MASTER THESIS

TITLE: Design and implementation of a simulator to explore cooperation in distributed environments

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Davide Vega D'Aurelio

DIRECTOR: Roc Messeguer Pallarès

DATE: June 17 th 2010

Títol : Disseny i implementació d'un simulador per explorar la cooperació en entorns distribuïts

Autor: Davide Vega D'Aurelio

Director: Roc Messeguer Pallarès

Data: 17 de juny de 2010

Resum

La manca de recursos computacionals dels ordinadors personals, afegit al increment de requeriments cada vegada més patent en el software actual, està demanant urgentment a una redefinició dels paradigmes de computació que hi havia fins ara. Les xarxes de computació distribuïda son una de les solucions que actualment s'estan valorant per resoldre el problema de la manca de recursos *hardware*.

No obstant, la aplicació d'aquestes solucions es veu aturada per la manca de coneixement sobre el seu funcionament i la necessitat tecnològica de resoldre reptes que la recerca i la innovació encara no han descobert com encarar. Un dels problemes pendents és com generar la suficient confiança entre els usuaris i per tant, assegurar un cert nivell de cooperació dins de la xarxa, o el que és el mateix: una col·laboració bidireccional.

Aquest projecte de màster pretén explorar els diferents mecanismes basats en incentius i topologies per promoure la cooperació utilitzant com a eina un simulador capaç de valorar aquests indicadors sobre múltiples topologies de xarxa, aplicacions i estratègies sobre sistemes GRID distribuïts. Per tant, una primera part està enfocada a l'estudi de la cooperació e identificar quins paràmetres poden portar-la a terme, i quins no influeixen. El segon objectiu és la creació d'una eina prou modular i potent com per extreure suficients conclusions de la seva utilització sobre diferents escenaris i, finalment, una ultima part centrada en l'anàlisi de resultats i la creació d'un protocol de negociació verificable per la compartició de recursos en xarxes distribuïdes.

Title : Design and implementation of a simulator to explore cooperation in distributed environments

Author: Davide Vega D'Aurelio

Director: Roc Messeguer Pallarès

Date: June 17, 2010

Overview

The lack of computational resources on personal computers, in addition to evident increasingly of requirements in the current software, needs a imperative redefinition of computing paradigms that existed until now. The distributed computing networks are the currently solutions that are being evaluated to solve the problem with the lack of hardware resources.

However, the implementation of these solutions is stopped by the lack of knowledge about its functioning and the need to resolve technological problems that research and innovation have not yet discovered how. One of the unresolved problems is how to generate sufficient trust among users in order to ensure a certain level of cooperation inside the network, or what is the same: how to generate a two-way collaboration.

This master thesis seeks to explore various mechanisms based on incentives and topologies to promote cooperation using a simulator as a tool capable of evaluating cooperation indicators over multiple network topologies, applications and strategies on distributed systems as a grids. Therefore, the first part focuses on the study of cooperation and identify which parameters are wearing out, and which do not influence. The second objective is create a tool as powerful and modular enough to extract sufficient conclusions before use it on different scenarios and, finally, the last part is focused on analyzing results and creating a verifiable negotiation protocol for sharing resources in distributed networks.

A la meva família, per la seva paciència i suport.

A Roc Masseguer, professor, tutor i amic per la seva inestimable ajuda i la confiança dipositada en la realització d'aquest treball.

A Felix Freitag i René Brunner per la seva ajuda amb les eines del grup i els seus comentaris al llarg del projecte.

A tots els companys de i2CAT que cada dia m'aguanten i, especialment als meus confidents: Xavi Calvo, Javi López, Dani Rodriguez i Fran Rillo.

A Xavier Miguélez, per els seus consells i ànims.

Però sobretot, a la persona que segurament ha patit aquest projecte més que jo: Milena D'Aurelio, gran amiga i millor mare.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Due to the strong increase in demand of computers power for applications and the gradual recession that has the electronics industry, more and more resources for personal computers (and scientists) are insufficient. In a scenario like this, one of possible solutions being proposed is the resource sharing across distributed networks. However, there are still many issues to be resolved to be these applications a reality, as for example the information management, the resource sharing or the incentive for cooperation.

