module is directly related with [4].

Observe that it is one of the modules that will need to be changed in the future work, because it does not take into account consumption of the memory and other devices.

5.3.6 LocalScheduler

This module represents a virtual machine that is being executed in a `HyperScheduler`. As it is said in Section 5.3.4, 'HyperScheduler', the `HyperScheduler` schedule and manage the virtual machines (LocalScheduler's). The main task of this module is to manage and schedule all the jobs that are being executed in this virtual machine. For simulating the execution of the jobs, we will follow an

abbreviation of the model that use the operating systems: the five-state process model.

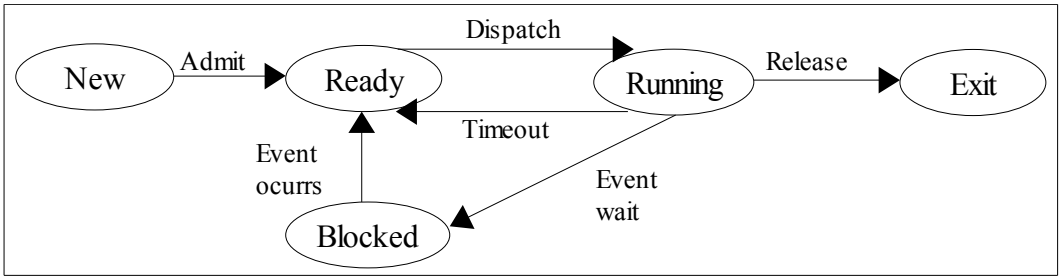


Figure 5.3 – The five-state process model

As we can observe in Figure 5.4, when a job begins its execution, it competes with the other jobs for the resources of the virtual machine. The mechanism used by the operating system for giving the sensation that all the jobs are executed at once is separateing the elapsed time executing the different programs in quantums. One quantum is the period of time that a process is allowed to run uninterruptedly. The scheduler is run once every time slice to choose the next process to run. For implementing that model, we need two different queues: one for the processes that are running and other one for the processes that are blocked. At every quantum of the execution, the processes that are in the running queue are executed and then are swapped with the next ones in the blocked queue.

5.4 Messages

In OmNet++, when a module has to communicate something to other modules it has to do it in two steps: the first one is to have a gate that connects it to the receiver module, the second one is to send it a message. So, the messages are used for communication between modules. They can be customized for transporting the required data. In this section we will describe the messages that are used in EEFSimV2.

5.4.1 SimMessage

In EEFSimV2 there is just one kind of Message, the `SimMessage`. It has four parameters: `cpuUsage`, `cpuIndex`, `bootTime` and `workShared`. The two first attributes are used between the communication of the HyperScheduler and `PowerSupply` modules. The last one is used as a pointer to any of the structures used in the simulator. Is a kind of parameter to enclosure data at the message.

One of the things that will need to be improved in the future work is the use different types of messages to customize every message to its purpose.

5.5 Other C++ files

Some times the simulation need some classes that do not represent any module because they do not

need to exchange messages. For instance, there could be some file to encapsulate some information or providing some functionality to third files. This is the reason why OmNet++ permits the use of any C++ class. In this section we describe the C++ files that are used in this simulation.

5.5.1 WorkloadComponent

The `WorkloadComponent` is the class that contains all the information of a job. It is used as an auxiliar for the `WorkloadGenerator`. The next table shows the main attributes of this class.

name	*kind*	*description*
`type`	int	0=normal;1=creation;2=migrationIn; 3= migrationOut
`name`	string	
`startTime`	unsigned long long	
`duration`	unsigned long long	
`jobID`	int	
`SLA`	double	Service Level Agreement.
`execTime`	unsigned int	The duration of the execution of that job.
`CPUFlow`	list<unsigned long long, unsigned int>	A list of pair of values where the first one refers to the moment of the execution in which the CPU value (the second one) is active.
`MEMFlow`	list<unsigned long long, unsigned int>	Is the same concept that is explained at CPUFlow but with the memory consumed by the job.

Table 5.1 – WorkloadComponent main attributes

5.5.2 VirtualMachine

The `VirtualMachine` class is used when the `PlugableScheduler` must give the order to create a new `VirtualMachine`. This class keeps the information and sets it to the corresponding `LocalScheduler` when it is created, after it is booted up. That means that the creation of a `LocalScheduler` is not instantaneous, so some class must keep the information between the decision of create it and the moment that it is active, when is booted up. The main parameters of this class are described in Table 5.2. In Section 5.2, 'Virtualization', we have given detailed information about how are used the `weight` and `cap` (from capability) attributes.

name	*kind*	*description*
`state`	int	-1 = init; 0 = boot; 1 = run; 2 = migratingIn; 3 = migratingOut ; 4 = migratingOutEnded
`localId`	int	
`vCpus`	int	The number of virtual CPU's that are assigned.
`mem`	int	The amount of memory that are assigned.
`cap`	int	The maximum real CPU that the VM can use.
`weight`	int	The priority of the VM, used by the `HyperScheduler` to manage the VM's.
`requested`	int	The amount of CPU that are requested by the jobs.

Table 5.2 – WorkloadComponent main attributes

6. First extension: Change the representation of the CPU

In EEFSimV2 the CPU was just represented by an integer called `numCPUs`. That is not a scalable structure, in the future work we want to simulate processors with scalable frequency and processors with different features. Furthermore, to get more details of the CPU consumption, we need to have some related information about it, for instance characteristics of the cache hierarchy. These are some of the reasons that make that our first extension was creating a new class that represent the characteristics of the CPU in a compact and extensible way.

6.1 CPU background

The Central Processing Unit (CPU) or the processor is the component of a computer system that carries out the execution of the instructions of a program, and is the primary element carrying out the computer's functions. It is formed by several devices like the datapath, the memory cache and the clock of the system. As it is in charge of executing the instructions, its characteristics have a direct impact on the performance of the computer.

6.2 Previous representation

In EEFSimV2 the data related to the CPU were represented by two parameters of the `HyperScheduler`: `numCpus` and `freq`, which represent the number of CPUs and its frequency. Because one of the goals of our work was to model CPU's with more detail, we decided to implement one new class which will encapsulate all the data related with the CPU of the machine and its components. This will include the two mentioned parameters, `numCpus` and `freq`, and possibly others that are required for realistic simulation of current CPU's

Apart from encapsulating this data to provide more cohesion to the simulator and easy scalability of the CPU modeling, we aggregate other functionalities affecting the execution of the workload and giving more realism at the simulator: permitting vary the frequency of the CPU's, also known as frequency scaling.

6.3 Used programs

In order to analyze the variation of the CPU frequency we created a simple microbenchmark that only performs CPU instructions. There is a pseudo code of this program in the Figure 6.1. If want to watch the source code find it at Annex A.

```
program(nAccesses, nIterations){
        size = 20000 * 20000;
        partition = size / nAccesses;
        matrix = createMatrix(size,size);
        aux = 0;
        for(i=0;i<nIterations;i++){
                position = random() mod partition;
                aux++;
                if (nAccesses>1) aux++;
                if (nAccesses>2) aux++;
                if (nAccesses>3) aux++;
                if (nAccesses>4) aux++;
        }
        free(matrix);
}
```

Figure 6.1 – Pseudo code of the implemented benchmark without memory accesses

6.4 CPU experimentation

We make the hypothesis that the relation between the time elapsed for the execution and the frequency of the processor would be linear for different CPU frequencies, but this must be verified because there could be external factors that take part of the elapsed time.

The experimentation consists on measuring the duration of the execution of that benchmark for a determined frequency and then we change the frequency of the machine obtaining the new duration. Table 6.1 shows the results obtained.

frequency	2.10 GHz	1.60 GHz	1.20 GHz	800 Mhz
duration (s)	9.231	12.293 (12.115)	16.176 (16.154)	24.214 (24.23)

Table 6.1 – Results of the execution times of one job with frequency scaling

We consider an original frequency of 2.10 GHz and compare frequencies to the first one. The result specified between parentheses is the expected result. As we expected the relation between CPU frequency and the elapsed time is approximately linear (for this range of frequencies). It could be appreciated in Figure 6.2.

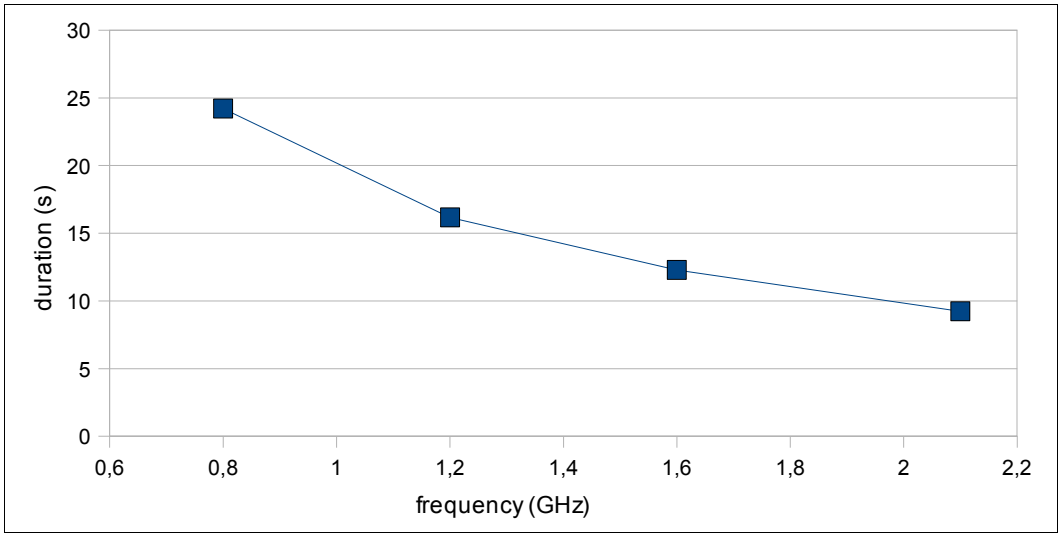


Figure 6.2 – Results of the execution times of one job with frequency scaling

6.5 CPU modeling

Up to now, the duration of one job was constant. But as we observed in our experiments the time elapsed executing CPU instructions has a linear relation with the frequency of the CPU. That means that if we have one program running CPU instructions during T_{CPU} seconds in a machine with f_0 frequency and we run the same program in another machine with a frequency f_{host}.(or the same machine with different frequency), the elapsed time in this machine will be:

$$T_{CPU}' = T_{CPU} * (\frac{f_{host}}{f_0})$$

With that simple formula we obtain the resulting CPU utilization time that corresponds to this new frequency. We highlight that we just proved that for a concrete rank of frequencies (between 800MHz and 2.1 GHz).

6.6 CPU implementation

In this section we will explain the behavior and attributes of the new implemented class, CPU. Moreover we will explain the modifications to the `WorkloadComponent` and `LocalScheduler` classes for modeling the variation in their duration depending on the frequency of the host where it is executed, because of the frequency scaling.

CPU

This new class have the following attributes and methods.

Attributes		
name	***kind***	***description***
`idCPU`	`int`	Identifier of the CPU.
`working`	`bool`	Indicates if the CPU is working.
`consuming`	`bool`	Indicates if the CPU is consuming.
`currentCPUUsage`	`float`	Indicates the CPU that is being used.
`powerConsumed`	`float`	Indicates the power that is being used.
`qRunning`	`list <WorkloadComponent *>`	The Running queue of the CPU with all the jobs that are being executed in it.

Methods	
header	***Description***
`void initialize();`	Initializes the attributes of the class.
`addJob(WorkloadComponent * job);`	Add the specified job to the CPU, which means that it will dedicate its quantum to execute that job.
`bool feet(WorkloadComponent * job);`	Return true if the specified `job` cpu usage can be hold by the CPU.
`bool hasFinishedJobs();`	Return true if the CPU has some job that finished its execution.
`list<WorkloadComponent *> collectExitingJobs();`	Return the list of the jobs that request exiting from the CPU.
`void clear();`	It clears the `qRunning` and `currentCPUUsage` of the CPU.
`bool isNotFull();`	Returns true if the CPU has some free portion of its usage.
`int AssignMaxim(WorkloadComponent * job, int required );`	It allocates the maximum portion of the CPU quantum to the specified job.
`int getId();`	Returns the identifier of the current CPU.

Table 6.2 – CPU parameters and methods

Shown in Table 6.2 this new class collects all the relative information about the CPU and can be easly extended.

WorkloadComponent

As we said before, the duration of one program has to be recomputed depending on the frequency of the machine where it is executed. Here is the specification of the function that will implement that.

Methods	
header	***description***
`void recalculateDuration(unsigned int frequency);`	This function recomputes the duration of the job considering the frequency of the machine where it is going to be executed, as it is explained in Section 6.4, 'CPU Experimentation'.

Table 6.3 – Workload added methods

LocalScheduler

In this module we add a function calling `recalculateDuration(unsigned int frequency)` from the `WorkloadComponent` every time that some job is added. Apart from implement this change, we have also to implement the full integration of `CPU` class because before we added this new class the `numCPUs` attribute was used.

6.7 Future work

As a short term future work we will study the behavior of the duration of one program in a wider range of frequencies. The fact that its relation with the time spent executing some program is linear in the frequencies that we tested does not mean that it behaves like that for all the range of possible frequencies. Apart from that, its behavior could vary depending on the CPU architecture so we could also study if there is the need of implementing a different characteristic function for every CPU brand.

As a long term future work there are a lot of possibilities to extend the functionalities related with the CPU. For taking more detailed data about its energy consumption we could integrate the ACPI [22] (Advanced Configuration and Power Interface) which establishes industry-standard interfaces enabling OS-directed configuration, power management, and thermal management of mobile, desktop, and server platforms. Another future work could be to implement the quantum variation depending on the CPU frequency.

7. Second extension: Modeling the memory

Until now the only factor that was took into account while modeling the jobs was the time it needs to be executed in a machine. For developing a good scheduler is necessary to take into account more factors. For instance, If you know the memory usage of a program you can take better decisions when you have to schedule it to one machine or another. Lets imagine that you have two different machines. One has better memory latency than the other and both have the same cost per byte. If you receive a new job with a lot of memory accesses you should schedule it to the machine with better memory latency, but if it does not, maybe is more interesting to schedule it to the other machine and reserve the best machine when a possible future work comes up demanding a lot of memory accesses.

7.1 Memory background

In practice, a memory system is a hierarchy of storage devices with different capacities, costs, and access times. The closer the device is to the CPU the less access time it has. CPU registers hold the most frequently used data. Small, fast cache memories nearby the CPU act as staging areas for a subset of the data and instructions stored in the relatively slow main memory. The main memory stages a subset of the data stored on large, slow disks.