Research groups of universities have long been carrying out projects to solve some of the problems discussed. VRIMS is a new project in this way that aims to provide a first fully integrated solution that solves most of them. But, before start their design and implementation is mandatory known how to solve some related problems. The aim of this project is to generate a simulation tool capable of running various games, strategies and topologies in a fully distributed environment in order to know what is the relationship between these elements and cooperation with long term objective to apply the learned lessons to VRIMS project.

The first chapter focuses on a brief description of the past projects that justify the creation of the VRIMS project and the importance of the realization of a simulator to know how to encourage cooperation in distributed networks. Also, a concrete objectives of this Master Thesis will be listed in two groups: knowledge related and simulator.

Second chapter explain all the theoretical concepts that appears along this project in order to use a common language to explain their development and fix some ideas used. Third chapter explains all tools involved on the project development and statistical results study. Also, some concrete concepts related to topologies creation tools are resumed in order to understand the results.

Then in the fourth chapter shows the process of designing and implementing the proposed simulator, with emphasis on particular aspects related to the proposed game, the strategies are to try and some other features. Some UML diagrams are used to simplify explanations of software modules and simulator layers.

The fifth chapter shows the possibilities of the simulator from a practical approach. After explaining as is have verified its operation are discussed at same time that explained strategies and topologies used restatements and discussing some lessons learned. As a global result of the project, a new negotiation protocol that performs the simulators functioning will be also explained.

Last chapter resumes all the conclusions achieved through the results commented on last chapters and some future considerations for VRIMS project.

CHAPTER 1. OBJECTIVE AND PROPOSAL

LRM (*Local Resource Manager* [1]) is a PhD project of the Distributed Systems Group of the UPC (*Technical University of Catalonia*) finished last year with the objective to demonstrate that is possible create a software system to share resources from virtual machines between network nodes. The application is able to manage local computer CPUs capabilities and status, and share the information over a self-managed xeon OS. Other projects of DSG, as DMIS (*Distributed Market Information System* [2]) that "retrieves economic data in distributed markets in a decentralized manner to provide it to individual market participants with the objective to solve the challenge of a large-scalability by underlaying a peer-to-peer-system".

One of naturals evolution of this projects is the VRIMS (*Virtual Resource Infrastructure Management System*) proposal, a distributed p2p application to share physical resources from virtual and real machines placed in a cloud system as same manner that occurs over actual file sharing applications. From research point of view, this project implies a lot of unresolved problems related to software design, resource self management, distributed intelligence, large-scalability, cooperation mechanisms and negotiation protocols.

This master thesis is the first related project, with exceptions of LRM and DMIS, and aims to determine what relationship exists between the network overlay topology, the strategy of the nodes and methods of incentives, and how it affects the cooperation of whole system in a distributed network GRID. After solving this problem is expected to justify a negotiation protocol that allows exchange of information necessary.

1.1. Simulation process

Three ways exists to accomplish described objective: develop and test the real implementation, study the system from a mathematical model or develop a simulator in order to extract some conclusions. First solution is not implementable, since it is designed to solve a nonexistent software and involves a large investment in time without guarantees.

Second, is a good solution for well-known systems, but not in this case when the system is under definition process. Also, require a strong knowledge about mathematics, statistical process and non chaotic systems.

Third and last solution requires a extra time to achieve valid conclusions in order to have a valid set of simulations. But in contraposition have the advantage that developed tool can be reused in many future scenarios or projects to test different games. Also, as all simulators, change some algorithm or decision process (or input data) can be easily.

So, objective has been redefined as *develop a simulation tool to study the effect of topology and incentive methods over cooperation on distributed systems games*. Next section describes the concrete objectives.

1.2. Objectives

Objectives has been divided into direct objectives of this project, so its related to the study of cooperation, the conclusions achieved and the design and evaluation of the concrete application and a second set of objectives related to topic knowledge and learning process that have also been taken into account. Next sections lists these objectives in order to evaluate the project accomplishment.

1.2.1. Learning process objective

- Study GRID systems, known their architecture and functions of each part and understand and have a general idea of how they can be developed.
- Learn to use the tools of creation and manipulation of topologies BRITE, Otter and Pajek.
- Achieve a good knowledge about topologies creation, properties, classification and characterization. Concretely, differentiate between node placement and connectivity and the importance of each it over topology properties.
- Understand and compare different cooperation techniques, frameworks and algorithms with the objective to design a distributed game that can represent CPU sharing application described.

1.2.2. Main objectives

- Design a new game framework and two different strategies in order to compare the tit-for-tat algorithm with some else strategy on concrete game created.
- Generate at least two kind of different topologies for the purpose to compare the results with reference papers.
- Design and implement a scalable software application able to play a cooperation game. Simulator should be give interfaces to developers in order to implement whatever other game or strategy and test them. Also, they must accept different topologies from a configuration files and save the results secure.
- Extract statistics from simulations and analyze the results in order to differentiate where this results are influenced from topology and when from the game. From topology analysis is important extract the relation between nodes placement and connectivity with their results achieved and, from strategies analysis, extract a primality design of negotiation protocol for designed game.

CHAPTER 2. BASIC CONCEPTS

In this chapter is explained the basic concepts related to this work. Firstly, a brief explanation of grid paradigm and its relationship with p2p (*peer to peer*) networks is detailed. Basic grid components and their typical architecture are also described in order to have a overview of the project scenario. Next, basic network topologies are described, with particular emphasis on mesh scale-free models with special properties from a statistical and/or strategic point of view.

Finally, the relation between cooperation algorithms and middleware grid topologies has been explained in order to extract some basic conclusions which will be verified in our simulations and then, be implemented in our proposal. Concretely, "FreeRiding" user behavior and its effect over p2p networks will be discussed. At same time, *tit for tat*, and other cooperation games, has been studied and its evolution over different kind of topologies described in order to maximize the cooperation of the nodes.

2.1. GRID

Since few years ago computer technologies and electronic industry have been substantially increasing computing power (processing speed, memory, etc) of personal computers, but not enough. Moore's law [3] [4] predicted that the pace at which technology advances every eighteen months we are able to double the number of transistors on each chip, also doubling the computational capabilities. But in many cases, technological developments of pasts years, have not focused solely on increasing the power of computers, but on efficiency, decreasing chipsets size or improving parallel computing capabilities; so Moore's prediction don't accomplishes yet. Anyway, applications continue requiring each time more computer power.

GRID computing is a technological evolution of personal computing, where the infrastructures, information, services and communications takes place in the network, and not locally. From the user point of view, the grid is like an opaque cloud to which is connected via the network (LAN or via Internet) to use one resource (another device on the network, file data, etc) or more of them together, creating a new complex, but virtual, service in the cloud. The system must be transparent to the user, so does not need or should know what happens inside.

However, as in the case of clouds, from within the grid is aware of its own architecture, its environment and conditions under which it is located. It is able therefore to configure the services that users need at every moment, intelligently responding to requests and reconfiguring some of them in real time. Thus, the responsibility of creating a robust and reliable system is the cloud, not the user.

Of course, the main aim of this kind of architecture is to provide a platform much more powerful than a single user could achieve individually to users, solving the lower computer technology progress problem. Distributed computing paradigm is a particular case of GRID. In this case a single program or process is divided into self contained parts, and processed in parallel by several different computers connected to each other through the

network. In the same way that occurs with parallel computing, the use of these techniques require the segmentation of the calculation processes and information, so a payload communication cost.

There are several important differences between distributed computing and parallel computing that is important to note:

- The process of decision making (scheduling, how code are divided, jobs control, transfer of messages) is delegated to a group of computers (the grid), rather than a single computer. This means that collaboration and cooperation decision is critic.
- Generally, the computer elements (processors, processing power, available memory, hard disk space, hard disk speed) are shared between other users and programs running at same time. Even it not well and a user has full access to various nodes, the composition of each of the computers on the network differs from the other.

So, in conclusion, the collaboration and cooperation between nodes make a more important role in distributed systems.

A particular case, as described previously (see chapter 1) are the *VRIMS*, a distributed computing grid that consists on a set of nodes connected globally, many times composed only by user nodes, as same manner that occurs on p2p networks [5]. So, all jobs related to grid management, resource sharing included, are performed by the edge nodes. This architecture reduces the infrastructure cost, increasing the complexity.

Next subsections describes the main characteristics of GRIDs and its architecture.

2.1.1. Characteristics

The grid has a number of requirements to be met to perform its function. Listed below are strictly related to the problem to be solved in this project: the study of cooperation through the topology and incentives.