Memory hierarchies work because programs execute the same instructions the most of time. So the storage at the next level can be slower, and thus larger and cheaper per bit. The overall effect is a large pool of memory that costs as much as the cheap storage near the bottom of the hierarchy, but that serves data to programs at the rate of the fast storage near the top of the hierarchy.

This idea is based in a fundamental property of computer programs known as locality. Programs with good locality tend to access the same set of data items over and over again, or they tend to access sets of nearby data items. Programs with good locality tend to access more data items from the upper levels of the memory hierarchy than programs with poor locality, and thus run faster. For example, the running times of different matrix multiplication kernels that perform the same number of arithmetic operations, but have different degrees of locality, can vary by a factor of 20!

Analyzing the usage of the different levels of the memory hierarchy is difficult and is not enough for computing how much time is elapsed accessing the memory in the execution of one program. Managing the memory is not trivial and performance memory access can vary depending on the different policies that can be adopted by the memory management unit.

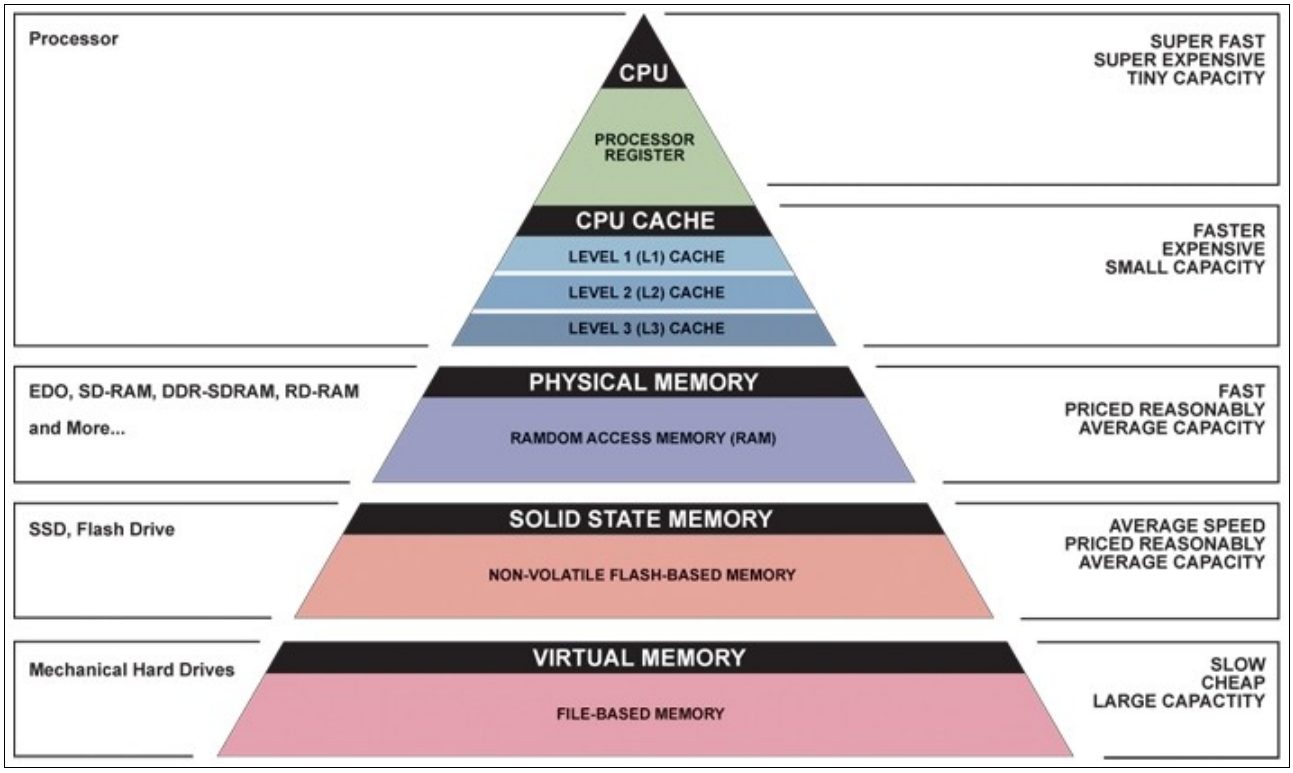


Figure 7.1 – Memory hierarchy[7]

These are some of the terms related with the memory management unit that affect to the memory access performance:

Page fault

A page is a fixed-length block of memory that is used as a unit of transfer between physical memory and external storage like a disk, and a page fault is an interrupt (or exception) to the software raised by the hardware, when a program accesses a page that is mapped in some of the memory devices, but not loaded in the cache.

Paging and swapping

There are various ways in which the operating system can store and retrieve data from secondary storage for use in main memory. One such memory management scheme is referred to as paging. In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called pages.

As we explained every level of the memory hierarchy contains a subset of the data stored in the lower level. So the size of the page that is exchanged between the main memory and the secondary memory will affect to the elapsed time for swapping this amount of data. The same happens with the exchange of data in the upper levels of the memory hierarchy. The term used for describing the set of the data exchanged in that level is called block.

7 Image obtained from http://images.bit-tech.net/content_images/2007/11/the_secrets_of_pc_memory_part_1/hei.png

Virtual memory

The virtual memory is a technique where the computer looks at RAM for areas that have not been used recently and copies them onto the hard disk. This copying happens automatically, so the user does not even knows that is happening, and it makes the computer look like is has unlimited RAM space.

Trashing

Virtual memory systems work most efficiently when the ratio of the working set to the total number of pages that can be stored in RAM is low enough to minimize the number of page faults. A program that works with huge data structures will sometimes require a working set that is too large to be efficiently managed by the page system resulting in constant page faults that drastically slow down the system. This condition is referred to as thrashing: pages are swapped out and then accessed causing frequent faults.

Optimizations

Accessing the memory is one of the bottlenecks during the execution of a program, so a lot of optimizations have been developed in several different levels of a computers architecture (hardware, operating system, compilers,..) to obtain less page faults. For instance, there is a technique called swap prefetch. A few operating systems use anticipatory paging. These operating systems periodically attempt to guess which pages will soon be needed, and start loading them into RAM. Thereby when the page is needed it does not provoke a page fault.

7.2 Memory modeling

It is very difficult to model the memory usage by one program and which part of its execution corresponds to performing memory accesses. Determining these aspects is, even today, an active research field, and I am happy that I was able to touch one real research problem during my project.

During the modeling of the memory we try to find three different ways to model it. In this section we will describe these hypothesis and how we get from one to another.

7.2.1 First model: Considering all the devices that form the memory hierarchy

As the time spent accessing the memory depends on which of the different levels of the memory hierarchy allocates the requested data, we suppose that it is important to take into account all the devices that form the memory hierarchy for modulate the memory access behavior.

Finding how much of the total time spent in a execution of a program corresponds to each of the levels of the memory hierarchy is not trivial. This information is difficult to calculate because it depends on many factors, and can vary from one execution to another, but as main factors we can consider two characteristics which every level has: the latency and the probability of a miss.

Understanding all the facts related with the memory management unit is very difficult. Besides there are several different levels related with the memory management (hardware, operating system, compiler,...) there are also many optimizations that makes practically impossible to measure how much time is used to access the memory in the execution of a program. But of course, it is possible to make an approximation considering some of the main important facts that affect to it. In our case we measure the memory utilization and its latency for each of its hierarchy levels. Now its time to compare the theoretical used time and the practical one. This formulas computes the theoretical time:

$$T_{MEM1} = acc_{L1} * lat_{L1} + acc_{L2} * lat_{L2} + acc_{MM} * lat_{MM}$$

where T_{MEM1} refers to the time is used by accessing the memory theoretically. The real time used with memory accesses is computed as T_{MEM2}. To measure T_{MEM2} we execute two programs that do essentially the same but one access to a vector (which is loaded in the memory) and the other one does not. The difference between the execution time of those programs is the time used for accessing the memory.

$$Texec_1 = T_{CPU} + T_{MEM}$$
$$Texec_2 = T_{CPU}$$
$$T_{MEM2} = Texec_1 - Texec_2$$

Considering that there are a lot of facts that influence to the memory management that we are not considering, it is logical that it will be an error between the theoretical and real used time. Our intention is to investigate if that error is absolute or relative to the calculated time and if it is constant or not.

7.2.1.1 Used programs

It was not a trivial matter choosing the right programs to measure the latency and utilization of each memory level on a real systems. After considerable time investigating different possibilities, we decided to choose two different programs to measure complementary aspects. The first one is valgrind [14] and we used with its tool cachegrind [15] to measure memory accesses. The second one is LMBench [16]. Apart from using these programs we used a microbenchmark in C which we use for forcing a lot of page faults.

Valgrind

Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile programs in detail. The tool used for memory profiling is the Cachegrind. Cachegirnd simulates how your program interacts with a machine's cache hierarchy. It simulates a machine with independent first level instruction and data caches (I1 and D1), backed by a unified second level cache (L2). To use it you must run the next command:

```
$valgrind --tool=cachegrind prog arg
```

where prog is the name of the program and arg is the arguments the program requires to run (of course, it can be more than one). Once it finishes the simulation it give some information like this:

```
==31751== I   refs:      27,742,716
==31751== I1  misses:           276
==31751== L2i misses:           275
==31751== I1  miss rate:        0.0%
==31751== L2i miss rate:        0.0%
==31751==
==31751== D   refs:      15,430,290  (10,955,517 rd + 4,474,773 wr)
==31751== D1  misses:        41,185  (    21,905 rd +    19,280 wr)
==31751== L2d misses:        23,085  (     3,987 rd +    19,098 wr)
==31751== D1  miss rate:        0.2% (       0.1%   +       0.4%)
==31751== L2d miss rate:        0.1% (       0.0%   +       0.4%)
==31751==
==31751== L2 misses:         23,360  (     4,262 rd +    19,098 wr)
==31751== L2 miss rate:         0.0% (       0.0%   +       0.4%)
```

Figure 7.2 – Sample of Valgrind output (using Cachegrind).

You can appreciate the accesses that the program has performed in the different memories that compound the memory hierarchy and distinguishing the accesses to the instructions memory and the data memory. So, with that tool we can measure the utilization of each memory level.

LMBench

LMBench is a Suite of simple, portable benchmarks. It can be used to measure a lot of factors related with the usage and performance of the different resources of a system. There are three main groups of benchmarks: bandwidth benchmarks latency benchmarks and miscellaneous. We wont describe all the possibilities that it offers, just the way we use it. We decided to use it after reading the Intel paper [9] . As said this paper the LMBench has one benchmar for calculating memory latency, and that is the one we used. The memory latency test shows the latency of all of the system (data) caches, i.e., level 1, 2, and 3, if present, as well as main memory. The current version can be downloaded from [23]. These are the results obtained in our case (for memory latencies):

```
Memory latencies in nanoseconds - smaller is better
------------------------------------------------------------------------------
Host           OS             Mhz  L1 $  L2 $    Main mem
---------      -------------  ---  ----  ----    --------
IMMASTER  Linux 2.6.24-  2280 1.3220 6.6360   89.8
```

Microbenchmark

The pseudocode of the microbenchmark designed to produce as many page faults as possible while accessing the memory hierarchy is in Figure 7.1. In this program we declare a array of ints (integers) with 400 million positions. Considering that every int uses 4 Bytes of memory, that means that we use a array of approximately 1.6 GB's. We do this because we want to cause as page faults as possible. To warranty that the maximum page faults are raised we use an algorithm that generates as many partitions of the vectors as accesses we will do in every iteration. To select the

specific position in which we will access (in every partition) we set a random number. It will serve us as a program that causes the maximum number of page faults because at every iteration it accesses positions that are fare from each other. This means that these positions will be in different pages and the there will be no locality.

```
program(nAccesses, nIterations){
        size = 20000 * 20000;
        partition = size / nAccesses;
        matrix = createMatrix(size,size);
        for(i=0;i<nIterations;i++){
                position = random() mod partition;
                matrix[position];
                if (nAccesses>1) matrix[partition + position];
                if (nAccesses>2) matrix[2*partition + position];
                if (nAccesses>3) matrix[3*partition + position];
                if (nAccesses>4) matrix[4*partition + position];
        }
        free(matrix);
}
```

Figure 7.3 – Pseudo code of the implemented benchmark with memory accesses

7.2.1.2 Experimentation

To measure the utilization of different levels of the cache hierarchy we used the program described in Section 7.3.1.1, 'Used programs'. We vary the number of iterations for analyzing the accesses at the different memory hierarchy levels.

it	Tmem theor (s)	Tmem real (s)	Error	Relative Error
2	82,353	59,241	23,112	**0,281**
5	202,711	141,116	61,595	**0,304**
10	403,322	272,086	131,236	**0,325**
50	2008,064	1343,44	664,624	**0,331**

Table 7.1 – Relative error between the theoretical an real execution time varying the number of iterations

As it can be observed, the relative error is always close to 0,3. As we comment in previous sections, this because of the accesses that we are measuring with valgrind are the total accesses but not all the accesses that are performed during the execution of a program implies a time penalization. Some techniques as pre-fetch the memory are used by the OS. That techniques cause some memory access that do not implie that the program suffer the memory latency penalization. This is the reason why there is an relative error between the results we obtain and the real ones.

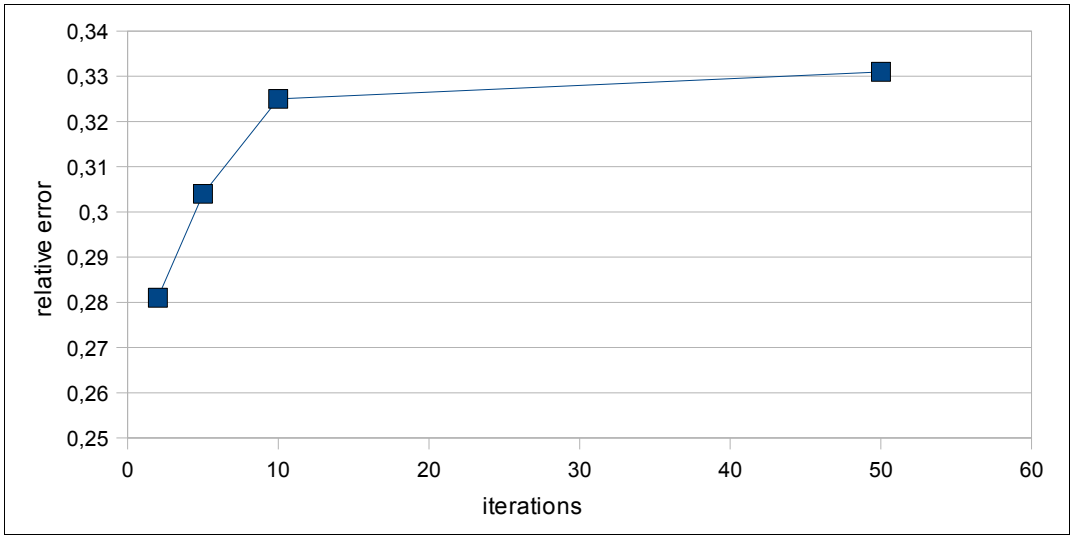


Figure 7.4 – Relative error between the theoretical an real execution time varying the number of iterations

7.2.1.3 Conclusion

Despite the relative error between the calculated time and real time trends to 0,3 assuming that this behavior will represent all the heterogeneous applications of the workload is too much risky. As we said in Section 7.2, 'Memory background', there are a lot of factors that interfere in the time elapsed accessing the memory by one application. One of the main factors are the page faults which can vary depending on the context of the execution of the application. So we decide to analyze the behavior of the memory considering it as a block without making distinctions on its hierarchy levels.