2.1.1.1. Collaboration

“Collaboration is a recursive process where two or more people or organizations work together in an intersection of common goals by sharing knowledge, learning and building consensus” [6]. However, in a scenario as described, the objectives of the nodes are individual, rather than global and consensus is not mandatory. So, it is better to talk about cooperation, that is more related to reciprocity. “Cooperation is the process of working or acting together, which can be accomplished by both intentional and non-intentional agents” [7]. The difference is that cooperation is mutual, while the collaboration does not have to. This means that in a cooperative scenario, the nodes search the individual collaboration of other nodes at same time that expects a reciprocity for continue cooperating. But the main objective of the system is global cooperation, that is really different from individuals objective.

2.1.1.2. Aggregation

The grid and its nodes are responsible to compose complex services through attachment the individual nodes services or infrastructure. Messaging protocols may ensure that this composition can be done transparently to users.

2.1.1.3. Scalability

Scalability is the characteristics of a grid that ensures that aggregation begin be possible. Network topology, node composition and services deployed over their physical layer must be scalable without performance drawback. So, all of its components can be added, removed or changed on fly as efficient as possible.

2.1.1.4. Heterogeneity

Each node is composed of any combination of different hardware or software components. And each of this components, can be present a variety performance level or availability that vary with time. As a consequence, nodes needs a second set of protocols to locate this information and ask for a concrete resource availability.

2.1.1.5. Reconfigurability

While scalability refers to no performance reduction on configuration operations, reconfigurability refers to the ability of a grid to incorporate a new physical component, remove or change their composition/disposition intelligently when is required. To do this, nodes must know their status and that of their neighbors and act as consequence.

2.1.1.6. Security

Security can be described from different points of view, such as secure communication between nodes or software execution integrity, all its important and necessary. In our case, security refers to avoid “Free Riding effect”(See section 2.3.), or similar problems that allow a node out of non-cooperation benefited.

2.1.2. Architecture

The grid or peer to peer architecture are separated by domains of responsibility called layers (See figure 2.1). As occurs on other layered models such TCP/IP model [9] each layer provides services to layers above it and uses the services provided by below layers.

The lower layers are focused on hardware resources and capabilities, while as we go up into architecture the layers are more focused on grid responsibility and user services and

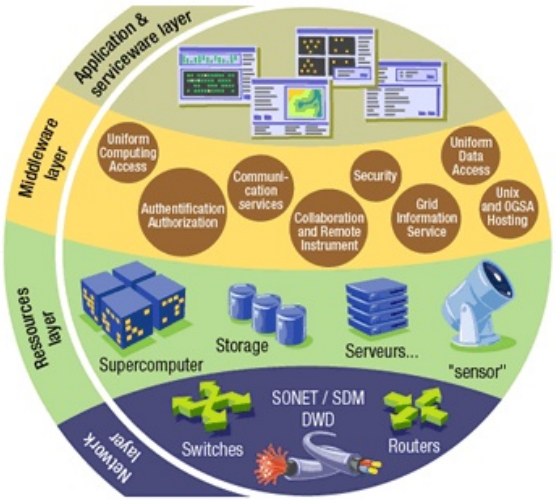

Figure 2.1: Layer schema of GRID architecture

operations. Upper layers are typically controlled by software components and users rather than hardware. This separation allows to change the physical hardware transparently to software or vice versa. Layers and its responsibilities are described next from bottom to top.

2.1.2.1. Network layer

Also called Physical Network Layer, it provides connectivity (both, logical and physical) to all resources into the architecture regardless if the physical resource is in use or not, or the concrete connectivity established by each application.

Some of the services that this layer offers to others are load traffic balancing, high-availability routing protocols and redundant/diverse configurable network paths [8].

2.1.2.2. Physical layer

This layer contains all the physical resources involved in the grid that not are strictly network components. Therefore in this group can be found from dummy resources such as hard drives or processors, up to more configurable and complex devices such as video transcoders or MCUs (*Multi Control Units*). Obviously, this part of the network also locates the data store architecture.

Some of services offered by this layer are data mirroring, replication, parallel processing, hardware backup and recovery, alerts and monitoring to signal a failure within the environment.