7.2.2 Second model: Considering the memory as a full block and supposing that the CPU time consumption and the memory time consumption are independent on each other

This new model tries to analyze the memory as a whole instead of analyzing each component of the memory hierarchy. As we appreciated, when we take into account all the memory levels separately is very difficult to analyze the behavior of the execution of one program in respect to its use of the memory. Despite that, the results that we obtain have a common behavior for the program that we designed but this does not proof that the behavior will be the same for another application. So we decided to look at the behavior of the memory in a simplest way just analyzing the elapsed time accessing the memory as a full block.

To model the memory utilization of one program we consider that only some of the time that it is being executed will be spent accessing the memory. So for modeling it is necessary to know which is the ratio between the memory and the CPU utilization. Furthermore it is important to take into account that for modeling a job what we do is to execute it in a machine and measure its behavior (time consumption, memory utilization,...). We can not assume that the behavior of that job will be the same in every machine where it is executed. Of course, the time it needs to be executed will vary depending on the architecture and characteristics of the machine where it is executed. The

memory takes part of that variation, so apart from model the ratio between the memory and CPU utilization its important to model the latency of the memory of the machine where the job was originally modeled.

For measuring the ratio between the time spent accessing the memory and the execution time we measured the execution we use the same formulas that in Section 7.3.1, 'First model: Considering all the devices that form the memory hierarchy':

$$Texec_1 = T_{CPU} + T_{MEM}$$
$$Texec_2 = T_{CPU}$$
$$T_{MEM} = Texec_1 - Texec_2$$

And the ratio between the memory and the CPU utilization will be the following:

$$MEMvsCPU = \frac{T_{MEM}}{T_{CPU} + T_{MEM}} = \frac{T_{exec1} - T_{exec2}}{T_{exec1}}$$

For measuring the latency of the memory (as a full block) we will calculate the time needed to serve one access the memory. To compute it we will use:

$$lat_{MEM} = \frac{T_{MEM}}{acc_{MEM}}$$

where lat_{MEM} is the memory latency and acc_{MEM} are the total memory accesses.

With these two factors, the ratio between memory and CPU usage and the memory latency where the job was executed, we estimate the duration of the execution of the same program depending on the machine where it is executed. We will present an example of this kind of extrapolation. Imagine two different machines with the following characteristics.

Machine	$Texec_1$ (sec)	$Texec_2 = T_{CPU}$ (sec)	T_{MEM} (sec)	acc_{MEM}	lat_{MEM}	freq(MHz)
A	20	10	10	100	0,1	2000
B	30	10	20	100	0,05	2000

Table 7.2 – Sample of data required by the model

Now imagine that the job characteristics are measured in the machine A. The model of that job will be as follow:

job<Texec=20,...,MEMvsCPU=0.5,$latency_0$=0.1,$freq_0$=2000,1>

If that job is executed in the machine B the time spent executing that job will vary because its latency is worse than the one of the machine A. We can obtain the new execution time with the formulas:

$$T_{MEM_A} = T_{exec} * MEMvsCPU$$

$$T_{CPU_A} = T\,exec * (1 - MEMvsCPU)$$
$$T_{exec_B} = T_{CPU_A} * (\frac{f_0}{f_B}) + T_{MEM_A} * (\frac{lat_B}{lat_0})$$

that can be resolved as follows:

$$T_{MEM_A} = T_{exec} * MEMvsCPU = ٢٠ * ٠,٥ = ١٠$$
$$T_{CPU_A} = T\,exec * (١ - MEMvsCPU) = ٢٠ * ٠,٥ = ١٠$$
$$T_{exec_B} = T_{CPU_A} * (\frac{f_.}{f_B}) + T_{MEM_A} * (\frac{lat_B}{lat_.}) = ١٠ + ١٠ * (\frac{٠,١}{٠,٠٥}) = ٣٠$$

As can be observed, with the characteristics of the job and the machine which is going to execute the job we can estimate the execution time in the new machine.

7.2.2.1 Used programs

In this section we use again valgrind (described in Section 7.3.1.1, 'Used programs') for measuring the number of memory accesses. Apart from that we use the same benchmarks that we have described in Figure 6.1 and Figure 7.1.

7.2.2.2 Results of the experimentation

To evaluate the results obtained we will make two comparisons: the first one will be between programs with different ratio between the time spent in memory access and the time spent with CPU usage and the second one will be varying the memory latency of the machine.

Results obtained varying the ratio

For this test we run the program described in Section 7.3.2.1, 'Used programs', with different accesses per iteration. This are the results obtained:

Acc x It	Texec 1 (sec)	Texec 2 =T_{CPU} (sec)	T_{MEM} (sec)	ratio
1	15,85	82,43	66,58	**0,807**
2	19,28	110,62	91,34	**0,826**
3	19,68	140,69	121,01	**0,86**
4	19,98	171,32	151,34	**0,883**
5	19,38	189,55	170,17	**0,898**

Table 7.3 – Results obtained varying the ratio

As it can be appreciated, the time spend executing the CPU operations remain practically constant, but the time spend accessing the memory increase while we take more accesses per iteration.

Results obtained varying the latency

For do that test we execute the programs described in Section 7.3.2.1, 'Used programs', in different machines. These are the results we obtained.

Machine	Texec 1 (sec)	Texec 2 =T_{CPU} (sec)	T_{MEM} (sec)	Accesses	Latency (ns)
original	83,746	17,857	65,889	17.793.658.194	**3,703**
pctinet	39,660	7,328	32,332	17.793.658.164	**1,817**
pcabril	81,743	20,458	61,285	17.793.657.599	**3,444**
lucifer	55,586	17,251	38,335	17.793.657.970	**2,154**

Table 7.4 – Results obtained varying the latency

As it can be observed the accesses needed in the different machines are almost the sames but the time spend accessing the memory vary depending on the machine.

Results obtained varying the frequency of the computer

For do that test we execute the programs described in Section 7.3.2.1, 'Used programs', in the same machine with different frequencies. This are the results we obtained.

frequency	2.10 GHz	1.60 GHz	1.20 GHz	800 Mhz
Texec2 (s)	9.231	12.293 (12.115)	16.176 (16.154)	24.214 (24.23)
Texec1 (s)	80.289	86.1 (83.174)	102.848 (87.212)	133.953 (95.289)

Table 7.5 – Results obtained varying the frequency of the computer

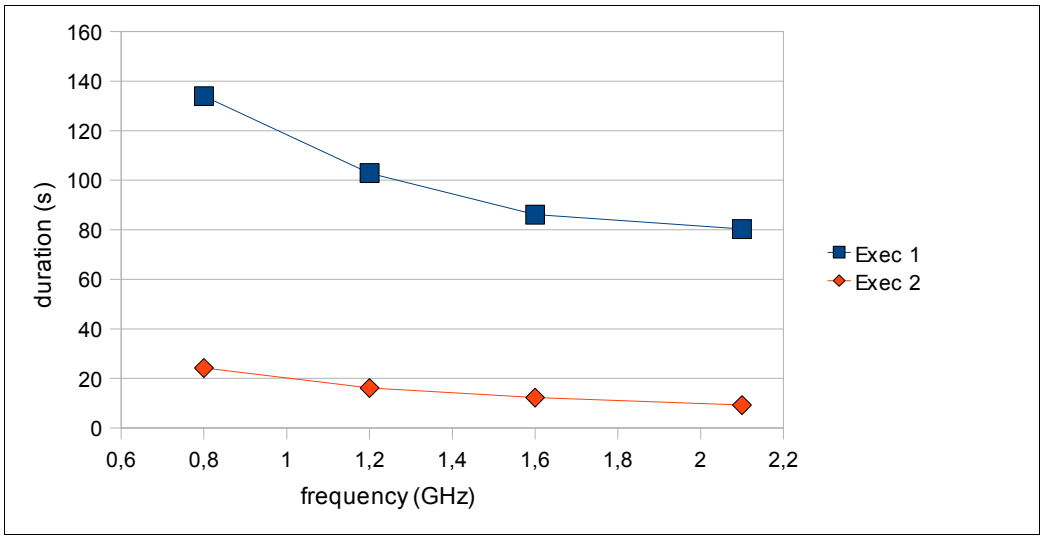


Figure 7.5 – Results obtained varying the frequency of the computer

We assume that the original frequency was 2.10 GHz. The results showed in parenthesis are the predicted result using our modelation of the problem. As we explain in Section 6.5, 'CPU modeling', the predicted results when we just take into account the CPU usage are correct but the predicted results related with the execution of the program that accesses the memory are not correct. The only explanation of that problem is that we made a bad assumption: we considered that the time elapsed during the memory accesses is independent from the CPU frequency, and obviously it is not.

7.2.2.3 Conclusion

This modeling of the behavior of one program with respect to its CPU and memory usage is not correct because we assume that the time elapsed accessing the memory is independent from the CPU characteristics of the system. As we proved in Section 7.3.2.2, 'Results of the experimentation', this is not true.

The explanation to that issue is that there is a synchronization between the CPU and the memory management unit when some instruction has to access the memory, so when the frequency of the machine vary, there is a penalization related with this synchronization.

7.2.3 Third model: Considering the memory as a block and supposing that the CPU time consumption and the memory time consumption have dependencies on each other

Because there exist dependencies between the frequency of the CPU and the elapsed time by accessing the memory we decide that it is very difficult to try to model this problem with one simple formula. It is important to insist on the fact that there are many factors related with the time consumption of the instructions that access the memory in the execution of one program.

Apart from that our objective is to simulate an approximation of the real behavior of one machine so we try to change our strategy for facing that problem by this other one: to analyze different types of programs in relation with its memory usage and try to design some program prototypes that will have different behavior in respect to calculating its penalization when they are migrated from one machine to another with different memory characteristics.

Our proposal was to experiment with three different applications: one that does not access many times the memory (high performance), one that use a lot of memory (movie codec) and another one that loads a lot of data during its execution (weka).

The objective of that modeling is to add an extra factor to the formula that we use in the previous model. This factor will vary depending on the type of the job which will be classified as one of our prototypes. This is the next formula that we propose for recalculating the time execution of one job depending on the characteristics of the machine where it is executed:

$$T_{CPU}' = T_{CPU} * (\frac{f_{host}}{f_0})$$

$$T_{MEM}' = T_{MEM} * (\frac{lat_0}{lat_{host}})$$
$$T_{EXEC}' = T_{CPU}' + T_{MEM}' * \alpha$$

Where the α is the factor that will vary depending on the type of program.

7.2.3.1 Used programs

We use another tool from valgrind to profile the memory accesses that are performed during the execution of the three programs that we selected for prototyping the different kind of programs that could exist considering its memory usage. This tool is called lackey, and is a simple Valgrind tool that does various kinds of basic program measurement.

We run it enabling the trace-mem property which prints the size and address of almost every memory access made by the program. So we used the next command for measuring the memory accesses of one program:

```
$ valgrind --tool=lackey --trace-mem=yes prog arg 2> file
```

where prog is the name of the program and arg is the arguments you pass it and file is the file where we store the output of that application.

Once the execution finishes we run the next command:

```
$ wc -l file
```

which gives us the number of lines of the file. This is how we get the number of memory accesses. Apart from that this program gives this output:

```
==18056==
==18056== Counted 1 call to main()
==18056==
==18056== Jccs:
==18056==   total:        16,683
==18056==   taken:         6,861 ( 41%)
==18056==
==18056== Executed:
==18056==   SBs entered:   17,224
==18056==   SBs completed: 10,669
==18056==   guest instrs:  99,554
==18056==   IRStmts:       564,593
==18056==
==18056== Ratios:
==18056==   guest instrs : SB entered  = 57 : 10
==18056==       IRStmts : SB entered  = 327 : 10
==18056==       IRStmts : guest instr = 56 : 10
```

```
==18056==
==18056== Exit code:      0
```

Figure 7.6 – Sample of Valgrind output (using Lackey).

where IRStmts gives the number of performed instructions.

7.2.3.2 Experimentation

These are the three programs that we selected for prototyping the different kind of programs analyzing its memory usage: the simulator, which represents a program with high performance, one movie encoder, which represents a program that performs a lot of memory accesses, and weka, which represents programs using a lot of RAM memory during its execution.

These are the results obtained:

Program	# memory accesses	# executed instructions	Relation (memory acc/instr)
simulator	28M	110M	0,254
video encoder	459M	1817M	0,253
weka	500K	1747K	0,286

Table 7.6 – Results obtained while trying to identify memory usage prototypes

7.2.3.3 Conclusion

The results obtained during the experimentation are very different that the ones we expected. As can be appreciated, the ratio between the instructions that use the cpu and the memory accesses does not vary as much as we expected. In fact, it is approximately constant. That means that on the one hand optimizations related with memory make that its usage remains more than less constant despite the amount of memory that is used by the program; on the other hand. that even if the program makes a lot of memory accesses, it usually performs the same ratio of instructions that do not depend on memory accesses.

That made us decide considering the modeling of the variation of the time execution of one program depending on its memory usage as a future work. All this work will be used in short term future but there are other factors that are interesting for extending the simulator and we dedicate already too much time to this part. It is important to accomplish our objectives but is also important to be practical and in this case it was preferable to develop other extensions rather than that one.

7.3 Memory implementation

Even if we did not succeed in finding how the memory characteristics affect program executions in

general, we implemented some new classes and modified others to model the memory in the simulator. It is important to highlight that although we did not succeed by now, we consider it as a future work an exhaustive study on that issue. In this section we present a new class named *Memory* and specify all the details that have been implemented in relation to memory usage.