2.1.2.3. Middleware layer

The middleware layer is the most complex layer of grids architecture and typically is divided into four basic groups [8], responsible to manage the system:

- Security: provide the cryptographic architecture and tools to verify resources, user and applications identities. Also, is the responsible for providing security to the communication system.
- Connectivity: enable data exchange between the resources of the network layer. Include transportation, route and identification of entities that form the network.
- Resources management: provide the tools that enable the various elements (servers, storage, networks, etc.) to participate in a grid. Also implements the protocols responsible for the management of information (as grid directory) and negotiation between nodes (as data storage).
- Service management: on the one hand, enable the virtual services composition, planning, coordination and management of all the services over the middleware layer (as indexing service or dynamic grid software clustering). On other hand provides policy management and operation of automated control to ensure that all resources are always operational.

2.1.2.4. Application layer

Application and service layer contains all the software that can be executed by the edge nodes (users nodes).

As the highest layer of the infrastructure, should be include complete applications (for example p2p file sharing, a CDNS portal or scientific data-visualization grid), development toolkits to support the applications (lower layers interfaces) or development frameworks to easy deploy applications over the grid. Finally, the application layer often includes the so-called service-ware, which performs general management functions like tracking who is providing grid resources and who is using them.

2.2. GRID Topologies

In the previous chapter has defined the different layers of a grid and interaction between them. Assuming that network layers and physical resources layer are, in fact, at the same level of management, we can say that only exists three different layers in the grid. Firstly, the physical and real network topology defined by the emplacement and connectivity between each node on the network. Second, the overlay network provided and defined by each of the services. The last one, application network, is the logic connectivity used by the applications, that usually are the same as the overlay network provided by middleware layer, maybe adapted to concrete application requirements. Differentiation from overlay and application layer is not mandatory.

In order to simplify the explanation, from here terms of “network topology‘” or “topology‘” will be used indistinctly to refer to the overlay network topology, that we will assume identical to application-level network. If required, “network physical topology”, will be used to refer to lower topology network level.

Although is not required that the grid uses a particular network topology, the choice can provide a number of properties for overlay network interesting or necessary for some applications. In fact, exists networks without being aware of it, or at least it was not a condition that wanted to meet, end up having a fixed topology. Next part defines the elements of a topology and the most interesting configurations.

All topology depends of its two basic elements: nodes and links. A node is a connection point, that is enable to send, receive and forward information trough their links. Link is a channel or connection that enables the communication between two nodes. If the link is unidirectional (only one of two nodes can be send information to the other) the link is called also vertex, while if it is possible two-way communication (both can send and receive information) will be called arc. From topology point of view, the concrete characteristics of nodes and links are not relevant. Finally, the combination of this two elements creates a new concept: degree. The node’s degree represent the number of links that allows a node to send information to others, that is the number of vertex from which it can send information plus the number of arcs to which is connected.

2.2.1. Basic topologies

Nowadays, exists many kind of topologies and network structures, but the simplest is the single-level types (See figure 2.2). These are named single-types because their properties and statistic characteristics are shared between all the nodes, so not integrates other topologies inside.

Next, connectivity based topologies will be described.

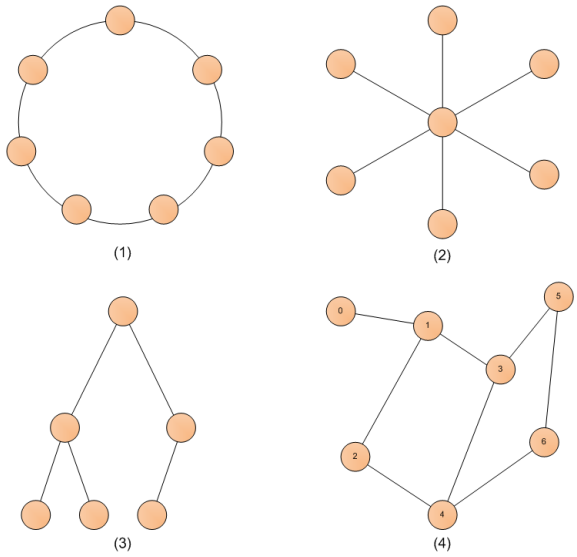

Figure 2.2: Basic topologies. **(1)** Ring. **(2)** Star. **(3)** Tree. **(4)** Mesh.

2.2.1.1. Ring topology

In this topology each node is connected only with two others, creating a closed circle. So, communication with other nodes depends directly of the distance in term of number of hops between them. Taking advantage of some techniques for reordering information, some implementations and protocols, such Chord [10], can reduce the search time to $O(logN)$ on N-node networks. These topologies are beneficial when information is sorted and indexed and not placed randomly.