Memory

This is a completely new class developed for represent the memory characteristics that we want to implement. In the future it can be extended. The attributes and functions of this class are the following ones:

Attributes		
name	***kind***	***description***
`id`	`string`	Identifier of the memory.
`size`	`unsigned int`	Size of the memory, measured in MB.
`latency`	`double`	Latency per memory access, measured in ns.
`currentUsage`	`unsigned int`	Is the surrent usage of the memory, measured in MB.
`attendedJobs`	`std::list<jobAssigment>`	List of jobs allocated to this memory with its correspondent memory usage. The type `jobAssigment` is defined as follows: `typedef pair<WorkloadComponent *, unsigned int> jobAssigment;`
Methods		
header		***description***
`Memory(string id,unsigned int size,double latency);`		Class constructor.
`double getLatency();`		Returns the latency of the memory.
`unsigned int getSize();`		Returns the size of the memory.
`unsigned int getCurrentUsage();`		Returns the current usage of the memory.
`bool allocate(WorkloadComponent * job);`		Allocates the job passed by parameter in the memory which means that it is inserted in the `attendedJobs` list. It returns true if the job can be allocated (there is enough memory) and false if it cannot.
`bool erase(WorkloadComponent * job);`		Delete the job passed by parameter in the memory which means that it is erased from the `attendedJobs` list. It returns true if the job is allocated in the memory and false if it does not.
`void clear();`		Clears the `attendedJobs` list and sets the memory usage to 0.

Table 7.7 – Methods and attributes from Memory class

As it can be observed, apart from defining the main attributes that we need to respect the memory modelation we add the list of the jobs that are being allocated inside the memory. This can be used in a future work to take into account that depending on the current usage of the memory its latency can vary assuming that it would need more swapping.

HyperSched

As explained while we introduce the modules of the simulator, the `HyperSched` is the module that represents the hypervisor of the machine and is responsible of managing and scheduling the virtual machines being executed on it. For managing the memory we add one attribute of class *Memory*. The next table shows the information about that attribute.

Attribute		
name	***kind***	***description***
`memory`	`Memory`	Represents the memory of the machine.
Methods		
header	***description***	
`unsigned int` `getMemoryUsed();`	Returns the amount of memory that is currently being used.	
`double` `getMemoryLatency();`	Returns the latency of the memory.	
`void` `setMemory(Memory * pMem);`	Sets the memory pointed by `pMem` to the `HyperScheduler`.	

Table 7.8 – Added methods and attributes to the HyperSched

Apart from that we had to introduce many modifications on its code because that is the module that boots, executes and migrates the virtual machines. We have to allocate or delete space in the memory every time that one of these actions is performed.

WorkloadComponent

As we said, the duration of one program has to be recalculated depending on the frequency and the latency of the memory of the machine where it is executed. Here is the specification of the function that will implement that.

Functions	
header	***description***
`void` **`recalculateDuration`**`(double memoryLatency,unsigned int frequency);`	This function recalculate the duration of the job considering the latency of the memory and the frequency of the machine where it is going to be executed, as it is explained in Section 7.3.2, 'Second model'.

Table 7.9 – Added methods and attributes to the WorkloadComponent

PluggableScheduler

As the definition of the `HyperScheduler` had changed, cause now it define the characteristic of its memory, the modules that parse its definition and create its instances had to implement some little variations.

7.4 Future work

Due to the fact that we dedicate too much time to this extension and we did not completely succeed we decided to leave it as a short term future work. It is important to highlight that it does not mean that we lost all the time. This is a current research topic, and this is how research works. Section 11, 'Project scheduling', will give more details about how the problems related with the memory modeling affected to the initial scheduling of this project.

As a long term future work we will take into account other facts related with the memory, such as the trashing phenomenon described in Section 7.2, 'Memory background'.

8. Third extension: Modeling the network

In the previous extensions we had been working in some aspects that had to be taken into account in the internal behavior of one machine (or one VM). Those resources (CPU and memory) are directly related to the behavior of one machine and its execution of the assigned jobs. Taking those extensions into account we can assure that we will be more accurate simulating the behavior oh the machines and we will be able to reproduce the behavior of works of wide range heterogeneity . But there are some aspects that were not taken into account in EEFSimV2 and are also important. In particular, we refer to the aspects that are external to the machine, the most important one being the network connecting the different hosts that form the cloud system.

The characteristics of the network have a direct impact into the cloud system performance. As it is a distributed system it is directly related with the network where it is executed because the network is the infrastructure that provides the message exchange (in this specific case the VM migrations). There are many different aspects that participate in the behavior of the network: the topology, the protocol used for exchange the data, the bandwidth, the infrastructure supporting it,.... As in the previous sections, the key is to begin modeling the main factors for make a representation of the real behavior. If in the future more accuracy is needed it can be extended.

The are two main goals that we are following with all the extensions that we are implementing. The first one is to provide more intelligence to the scheduler for making its decisions more intelligent. That means that the more information that it can take into account, the better decisions it will make. So taking into account some aspects related to the network is fundamental. For instance, if on must to migrate one virtual machine from one host to another, one has to take into account the time that the migration will take depending on the destination host and that is directly related to the network bandwidth between the two hosts. The second one is to offer a lot of polymorphism in the systems that will be simulated. In EEFsimV2 the definition of the cloud system were very limited. There were just one data center that contains all the hosts specified in the '.ini' configuration file.

In the next sections we will begin describing how we decided to model the network, which means that we will describe which of the characteristics of a network are took into account and how do they affect to the behavior of the simulator. Then we will describe how it was implemented.

8.1 Network background

A network is a collection of computers and devices connected by communications channels that facilitates communications among users and allows users to share resources with other users. Networks may be classified according to a wide variety of characteristics. These are some samples of the different categorizations that we can do from one network depending on its characteristics:

Connection method

Computer networks can be classified according to the hardware and software technology that is used to interconnect the individual devices in the network, such as optical fiber, Ethernet, Wireless LAN, etc...

Ethernet uses physical wiring to connect devices. Frequently deployed devices include hubs, switches, bridges and/or routers. Wireless LAN technology is designed to connect devices without wiring. These devices use radio waves or infrared signals as a transmission medium.

The used connection method determines the limitations in respect to some parameters as the bandwidth of the network, the number of connections that it can support or the probability of loosing some of the packages that are transmitted.

Network topology

Computer networks may be classified according to the network topology upon which the network is based, such as bus network, star network, ring network, mesh network,.... Network topology is the coordination by which devices in the network are arranged in their logical relations to one another, independent of physical arrangement. Even if networked computers are physically placed in a linear arrangement and are connected to a hub, the network has a star topology, rather than a bus topology.

Network complexity

Network complexity is the number of nodes and alternative paths that exist within a network, as well as the variety of communication media, communications equipment, protocols, and hardware and software platforms found in the network.

A sample of a simple network could be a small LAN with no alternative paths, a single communication protocol, and identical hardware and software platforms across nodes. For other hand an enterprise-wide network that uses multiple communication media and communication protocols to interconnect geographically distributed networks with dissimilar hardware and software platforms would be classified as a complex network.

The complexity of the network affects directly to its transmission times. There are some factors like the distance between the nodes that compound the network that determine the elapsed time for transmitting the data between to different points.

8.2 Network modeling

In EEFSimV2 there was only possible configuration of the cloud system: it consisted of only one data center formed by the number of hosts that were defined in the '.ini' configuration file. That means that we could not specify more than one data center. Furthermore, the bandwidth of the infrastructure that connect all the hosts was not taken it into account. The only part where it was simulated was in the specification of the load of the migration of one virtual machine. That means that the migration workload of the machines were measured in a real system so it was measured under a specific network. But if we want to take the network characteristics into account we need to provide a way to specify it and then make vary the workload of the migrations depending on the velocity of the network.

As it is a first step to provide a network specification we will make it in a simple way: we will just take into account the main factor related with the velocity of the transmission of a network, its bandwidth. We will assume that the bandwidth for all the hosts of one data center will be unique and we will specify the bandwidth of the connections between data centers. Taking these facts into account we can specify our cloud system as one can see in Figure 8.1. It is important to highlight

that we will not take into account the occupation of the network which means that the bandwidth will not be penalized depending one how many data are being transmitted. That will be contemplated as a future work.

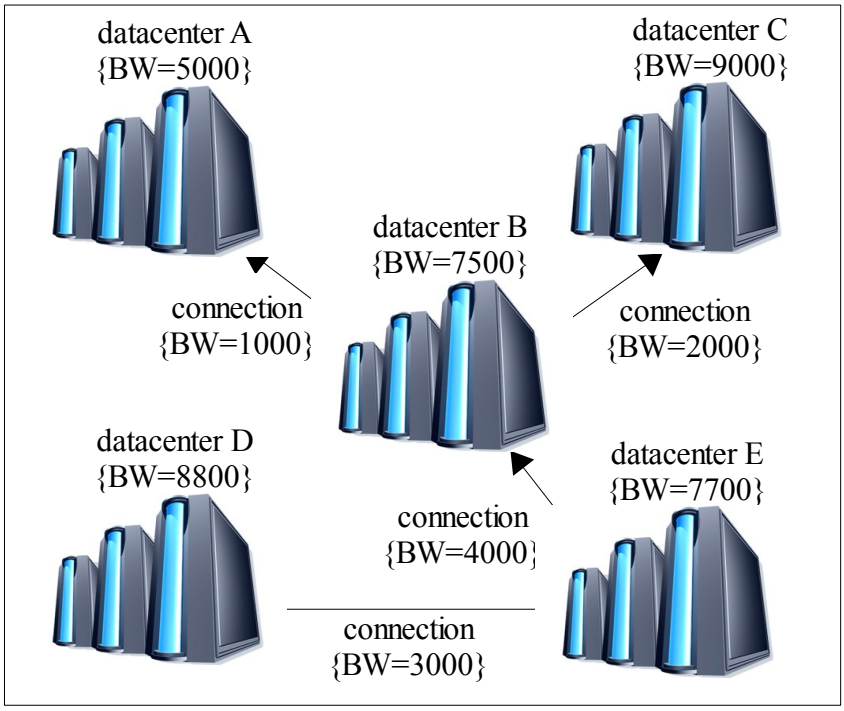


Figure 8.1 – Network sample

To provide one way to defining more than one data center it is necessary to split the hosts of the simulation in the different data centers that perform the simulation. That means that from now on we will need to specify the data center where each host is located. The bandwidth between the different data centers will be needed too.

Apart from analyzing which data are needed to model the network in our cloud system is important to analyze which components of our simulator will be affected by this new information and which will be the better way to design and implement the needed modifications to conserve the modules of the simulator with cohesive and scalable structure.

In the particular situation of modeling the network there are two components that need to know the characteristics of the network: the bandwidth of every data center and the bandwidth in the connections between data centers. These two components are the simulator and the scheduler (which is external to the simulator). The reasons why both of them need that information are clear. The Scheduler need to know which is the cost (the time needed) to migrate one VM from one host to another for taking better decisions. The simulator needs to know the bandwidth of the network for adjusting the migration time.

Once we notice that both components needs to know the same information there were three different possibilities of doing that: to make that both components get the information, to make that the scheduler get the information and communicate it to the simulator or to make that the simulator get the information and communicate it to the scheduler. We decide to do it in the third way: to

make that the simulator get the information and communicate it to the scheduler. There is an important reason that decided us to make that decision. If in a future work we wish to take into account the occupation of the network for computing the effective bandwidth, it could easily be done by the simulator because it is the entity that knows how many jobs are being transmitted at the same time.

When we have decided all that things there is just one more detail to analyze: how will be that data transmitted from the simulator to the scheduler? which means, what structure should we use to transmit it? We decided to do it by one matrix that has all the bandwidths between one host to another. With that information, the scheduler can predict the cost that implies migrating one VM from one host to another.

8.2.1 Network modelation sample

The best way to understand the whole network model is by presenting one sample. To begin with we will show one sample of how we specify the information about the bandwidth of every data center and the bandwidth of the connections between the data centers. The sample of specification showed in Figure 8.2 would be the correspondent one to the Figure 8.1.

```
<dataCenter name="A" bandwidth=”5000”>
        <host num=”1” ... />
</dataCenter>
<dataCenter name="B" bandwidth=”7500”>
        <host num=”1” ... />
</dataCenter>
<dataCenter name="C" bandwidth=”9000”>
        <host num=”1” ... />
</dataCenter>
<dataCenter name="D" bandwidth=”8800”>
        <host num=”1” ... />
</dataCenter>
<dataCenter name="E" bandwidth=7700>
        <host num=”1” ... />
</dataCenter>
<connection from="A" to="B" bandwidth=”1000” bidirectional=”1” />
<connection from="B" to="C" bandwidth=”2000” bidirectional=”1” />
<connection from="D" to="E" bandwidth=”3000” bidirectional=”0” />
<connection from="E" to="B" bandwidth=”4000” bidirectional=”0” />
```

Figure 8.2 – Network sample (XML content)

As it can be observed we specify this information in a standard XML format. In that sample we just keep the information related with the specification of the different data centers and the network bandwidth. In Section 9, 'Fourth extension: Using xml format', there is more information about which information will be presented in that format from now on.

Taking that specification into account we generate the matrix of bandwidth between hosts specified in Table 8.1.

From/To	host 0 (Data Center A)	host 1 (Data Center B)	host 2 (Data Center C)	host 3 (Data Center D)	host 4 (Data Center E)
host 0 (Data Center A)	5000	1000	1000	0	0
host 1 (Data Center B)	1000	7500	2000	0	0
host 2 (Data Center C)	1000	2000	9000	0	0
host 3 (Data Center D)	1000	3000	2000	8800	3000
host 4 (Data Center E)	1000	4000	2000	0	7700

Table 8.1 – Sample of required data for modelating the network

8.3 Network implementation

In this section we will describe the new files that implement the network model just presented, and the files formats that were previously used in EEFSimV2 but have suffered some change. In this case we implement one new class called `DataCenterMapper` and we modified the `PlugableScheduler` module.

`DataCenterMapper`

This class is the responsible for parsing the network specifications and generate the matrix used to communicate the bandwidth between hosts to the scheduler. It may seem that having to fill one matrix with a size of the square of the number of total hosts (the sum of the hosts of all the datacenters) will affect to the time elapsed by the simulator but this matrix has to be calculated just once during all the execution. Apart from filling the matrix it implements one method to communicate it to the scheduler.

Due to the fact that the information related with the network does not depend on the context all the attributes and methods of that class are specified as static which means that they do not vary from one instance to another. In fact, what really means declaring one method/attribute as static is that it is a method/attribute of the class and not from a particular instance of that class. To understand that explanation is important to have some background about how the object oriented programing paradigm works. For more information, see [17], and for more information on the implications of declaring some method/variable as static, refer to [18].