2.2.1.2. Star

On a star topology, exists one central node that is connected to all the others with arcs. So all the information must pass through it. If the central node falls, the network stops working. This is a clear example that network reliability not only depends of nodes degree. In fact, this topology is not considered a distributed topology, but there are many grids that uses.

2.2.1.3. Tree

On a tree, nodes are connected always with a parent node and with one or two child. Only the end nodes of the structure (those on the far side of the first node) may have no children. When a new node (A) wants to connect to the network, first must choose a node (B) of bottom the structure that has one or none child. Then, node A appends it, transforming B node of its parent.

The advantage of this topology is that access to information is very ordered, and although despite be centralized topology, not all information must pass through the central node. In contraposition, the management of add, reallocate and remove the non-edge nodes is more complicated.

2.2.1.4. Mesh

In mesh topologies nodes are connected to one or more others, without any special restriction. So, the connectivity of each one depends of their degree and the concrete situation of all of them. There are several special cases of this topology, where the distribution and connectivity of the nodes provided special properties to the network. As example, in the “Fully connected” mesh topology, where all nodes are connected to the others, any node that remains on the network can be isolated. It only is possible if its node degrees is reduced to zero, so not is in the network yet.

Two other special cases of mesh networks will be described below.

2.2.2. Random

Mesh topology is considered random when the probability of a node is connected to each of others is equal. This means that if we look the structure of the network from a remote spot, the node placement can be anyone; do not have to be evenly distributed. However, the connectivity degree of nodes it should be similar, or at least comparable.

Let define N as number of nodes on the topology, deg(n) denotes degree of vertex n, MaxDeg maximum degree of vertex in a network, the clustering coefficient of a node is:

$$C(n) = \frac{deg(n)}{MaxDeg} \tag{2.1}$$

Random topology must be accomplish for all nodes v_x, that:

$$C(v_x) \simeq \frac{\sum_{n=1}^{N} C(n)}{N} \tag{2.2}$$

Even if this kind of topology exists in the nature and real world, create a completely random network synthetically is not a trivial task. Using a simple probability algorithm not ensures that all the nodes has been connected. Otherwise, using a incremental node aggregation [13] can create a non random topology more of the time, because more times the new node placement is guided or forced.

Usually, topology generation toolkits such BRITE uses a set of statistical properties of other networks to generate a special mesh nodes placement. Then creates a node connectivity using other algorithms based on nodes distance. This technique not allows a perfect random topology, but yet a statistical random topology. Their software construction and algorithms will be detailed on section 3.2.1.

2.2.3. Scale-free networks

Scale-free term began to be known around year 1999, when Albert Barabasi and their colleagues make a research work about Internet and mapped the topology of a portion of the web, finding that nodes connectivity follows a power law distribution [11]. Many other studies have been made since then demonstrating that a few other networks also had heavy-tailed degree distributions.

From social point of view many authors demonstrates that social relations between species (also humans, of sure) tends to a distribution of this kind in a social aspect such trust relationships, film actors collaboration [12] and so one. In line, biological phenomenons related to life, as for example epidemic transmission or geographic distribution of people around the world, also shows the same pattern. So these networks are able to reflect the type of relationships that exist in the real world with a good fidelity.

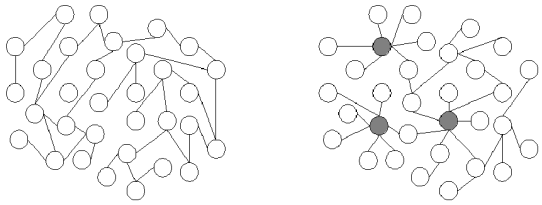

Figure 2.3: Random vs. Scale-free topology

Unlike random networks, where all the node relations seems equal, on free-scale networks is possible to detect two types of nodes based on their degrees connection (See figure 2.3) and placement. Exists some nodes that have a connectivity degree much larger than the rest. Its nodes, called “hubs” are very interesting for the communication process, since many of communications between nodes must necessarily pass through them. As a consequence, the other nodes tends to connect with this nodes in order to survive.

Indirectly when this happens network is giving more power to the hub nodes that others in terms of importance on collaboration relations. As as explained at section 2.3., well-selecting this nodes is possible to improve or encourage the cooperation inside the network.