The Table 8.2 specifies the attributes and methods of this class:

Structs		
DataCenter		
This structure is used to represent a data center. Their attributes are defined below.		
name	***kind***	***description***
`position`	`int`	Represents the position of the data center in the matrix `connectionLatency`.
`offset`	`int`	Represents the number of hosts that has been parsed before that data center.
`numHost`	`int`	Represents the number of hosts of that data center.
`bandwidth`	`int`	Represents the bandwidth of the data center.
Attributes		
name	***kind***	***description***
`connectionLatency`	`static int *`	Pointer to the matrix that have the connection latencies between the different data centers.
`computerTransmisionLatency`	`static int *`	Pointer to the matrix that have the connection latencies between the different hosts.
`idDC`	`static list<<string,DataCenter>>`	List of the different data centers that compound the network.
`numDCs`	`static int`	Number of data centers that compound the network.
`numHosts`	`static int`	Number of hosts that compound the network.
Methods		
header		***description***
`private static int getIdentificador``(string name)`		Returns the identifier of the data center with the same name as the specified `name`.
`private static void printCTL();`		Prints the data of the `computerTransmisionLatency` matrix.
`private static void printCL();`		Prints the data of the `connectionLatency` matrix.
`public static void calculateLatencys``(const string fileName);`		Computes the transmission latencies of all the hosts parsing the file specified by the `fileName` attribute.
`public static int getConexionVelocity``(string hostOrig,string hostDesti);`		It returns the connection latency between the hosts with name `hostOrig` and `hostDesti`.

Table 8.2 – Methods and attributes from Memory class

The core of that class is the method `calculateLatencys` which is the responsible of computing the matrix `computerTransmisionLatency`. This method use all the variables of that class and initializes it. The first thing to do by this method is calculating the number of datacenters and hosts (set in the `numDCs` and `numHosts` variables respectively) that specify the network configuration file (which have a format similar to the one specified in Figure 8.1,'Network sample'). This is important

for initializing the size of the matrices `connectionLatency` and `computerTransmisionLatency`. Apart from calculating these values we take advantage of the fact that we are parsing the network configuration file to declare one `DataCenter` structure for each data center which will be used for filling the `connectionLatency` and `computerTransmisionLatency` matrices.

Once we have all the `DataCenter` structures in the `idDC` list we use it for filling the `connectionLatency` matrix. What this method do is to fill first the matrix with the bandwith among the different datacenters and the create the matrix with the bandwidth between hosts (`computerTransmisionLatency`) from the first one. For calculating all the bandwidths between datacenters we first take into account the internal bandwidth of one data center. Then we fill the direct connections, which means the connections that are specified in the network configuration file and then we calculate the connections to third datacenters, which means the connections that pass trough one ore more datacenters. For filling the connections that pass trough one or more datacenters we keep the connection with the minimum value. For example, if one data center A is connected to another data center B and the data center B is connected to another data center C, the connection bandwidth from A to C will be the minimum between A to B and B to C.

Once we had filled all the `connectionLatency` positions, we create the `computerTransmisionLatency` that corresponds to it. It is important to remark that here is where we use the parameters `offset` and `numHost` from the `DataCenter` struct for calculating the position of the hosts that represent the data center.

`PlugableScheduler`

This module needs one special modification, the addition of these parameters:

```
string hostConfigXml;
string hostNetworkXml;
```

These parameters are used for setting the path of the host and network configuration files. As it refers to the fourth extension they will be described with more detail in the next section. For now its enough to understand that the `hostNetworkXml` contains the path of the file that specifies the data related with the network (as the one specified in Section 6.2.1 'Network modelation sample').

Apart from adding these parameters to the module, we add the call to the function `calculateLatencys` from the DataCenterMapper and we add the following method:

Method	
header	***description***
`void SendBandwithMatrix();`	This method sends the bandwidth matrix to the external scheduler.

Table 8.3 – Added method to PlugableScheduler

which is used to send the matrix `computerTransmisionLatency` to the scheduler.

8.4 Future work

At this moment we only allow the definition of different network topologies which include the interconnection of different data centers and calculate the bandwidth between of the connection between the hosts but we do not apply this bandwidths anywhere. As a short term future work we have to make that the migration time of one VM varies depending on the connection bandwith.

On the other hand, as a long term future work we can consider the occupation of the network to update the effective bandwidth of the network or we can take into account other facts as the used protocol or the characteristics of the different nodes that perform the transmission of the transmitted data.

9. Fourth extension: Using xml format

As experience demonstrates and I learned in my software engineering courses, the major part of the life cycle for a software application is its maintenance. This is the reason why one of the major objectives of the informatics degree is to provide good practices and methodologies to design maintainable software applications. Creating a software applications with cohesion, extensible, and easy to maintain is quite difficult.

In the particular development of this simulator there are a lot of factors that intervene in its execution. There are a scheduler which is external to the simulator and the simulator are compound by several modules and auxiliary classes. Apart from that, there are a lot of files are used to configure it and specify some characteristics of the modules that are being used in the simulator as the workload.

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
    with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies,
    an evil sorceress, and her own childhood to become queen
    of the world.</description>
  </book>
</catalog>
```

Figure 8.3 – XML sample

The complexity of the simulator makes it difficult to understand every line of its code. It is important to understand that it is the result of many different people that worked on its development during many years in the university. So to make it with a cohesive structure is one of the keys to make easy to understand how does it behaves to the people that are going to work with it in the future. To contribute to this goal, we decided to transform some of the files used for configuring and specifying some data to the simulator into the xml format.

One of the major benefits of the xml format is that it have a hierarchy structure that makes it easy to understand which concepts are more generic and which are more specific. Apart from that, it have a autodescriptive format because every field or attribute has its specific name. See an example of an xml file in Figure 8.3. As it can be appreciated in this sample, by taking a look at the document we can form a clear idea of what is being specified by that document.

9.1 Xml background

XML (Extensible Markup Language) is a set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification produced by the W3C [24]. XML's design goals emphasize simplicity, generality, and usability. The characters which make up an XML document are divided into markup and content. Markup and content may be distinguished by the application of simple syntactic rules. All strings which constitute markup either begin with the character "<" and end with a ">". Strings of characters which are not markup are content. One tag is a markup construct that begins with "<" and ends with ">".

Tags come in three flavors: start-tags, for example <section>, end-tags, for example </section>, and empty-element tags, for example <line-break/>. An element is a logical component of a document which either begins with a start-tag and ends with a matching end-tag, or consists only of an empty-element tag. The characters between the start- and end-tags, if any, are the element's content, and may contain markup, including other elements, which are called child elements. An example of an element is <Greeting>Hello, world.</Greeting> Another is <line-break/>. An attribute is a markup construct that consists on a name/value pair that exists within a start-tag or empty-element tag. For example, if we consider this element: , the element img has two attributes, src and alt.

There are several freely-available XML editors that allows to the user to create XML files easily and robustly. Without this, it would be counterproductive for a user having to create syntactically correct XML, and it would be simpler and easier to use a flat file because XML is designed for being easily readable by machines, not easily readable by humans.

These are the main characteristics of that format:

Structured format

which means that we can define exactly how the data is to be arranged, organized and expressed within the file. When we are given a file, we can validate that it conforms to a specific structure, prior to importing the data. As we know the structure of the file in advance, we know what it contains and how to process each item. Prior to XML, the only structure in a text file was positional , we knew the bit of text after the fourth comma should be a date of birth, and we had no way to validate whether it was a date of birth, or even a date, or whether it was in day/month/year or month/day/year order.

Described format

which means that within the text file, every item of data has a name that is both human, and machine, readable as well as being uniquely identifiable. We can open these files, read their contents and understand the data they contain, without having to refer back to another document to find out what the text after the fourth comma represents (and was that comma a separator, or part of the text of the second item?). Similarly, we can edit these documents with a fairly high level of confidence that we’re making the correct changes.

It can easily describe hierarchical data and the relationships between data

If we want to import and export a list of authors, with their names, addresses and the books they've written, deciding on a reasonable format for a csv file is by no means straightforward. Using XML, we can define what an Author item is and that it has a name, address and multiple Book items. We can also define what a Book item it is and that it has a title, a publisher and an ISBN. The hierarchy and relationships are a natural consequence of the definition.

9.2 Used library

The xml format has become very popular, and used in a wide range of different areas for different purposes. As a consequence there are a lot of different packages and toolkits that provide functionalities for creating and parsing its contents.

The standard of the xml format has evolved and nowadays it can offer a complex structure. But our purpose is to use it in a simple way, just for some specifications, so we do not need a complex toolkit for parsing our specification files. For this reason we decide to use simple library that could be used by C++ with the basic functions of parsing one xml file.

After discarding some options we found TinyXML [21]. TinyXML is a simple, small, C++ XML parser that can be easily integrated into other programs. TinyXML is designed to be easy and fast to learn. It is two headers and four cpp files.

9.3 Configuration files

The are five different kind of files that describe each of the next characteristics of the simulation: the different kind of hosts that form the cloud, the number and type of every host that form the cloud, the energy consumption of each host depending of the number of cpus that it uses, the workload that are going to be executed by the cloud during the simulation, and the load of every job that is described in the workload (because it can vary along its execution).

After applying the xml format to some of the specification files, all the data related with the specifications of the different types of machines are provided by the `hostConfig.xml` and all the data related with the network that compound the cloud system are specified in the `hostNetwork.xml` file. That makes sense because the `hostConfig.xml` trends to be unique for all the different experiments because the specification of the hosts does not vary while the `hostNetwork.xml` will vary for every experimentation because it describes the data centers that form the network that is being simulated.

One of the benefits of compacting all the information related with the host specifications (energy consumption and different components like cpu or memory) in the `hostConfig.xml` file is that it is presented in a cohesive way. Furthermore we have the benefits of the xml format which include a lot of semantic about what are we defining. In Figure 9.3 we show a sample file of how would be filled the `hostConfig.xml` file in our case of study.

```
<config name="pctinet">
        <memory id="mem" size="4096" latency="3.703" />
        <processor num="4" frequency="3000"/>
        <power>
                <item power="idle" wattage="230"/>
                <item power="boot" wattage="275"/>
                <item power="0" wattage="238.9" />
                <item power="100" wattage="267.8" />
                <item power="300" wattage="302.5" />
                <item power="400" wattage="317.9" />
        </power>
        <load type="bootHost" length="60" />
        <load type="boot" length="30" />
        <load type="migrationIn" length="40" />
        <load type="migrationOut" length="20" />
        <load type="checkpoint" length="30" />
</config>

<config name="pcperot">
        <memory id="mem" size="4096" latency="3.703" />
        <processor num="4" frequency="3000"/>
        <power>
                <item power="idle" wattage="230"/>
                <item power="boot" wattage="275"/>
                <item power="0" wattage="238.9" />
                <item power="100" wattage="267.8" />
                <item power="300" wattage="302.5" />
                <item power="400" wattage="317.9" />
        </power>
        <load type="bootHost" length="60" />
        <load type="boot" length="41" />
        <load type="migrationIn" length="60" />
        <load type="migrationOut" length="30" />
        <load type="checkpoint" length="30" />
</config>
```

Figure 9.1 – hostConfig.xml sample

With that format we can appreciate all the relevant data to each different types of machines of our cloud system without having to analyze the code of the simulator for understanding what do they mean.

In the `hostNetwork.xml` file we specify the network of datacenters that form our cloud system. It is important to highlight that defining different datacenters gives much more polymorphism to the simulation tests that can be performed by the simulator. In our case of study the `hostNetwork.xml` file would be specified as in Figure 8.2. As it can be observed we avoid having to specify one line per host as happened when we use EEFSimV2 (see Section 5.3.2, 'PlugableScheduler').

As we explain in this section we just adopted the xml format fore some files. The `.load` files keep its plain text format. The reason why we do that is because these files are generated by one script that its being executed during the execution of the correspondent job. It is easier to generate that kind of files by a text plain format. Apart from that, the performance of the simulator can decrease if we abuse from the xml format. This format is useful when its important to provide a lot of information about what is being specified by a simple view. This is the reason why it use to be used for the configuration files. But parsing this kind of documents is more difficult and take more time

than do it with a text plain format.

9.4 Implementation

We add two different methods for parsing the contents of the xml files. One of them is the `calculateLatencys(const string fileName)` method from the class `DataCenterMapper` which is described in Section 8.4.1, 'DataCenterMapper'. The other one is an added function to the `PlugableScheduler` class.

PlugableScheduler

This is the new method:

Method	
header	***description***
`void loadHostsConfigXml()`	That function parses the content of the host configuration and network configuration files and create the `PC` modules that represent the hosts that are specified. Apart from that it communicate s to the external scheduler the characteristics of the hosts that compound the network of the current simulation.

Table 9.1 – Added method to PlugableScheduler

It is important to highlight that before this function was implemented there was another one called `loadHostsCofig()` which do essentially the same but using the files with the previous format (plain text).

9.5 Future work

There is not much future work related with that extension. As we explained there is neither need nor sense in using this format for the `.load` files. So we adopt this format for the necessary files for the moment. Apart from that, the content of the files will vary every time that we want to add some new specification to the file. In fact, one of the reasons that made us adopt this new format is making it easier to add new features to its content.

10. Evaluation and validation of the simulator

When we explained the goals of that project we assumed one implicit goal: improve the results of the simulation. We develop the extension for improving the heterogeneity of the tests. But as we had said this work was done extending one simulator called EEFSimV2. The main goal of our simulator was to produce accurate results related with the energy consumption of the simulated cloud. We have developed some extensions and it is important to prove that we do not corrupt the previous results. What is more, we have to obtain even more accuracy, because we are modeling more details of the real system.

Once EEFSimV2 was finished their developers performed some validations for comparing the results obtained by the simulator with the real, measured, ones. The main objective of that simulator was to measure the energy consumption of the cloud depending on the usage of its machines. For that validation they needed to execute a real workload in a real cloud system and measure the energy consumption during this execution. Then they specified the same workload to the simulator and evaluated the difference between those ones and the real ones. As a result they obtain the data specified at Table 10.1, with 1163 rows (one for every second of the elapsed time during the execution of the workload).

In Figure 10.1 one sees these results in graphical form: one can see the evolution of both the real and simulated results during the execution of the workload. The average absolute error from the results obtained by the EEFSimV2 was **6,23 W** which was a **2,15%** of relative error.

Once we finished our extensions we run another time the same workload in the simulator EEFSimV3, which is the one that implements all the work developed within this degree project. The results were collected in a new table which was analogue to Table 10.1 but with the new obtained data. A representation of that table is shown in Table 10.2.

In Figure 10.2 one can see the evolution of both the real and simulated results during the execution of the workload. The average absolute error from the results obtained by EEFSimV3 was **4,99 W** which means a **1,72%** of relative error.

We add a third figure, Figure 10.3, for facilitate the comparison between the results obtained by EEFSimV2 and EEFSimV3.

As a conclusion we want to highlight that we succeeded with our proposals, because with EEFSimV3 we decrease the absolute error by an average of 1,24W which is a **0,43%** of relative error over the real measures, and a **19,9%** improvement on accuracy with respect to EEFSimV2.

Time	Real energy consumption	Simulated energy consumption	Absolute error	Relative error
0	239,2 W	242,85 W	3,65	0,02
1	239,2 W	242,85 W	3,65	0,02
2	239,15 W	242,85 W	3,7	0,02
		. . .		
1163	240 W	262,42 W	22,42	0,09

Table 10.1 – Results of the validation of the EEFSimV2

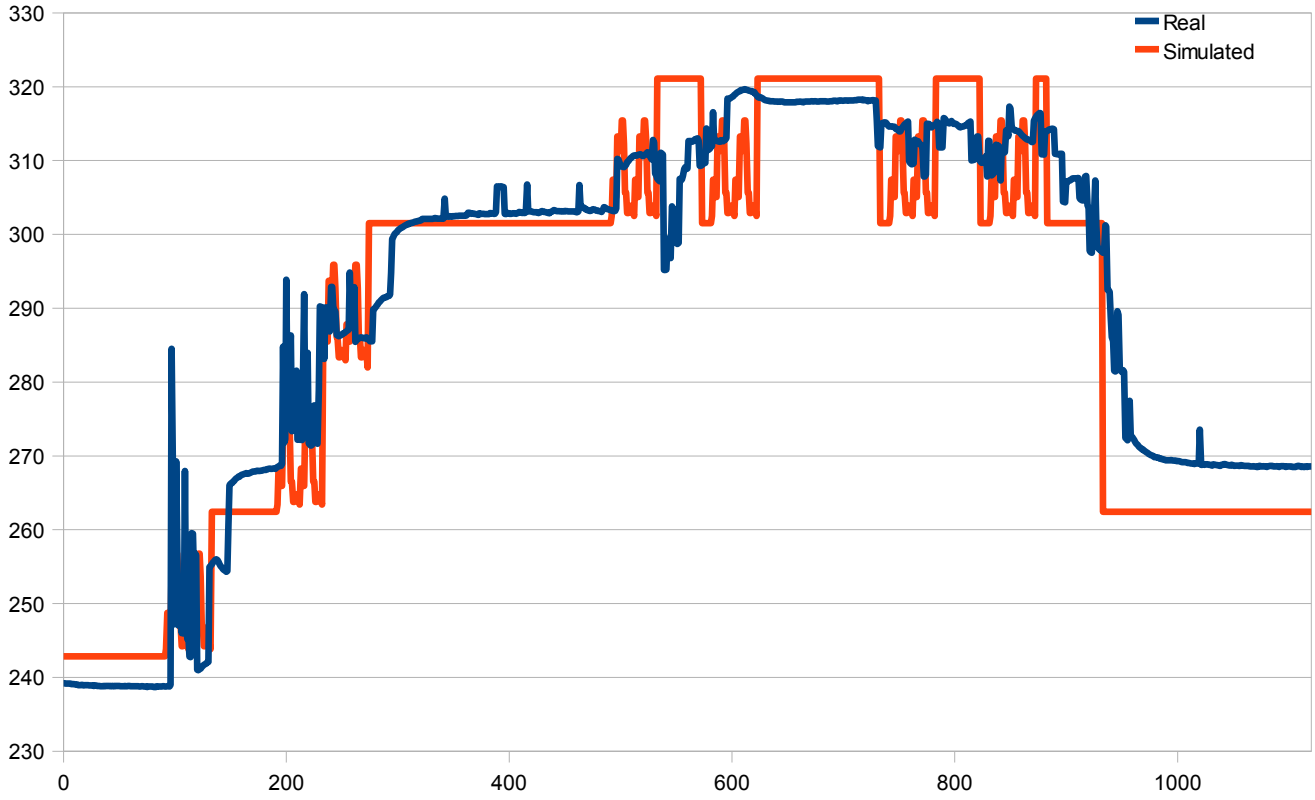


Figure 10.1 – Real and simulated (by EEFSimV2) results related with the energy consumption of the cloud (in Watts)

Time	Real energy consumption	Simulated energy consumption	Absolute error	Relative error
0	239,2 W	243,24 W	4,04	0,02
1	239,2 W	243,24 W	4,04	0,02
2	239,15 W	243,24 W	4,04	0,02
		. . .		
1163	240 W	243,24 W	3,24	0,02

Table 10.2 – Results of the validation of the EEFSimV3

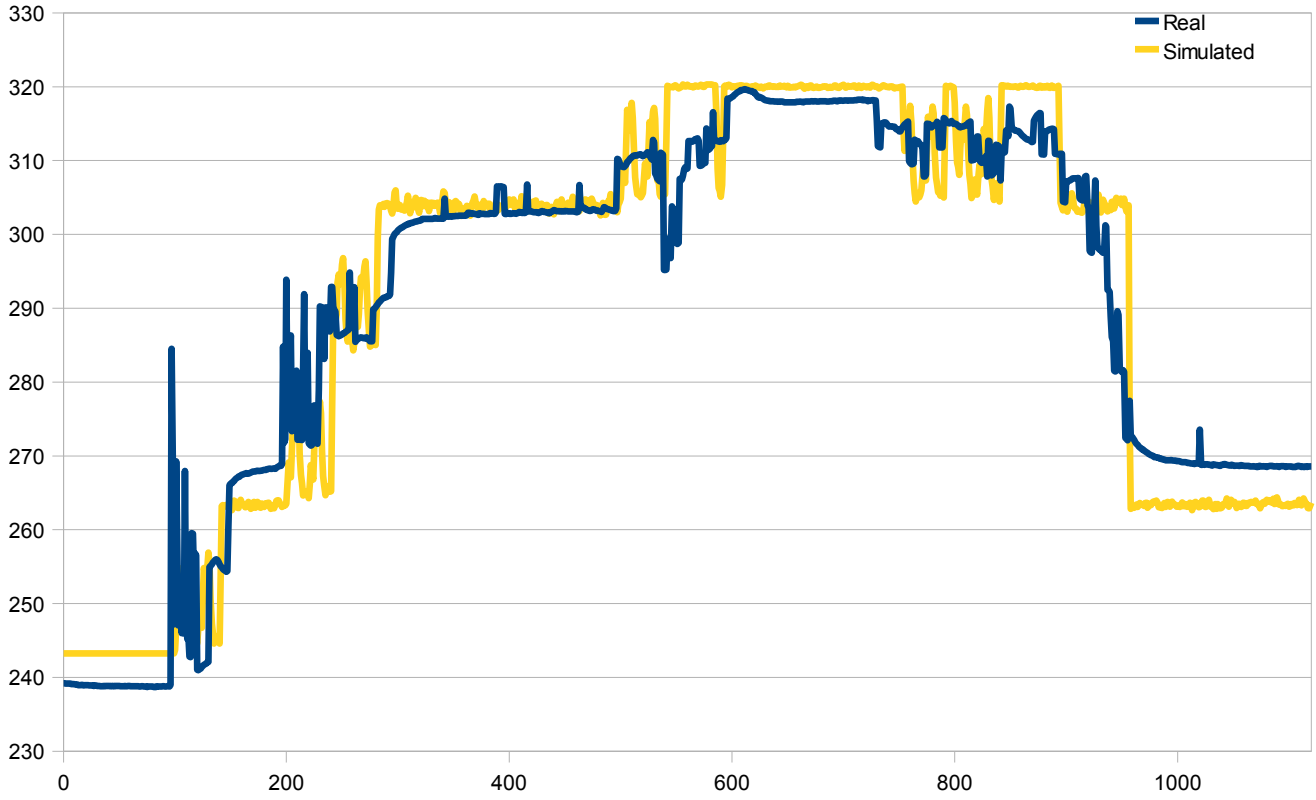


Figure 10.2 – Real and simulated (by EEFSimV3) results related with the energy consumption of the cloud (in Watts)

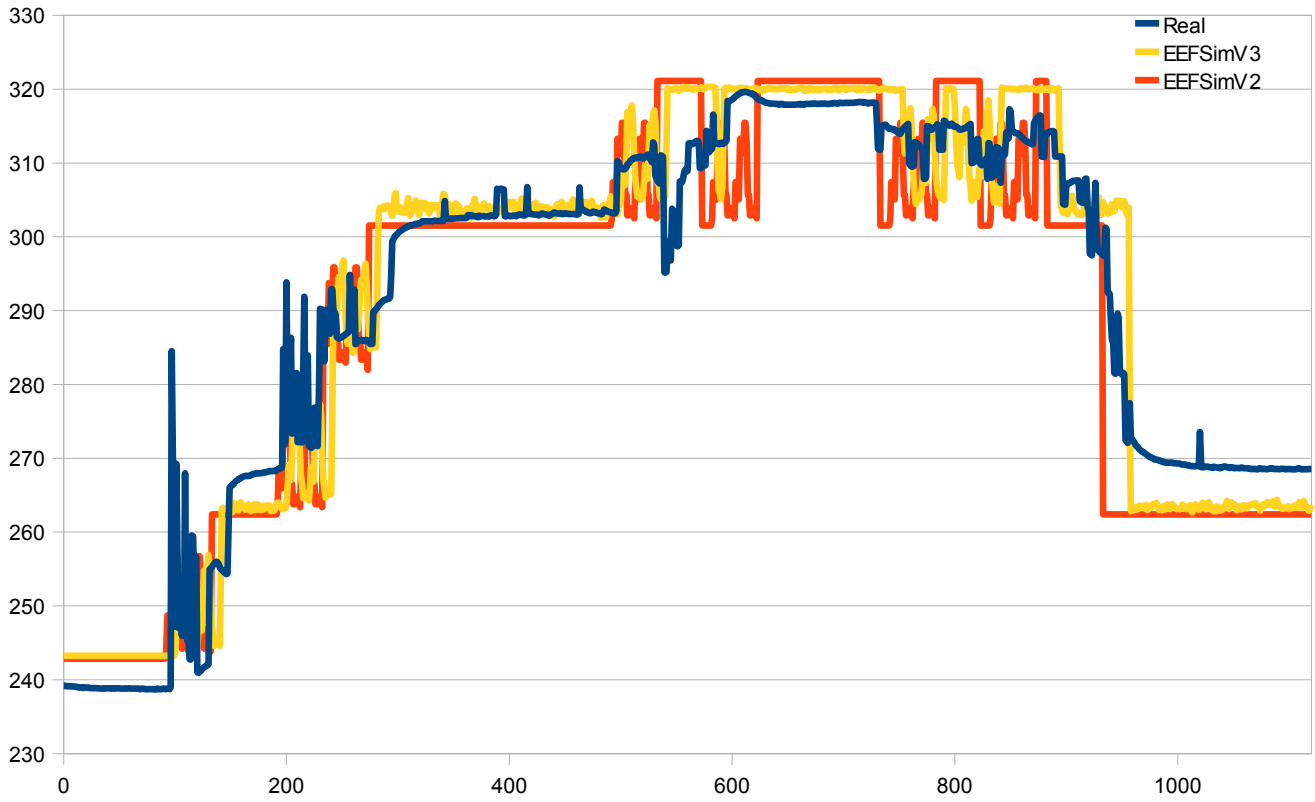


Figure 10.3 – Real and both of simulated (by EEFSimV2 and EEFSimV3) results related with the energy consumption of the cloud (in Watts)

11. Schedule of the project

Estimating software-product size and of software-project time and resource needs is undoubtedly hard. This is specially true of projects with a component of research, such as the present one, where it is easy to develop a lot of work to experimenting, trying innovative solutions that may or not wor, ideas that must be reconsidered later, etc. Scheduling aims to predict the future, and it has to consider many uncertainties and assumptions. While it is not possible to know with certainty how long a project will take, there are techniques that can increase your likelihood of being close. If you are close in your planning and estimating, you can manage the project to achieve the schedule by accelerating some efforts or modifying approaches to meet required deadlines.

One key ingredient in the scheduling process is experience in the project area, another is experience with scheduling in general. In every industry area there will be a body of knowledge that associates the accomplishment of known work efforts with a time duration. The most of the projects that we realize during the career are scheduled by the teachers so we (the pupils) are not used to schedule our projects. It is important to highlight that there are some subjects as *Projecte de Enginyeria del Software i Bases de Dades (PESBD)* that explain some techniques that serve as a guidance but as we said there are many facts that interact in a schedule of one project, this is the reason why it is important to have experience doing that kind of practice.

In this section we will describe the temporal distribution of the work of this project in respect to the time we dedicate to develop it and how does it differs from the initial schedule of the project. Apart from that we will analyze which human resources had been needed to develop it for make an estimation of the cost that it would have in a professional context.

11.1 Original schedule

The real schedule of the project has some similarities with the predicted one but it is important to highlight that we suffer important diversions. In our case we did not take into account the difficulty of modeling the memory usage of one program and above all, how it affects to its execution time. To begin with, lets take a view at the original schedule of the project.

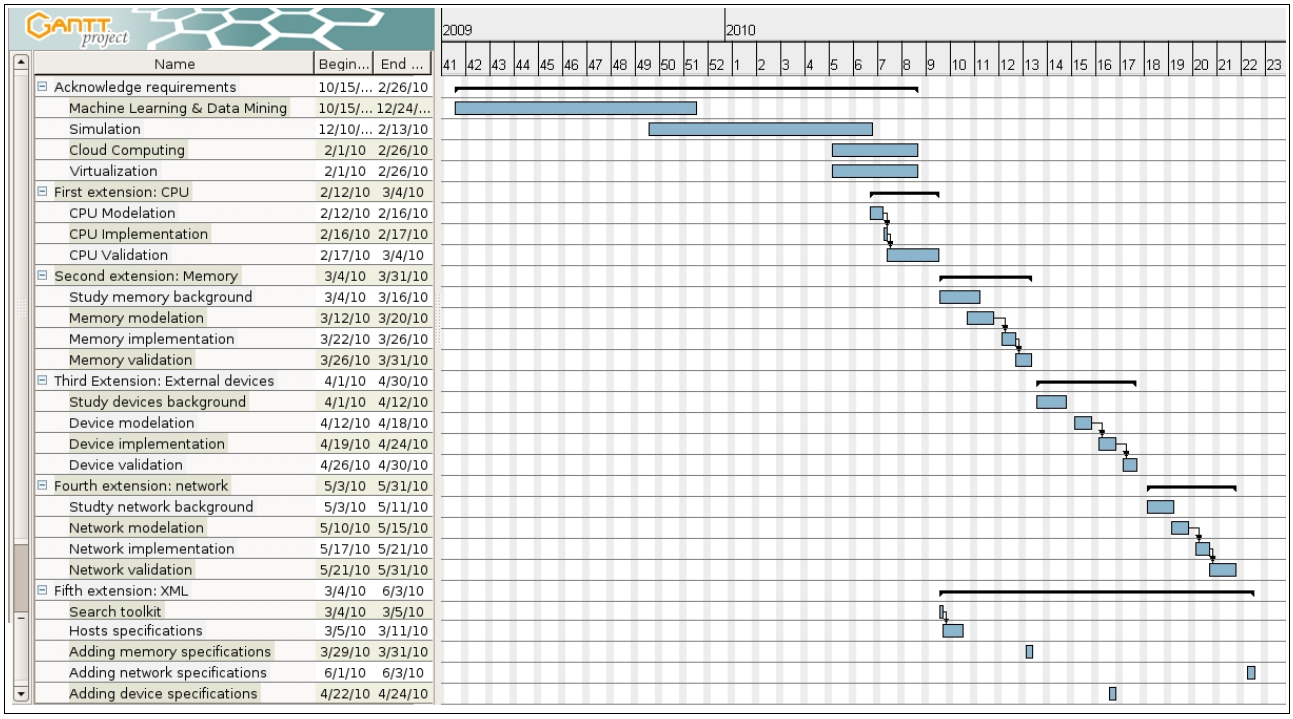


Figure 11.1 – Gantt chart of the original schedule of the project

Comparing the original prevision (Figure 11.1) one can observe that we figured that we would spend approximately one month to model memory and the rest of extensions. That made us think that we could perform more extensions than the ones that we did. The extension that caused the gap between the real schedule of the project and the predicted one is the memory modeling (see Figure 11.3). It turned out to be much more difficult than we predicted. That was because there where many factors that interact in the elapsed time by memory accesses, many more than we thought. That things could happen when you walk into a path with uncertainty.

11.2 Distribution of the workload

Although this project had been developed during the last year the workload has not been uniform. If we take into account the workload we can distinguish three different phases: the background acquisition, the effective work and the documentation of the work.

Learning, reading, and background acquisition

In the next gantt chart you can appreciate the main tasks and milestones of that period.

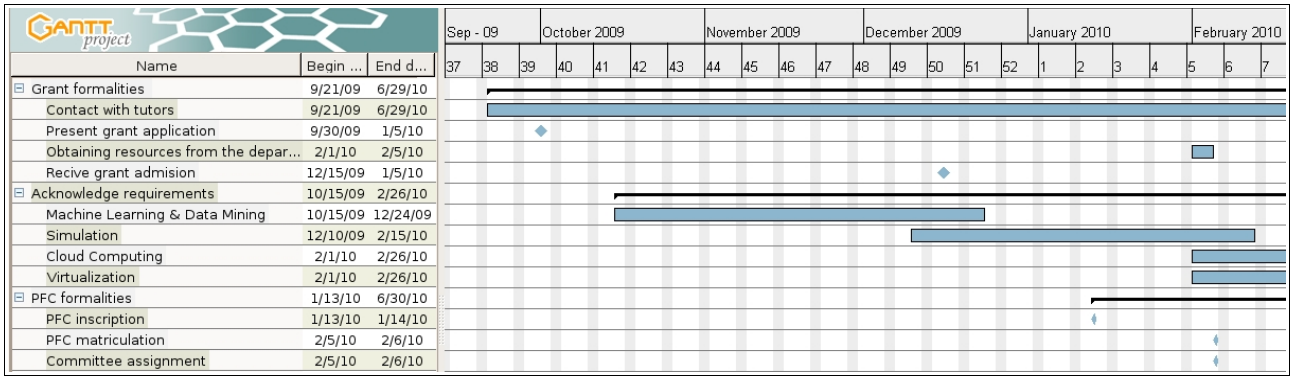


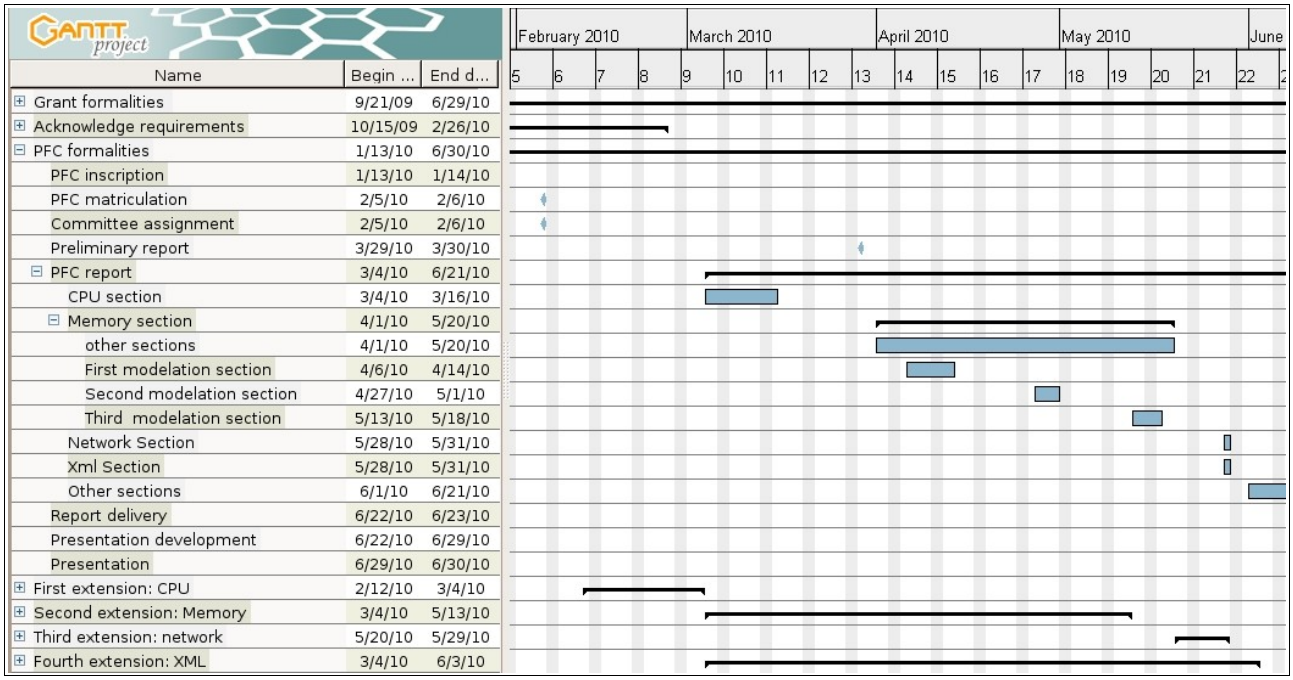
Figure 11.2 – Gantt chart with the tasks performed during the formalization of the grant

All began around 21th of September but I did not begin to read some papers and work with my background acquisition until October. In November I rode some of the main sections of [8] for begin to acquire the background related with Machine Learning and Data Mining. Then on 11th of November I received the [5] to begin to get in touch with the main purposes of the Josep Lluís Berral PhD thesis. From that moment we begin to have periodic meetings and we thing which could be my contribution to the project.

During December and January I was really busy because I had exams from the last subjects of my degree but I still dedicate some time to get in touch with the machine learning and I began to analyze which are the main artifacts that interact in the test-bed of the Josep Lluís PhD thesis and its code. After rule out some other alternatives we (me, Josep Lluís, and our advisors) decide that the best way to help Josep Lluis with my contribution would be performing some extensions to the simulator that he used (EEFSimV2) to endow it with more heterogeneity in respect to the tests that it could simulate and become a good test-bed for the Josep Lluís pretensions. On February I learned how does virtualization and cloud system work and I also analyze the code of EEFSimV2.

The effective work

After I understood all the important concepts related with the machine learning, cloud computing, virtualization and simulating (with special effort on how does Omnet++ works) I was ready to begin with the development of the different extensions of the simulator. Apart from that I made an effort to document all the information related with these extensions just when I finished with them because it is the moment when I had all the concepts and details of the extension in mind. Leaving the documentation to the end of the project would be an error because there could be a lack of memory respect to all the modeling, implementation and validation details. The next Gantt chart, Figure 11.2, shows the main tasks of that period.



Name	Begin ...	End d...
Grant formalities	9/21/09	6/29/10
Acknowledge requirements	10/15/09	2/26/10
PFC formalities	1/13/10	6/30/10
PFC inscription	1/13/10	1/14/10
PFC matriculation	2/5/10	2/6/10
Committee assignment	2/5/10	2/6/10
Preliminary report	3/29/10	3/30/10
PFC report	3/4/10	6/21/10
CPU section	3/4/10	3/16/10
Memory section	4/1/10	5/20/10
other sections	4/1/10	5/20/10
First modelation section	4/6/10	4/14/10
Second modelation section	4/27/10	5/1/10
Third modelation section	5/13/10	5/18/10
Network Section	5/28/10	5/31/10
Xml Section	5/28/10	5/31/10
Other sections	6/1/10	6/21/10
Report delivery	6/22/10	6/23/10
Presentation development	6/22/10	6/29/10
Presentation	6/29/10	6/30/10
First extension: CPU	2/12/10	3/4/10
Second extension: Memory	3/4/10	5/13/10
Third extension: network	5/20/10	5/29/10
Fourth extension: XML	3/4/10	6/3/10

Figure 11.3 – Gantt chart of the effective work tasks

In comparison to the previous phases (background acquisition) the workload increased. This is where the research begins and where we found the most of the problems that we did not predict from the beginning. This period comprises the most of the time of this project. It goes from 12th of February till 3th of June.

As it can be observed, we developed all the extensions sequentially except the use of the xml format. This is because the use of the xml format concerns the specification of the properties of the hosts so we made an effort to implement it from the beginning for taking as much advantage as possible and we have to extend it every time we perform another extension to the simulator, because it affects at the way the characteristics of the host and network should be defined.

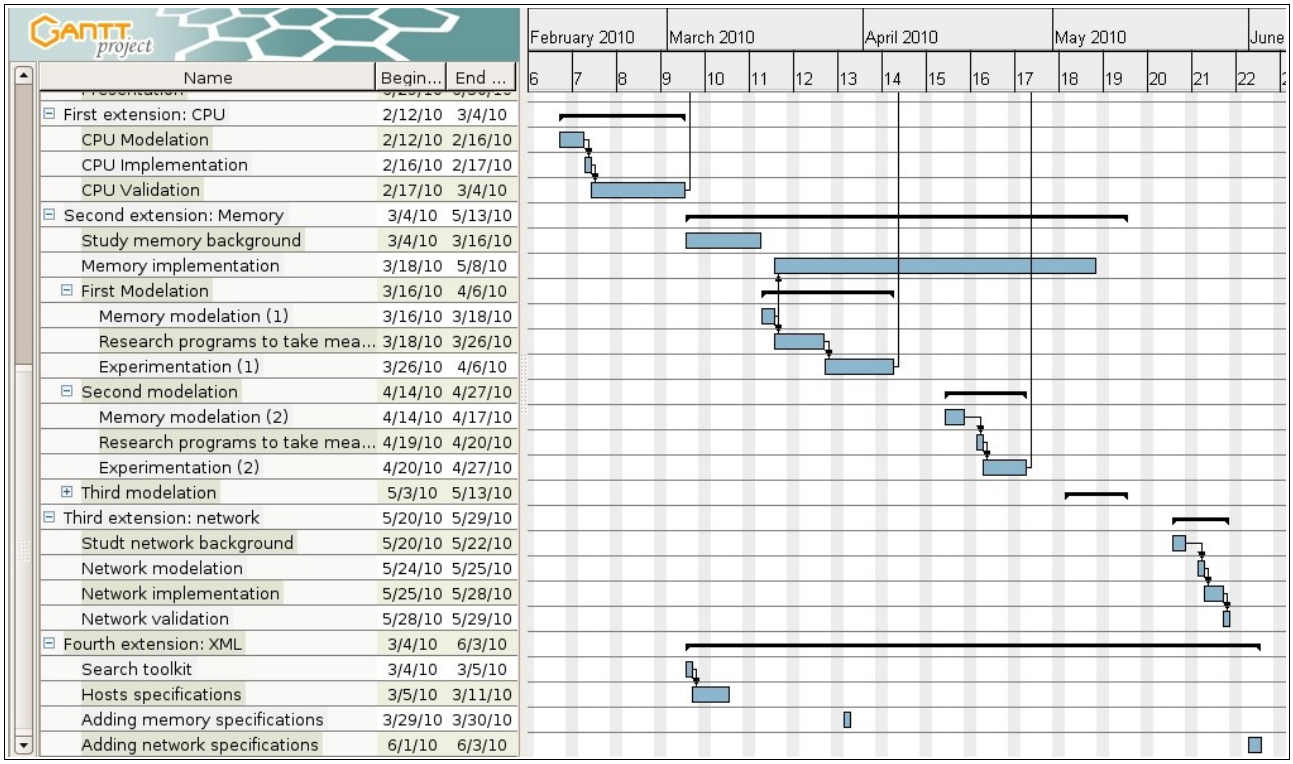


Figure 11.4 – Gantt chart of the tasks involved with the extensions

Another thing which we can highlight is that we followed the same methodology to perform each extension. First we had to study all the features related with the extension we want to model and then we had to model, implement, and validate it. In the case of the memory, we had to make an special effort to find some programs to obtain the measures we need to make the experimentation.

The documentation of the project

This is the shortest period. It just goes from 3th to 29th of June. Documenting all the work that has been developed within that project is not easy and it takes much time. Apart from considering all the contents of the memory and the presentation of the project, a lot of time is spent giving the appropriate format to those documents. Taking this into account and considering that we had less time than we wanted due to the prolongation of the previous phase it is easy to understand that the workload of that scenario had been frenetic. Of course it had been less heavy than it could be if we had not document all the work developed during the realization of the extensions.

The main tasks of that period are represented in the next gantt chart, Figure 11.4.

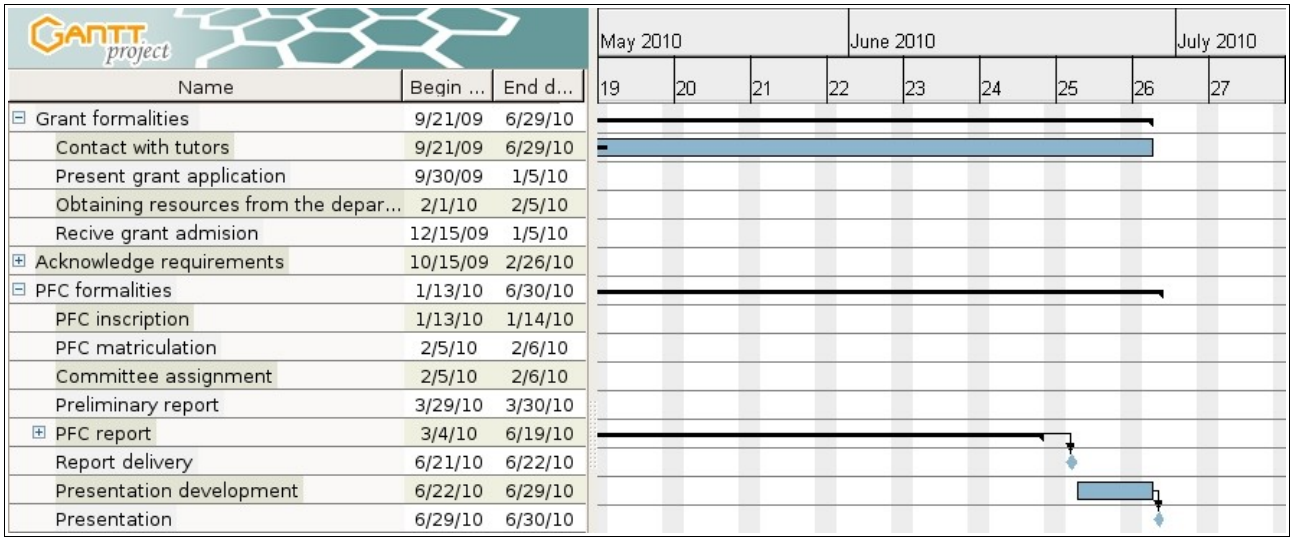


Figure 11.5 – Gantt chart of the tasks involved last steps of the project

11.3 Cost of the project

This section is aimed at analyzing all the costs related with the development of the project. We will distinguish between the costs related with human resources and the ones related with hardware, software, and maintenance.

11.3.1 Human resources

There are many people that have contributed to the development of the project. Of course, the workload has not been the same for all the contributors. The majority of the contributors supervise the developed tasks and give some guidance about which way to follow. It is important to highlight the work of Josep Lluís Berral who dedicated a lot of time and effort to this project. Other people that participate with the development of the simulator or the scheduler also dedicate some of its time to give some information and references to better understand the behavior of the different entities that interact in this project.

In the next table we specify the amount of time that everyone dedicate to this project per month. Apart from that, we specify the cost per month of all the participants and the total cost. We also specify the amount of months that every person contribute to the project because it could vary depending on the person. Of course, there are some people that contribute during all project realization but there are others that just participate when we begin to implement the extensions.

Nº of people	Rol	Dedicated time (h/month)	Period of collaboration (months)	Total hours (hours)	Cost (€ per hour)	Total cost
2	Department collaborators	8 h/m	6 months	48 h	8 €/h	768 €
1	Tutor of the project	15 h/m	9 months	135 h	8 €/h	1080 €
1	Designer and developer (grant beneficiary)	120 h/m	9 months	1080 h	2,5 €/h	2700 €
Total.....						**4548 €**

Table 11.1 – Human resources costs

As one can see from the table, it is estimated that the author of this degree project has dedicated about 120*9 = 1080 hours to it.

11.3.2 Hardware, software and resource maintenance

In this section we will specify the costs related with the needed hardware and the maintenance of the resources that the department of LSI has provided for developing the project. These resources consist on a laptop and a desk in one of the department offices, which implies internet connection, access to some printers and illumination.

Hardware			
Concept		**Total cost**	**Cost related with my usage**
Laptop		1080 €	108 €
Experimentation machines		5000 €	500 €
Maintenance			
Concept	**Cost (per month)**	**Period of utilization**	**Total cost**
Department office	100 €	9 months	900 €
Total cost.....			**1580 €**

Table 11.2 – Hardware and maintenance costs

The amount of money that is needed to buy the machines used for doing experimentations can be considered relatively big but it is important to highlight that those machines are used for a lot of experimentations done by the department, not just this project. We can consider that one machine can be used for five years. We use it for 6 months. Therefore we can consider the 10% of its total cost for computing the cost related with my usage.

One of the major benefits of this project is that it does not have any costs derived from the software usage. We have used only open software, that is: linux as operating system, Oment++ and C++ to implement the code, and other open-source, free programs as valgrind to do some experimentations.

Adding up all partial costs, we estimate the cost of this project to be approximately 6128€.

12. Conclusion

Once the project has ended let us draw some conclusions on the major problems encountered, the skills acquired, and the lessons learned.

12.1 Review/summary of the requirements and their achievement

1. **Improve the representation of the CPU.**

   This requirement has been completely carried out. With the new representation of the CPU we provide a cohesive and easy scalable representation of that resource. Apart from that we recompute the execution time of one tasks depending on the CPU features of the hosts were it is executed. As a result we provide more realism and accuracy at the results obtained by the simulation.

2. **Take into account the memory usage of one task.**

   This requirement has not been accomplished at 100%. We provide a new representation for the memory resources which will provide more cohesion and will bring more scalability to the simulator. Furthermore we spent a lot of time on research trying to model the behavior of the execution of one task depending on its memory usage. We did not succeed with this objective because there are more facts to be taken into account that the ones we predicted. There is the need to do more work with it and it will be contemplated as a future work. It is important to highlight that it does not mean that this work had not been useful. Without that work we could not know the real workload of that requirement.

3. **Take into account the usage of external devices of one task.**

   The excess time spent on the previous requirement affected mainly this requirement, and with less impact, the rest of the project. We had to skip this requirement due to the delay we suffered in the schedule of the project. It would be contemplated as a future work. Personally, it was a good lesson to learn about how unexpected problems can appear in the project and how rescheduling is often necessary.

4. **Improve the network representation.**

   Thus requirement had been practically completely carried out. We provide a new way to specify different data centers that interact with each other which improves the heterogeneity of the networks that could be specified to the simulator. Apart from that, we specify the bandwidths of all the network communications for taking them into account while the simulation should calculate the elapsed time of performing the migration of some virtual machine from one host to another (as a short term future work).

5. **Improve the way that some specifications are provided to the simulator.**

   This requirement had been completely carried out. We provide a new way, XML files, to specify the features of the machines that form the network. Apart from that, we improve the specification of some of the resources that form a host. By representing all that information in a XML format we take advantage of some of the advantages of that format, such as facilities to describe hierarchical data and the relationships between data and its self-described structure that provides more semantic respect to what is being defined.

12.2 Technical conclusion

In this project I improved my knowledge related to simulation. In the computer science degree there is no compulsory subject dealing simulation. Simulation is an underlying concept in many fields. Obviously, apart from improving my knowledge about that issue I also get experience working with one particular, widely used, simulation framework, Omnet++. Apart from that, some of the behaviors and mechanism of that framework are really interesting and frequently used in other environments as the execution focused on events, the message exchange and the schedule of events along the execution time.

Apart from that I also improved my skills in C++ programing . I am more used to java and it is important to be familiarized with as many programing languages as possible. Still, due to the fact that both languages have the same programing paradigm (they are object oriented) they have a lot of things in common.

Another important lesson that I learned is that scheduling one project is very difficult and I think that nowadays I would do it better than I did this time. As I mention in Section 11, 'Schedule of the project', one key ingredient in the scheduling process is experience in the project area and experience with scheduling in general. I also get more experience designing software with intention to provide a cohesive, modular and extensible structure to every “piece” of software.

12.3 Personal conclusion

Let me now highlight some important skills developed or strengthened in this project, which are not so much technical but also really important. From my point of view we learn a lot technical concepts in the degree and realizing the final project of the degree is an opportunity to improve other skills and feel free to try some of the contents or practices that we do not learn during the realization of the degree. For me it has been an honor to have the opportunity of collaborating with two of the departments of the university and get in touch with how does research work. I thing that researching is very interesting and it is a continuous challenge that provides a lot of satisfaction. To get what you are looking for without any previous work that could serve as a reference is really rewarding.

This experience taught me some good lessons. As I worked with a multidisciplinary group I had to make an effort to understand things about which I had not heard about and I also have to make myself understand. These skills could be categorized as communication skills. Furthermore I had to

learn a lot of things by my own (with help from my advisors that always gave me the best references). Furthermore I also had to interpret a lot of code developed by other people. This is not easy at all.

Finally I want to highlight that I do an extra effort to write this report in English. I am totally agree with the tendency of the new context for European education of integrating English (or some other international language) into curricula. I think that learning new languages and living in different countries with different people from all over the world is very interesting and enrich your personality. But living abroad reclaimed an effort that, for personal reasons, I could not do at this time. I wanted to write this report in English in order to make my contribution.

Bibliography

Papers and books

[1] Josep Ll. Berral, Ricard Gavaldà, Jordi Torres. *Imporved Self-Management of Complex Cloud Systems Applying Machine Learning.* Ph.D. Thesis Poposal June 2009.

[2] Ferran Julià, Jordi Torres, Jordi Guitart. *Towards Cloud Energy Efficient Resource Scheduling.* January 2010.

[3] Ramon Nou and Jordi Torres. *Aplicación de la simulación en tiempo real para mejorar la calidad de servicio del Middleware.* October 2008.

[4] Ramon Nou. *Energy Efficiency: A case study. March 2009*

[5] Josep Ll. Berral1, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà and Jordi Torres. *Towards energy-aware scheduling in data centers using machine learning.*

[6] I. Goiri, F. Julia and J. Torres. *Elastic management of Tasks in Virtualizad Environments*. In Proccedings of the XX Jornadas de Paralelismo (JP2009), pages 671-676, 2009

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield. *Xen and the art of virtualization*. *Proceedings of the nineteenth ACM symposium on Operating systems principles,* pages 164-177, 2003.

[8] Ian H. Witten & Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition).* ISBN: 0-12-088407-0. 2005.

[9] Joshua Ruggiero, *Measuring Cache and Memory Latency and CPU to Memory Bandwidth,* December 2008. link: http://download.intel.com/design/intarch/papers/321074.pdf

Websites

[10] Parsec official website, http://pcl.cs.ucla.edu/projects/parsec/. (February 2010)

[11] Ns2: http://www.isi.edu/nsnam/ns/ (February 2010)

[12] Omnet++: http://www.omnetpp.org/ (February 2010)

[13] Xen Wiki Credit Scheduler. http://wiki.xensource.com/xenwiki/CreditScheduler (March 2010)

[14] Valgrind: :http://valgrind.org/ (March 2010)

[15] Cachegrind: http://valgrind.org/docs/manual/cg-manual.html (March 2010)

[16] LMBench: http://www.bitmover.com/lmbench/ (March 2010)

[17] Obect Orientation: http://java.sun.com/docs/books/tutorial/java/concepts/ (April 2010)

[18] Static keyword: http://msdn.microsoft.com/en-us/library/s1sb61xd(VS.80).aspx (April 2010)

[19] COTSon: http://sites.google.com/site/hplabscotson/ (February 2010)

[20] emotive: http://emotivecloud.net/ (February 2010)

[21] TinyXml: http://www.grinninglizard.com/tinyxmldocs/index.html (March 2010)

[22] ACPI: http://www.acpi.info/ (March 2010)

[23] http://www.bitmover.com/lmbench/lmbench3.tar.gz (March 2010)

[24] W3C: http://www.w3.org/ (May 2010)

[25] http://www4.ncsu.edu/~hp/simulation.pdf (February 2010)

Annex A: Source code of the implemented benchmarks

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main (int argc, char *argv[])
{

	static int size = 20000 * 20000;
	int vegades = 1;
	static int acc = 1;
	int partition= (int) size / acc;
	int position, offset,aux = 0,i = 0,j = 0,k = 0;

	int* matriu = (int *) malloc (size * sizeof(int));


	srand ( time(NULL) );

	for (i=0; i < size; i++)
	{
		matriu[size] = aux++;
	}


	for (k = 0; k < vegades; k++)
	{
		for (i=0; i < size; i++)
		{
		position= rand() % size;
			offset = position % partition;


			switch(acc){
				case(1):
					aux++;
					break;
				case(2):
					aux++;
					if(position<partition) aux++;
					if(partition<=position<2*partition) aux++;
					break;
				case(3):
					aux++;
					if(position<partition) aux++;
					if(partition<=position<2*partition) aux++;
					if(2*partition<=position<3*partition) aux++;
					break;
				case(4):
					matriu[position]++;
					if(position<partition) aux++;
					if(partition<=position<2*partition) aux++;
					if(2*partition<=position<3*partition) aux++;
					if(3*partition<=position<4*partition) aux++;
					break;
```

```
                        case(5):
                                matriu[position]++;
                                if(position<partition) aux++;
                                if(partition<=position<2*partition) aux++;
                                if(2*partition<=position<3*partition) aux++;
                                if(3*partition<=position<4*partition) aux++;
                                if(acc<=1 && 4*partition<=position<5*partition) aux++;
                                break;
                        }
                }
        }



        free(matriu);
}
```

Source code of the implemented benchmark without memory accesses.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main (int argc, char *argv[]) {
        static int size = 20000 * 20000;
        int vegades = 1;
        static int acc = 1;
        int partition= (int) size / acc;
        int position, offset,aux = 0,i = 0,j = 0,k = 0;

        int* matriu = (int *) malloc (size * sizeof(int));


        srand ( time(NULL) );

        for (i=0; i < size; i++) {
          matriu[size] = aux++;
        }

        for (k = 0; k < vegades; k++){
          for (i=0; i < size; i++){
            position= rand() % size;
            offset = position % partition;
            matriu[position]++;
            switch(acc){
              case(1):
                  matriu[position]++;
                  break;
               case(2):
                  matriu[position]++;
                  if(position<partition) matriu[offset]++;
                  if(partition<=position<2*partition) matriu[partition+offset]++;
                  break;
              case(3):
                  matriu[position]++;
                  if(position<partition) matriu[offset]++;
                  if(partition<=position<2*partition) matriu[partition+offset]++;
                  if(2*partition<=position<3*partition) matriu[2*partition+offset]++;
                  break;
              case(4):
                  matriu[position]++;
                  if(position<partition) matriu[offset]++;
                  if(partition<=position<2*partition) matriu[partition+offset]++;
                  if(2*partition<=position<3*partition) matriu[2*partition+offset]++;
```

```
                    if(3*partition<=position<4*partition) matriu[3*partition+offset]++;
                    break;
                case(5):
                    matriu[position]++;
                    if(position<partition) matriu[offset]++;
                    if(partition<=position<2*partition) matriu[partition+offset]++;
                    if(2*partition<=position<3*partition) matriu[2*partition+offset]++;
                    if(3*partition<=position<4*partition) matriu[3*partition+offset]++;
                    if(acc<=1 && 4*partition<=position<5*partition) matriu[4*partition+offset]++;
                    break;
                }
            }
        }
    free(matriu);
}
```

Source code of the implemented benchmark without memory accesses.