2.2.3.1. Power law distribution

Power law distribution, is a mathematical model used to describe a relationship between two quantities (consumed value and consumers or people in the world and world surface) and follows next condition:

$$P(X > x) \sim x^{-\alpha} \tag{2.3}$$

as $x \to \infty$, $0 < \alpha < 2$.

Where represents the probability that X is greater than x. So, fixing α with a constant value, power law function behaves like Pareto distribution, that is the simplest heavy-tailed distribution. In this case the function is hyperbolic over its entire range and has probability mass function, that is described by equation 2.4:

$$P(x) = \alpha k^{\alpha} x^{-\alpha-1} \tag{2.4}$$

α, $k > 0$, $x \geq k$.

As a result, in our particular case, the number of nodes with a high degree connectivity (hubs) decreases exponentially as factor α, that only depends of its own connectivity.

2.2.3.2. Albert-Barabasi model

Other Barabasi important contribution was the creation of simple mathematical model for scale free networks generation. The algorithm [13] works as follows

a. Network begins initially with two nodes connected itself.

b. New nodes are added to the network one to one.

c. The probability of a new node is connected to a existing n node is proportional to the number of links that the existing node already has.

So that it is an incremental algorithm when the probability to a new node is connected to a node n follows the equation 2.5.

$$p(n) = \frac{deg(n)}{\sum_{v=1}^{N} deg(v)} \tag{2.5}$$

Where deg(i) is the degree of node i, N the number of yet existing nodes into the network. Heavily linked nodes (“hubs”) tend to quickly accumulate even more links, while nodes with only a few links are unlikely to be chosen as the destination for a new link. The new nodes have a preference to attach themselves to the already heavily linked nodes.

2.3. Cooperation through the topology

One of the emerging fields in the research of large-scale distributed networks is the evolution of cooperation. As there is no centralized control of behavior and actions of the nodes network security overlay layer must implement social policy methods to preserve common interests without harming the interests of individual users.

The specific method used depends on a number of factors. Some application protocols can encourage cooperation while others much more boost competitiveness. On the other hand, there are applications that do not require a concrete network topology, so the topology can be used as a means to develop or encourage cooperation. In all case nodes can be apply a lot of different strategies in order to maximize their achievements, so another possibility is encourage nodes to use a specific strategy positive for itself and community.

In practice, the main danger of such networks is the *free-riding effect*. It refers to a users or entities that consumes more than their fair should consume, or not pays the fair cost for their consumption. This behavior creates serious problem in distributed networks:

Free-riders, as beneficed user, not pay the fair cost, neither have the need to cooperate to earn a profit and don’t do it. Therefore consumed resources exceeds transferred resources and users not free riders fail to realize their expectations, at least in the proportion that are

cooperating in the network. As a result, this users that are the ones really are holding the system will react negatively and will to cooperate. So, system is not sustainable.

This section describes some of the most famous incentives methods tested over actual networks and try to show their advantages and drawbacks. Then, explain the tit-for-tat cooperation strategy and some related papers that studies the influence of network topologies over some variants of original game.

2.3.1. Incentive methods

Incentive methods are one of the solutions used to prevent and protect users of p2p networks from attacks by free-riding. Some times, through the evolution of the grid to a new grid or a different conditions; sometimes intrinsically. Selection ones or other depends of concrete game (application) and scenario (topology).

To decide the best strategy in each case is necessary to define two parameters: the variables to maximize and the assessment algorithms.

2.3.1.1. Variables maximization

Variable or variables to maximize must be an objective and measurable parameter of the nodes behavior, such as network throughput or the quantity of downloaded data. Additionally they should reflect indirectly the resources of nodes need or want to maximize individually if they can be take a advantages over others.

The degree of restriction of the method, increases linearly with the number of variables to maximize. A solution that claims to be fair with too many parameters probably end up being very computationally expensive and unreliable in a distributed network. Solutions with few parameters give more freedom to users to decide how try to maximize their possibilities and sometimes begin not secures.

2.3.1.2. Assessment algorithms

Not all maximization algorithms have equally aggressive justice. Some, like max min fairness algorithm used over networking distributes the available resources to the nodes satisfying first that demand fewer resources, and then to the most aggressive nodes. Others however, define justice as equality of resources consumed / disposed resources.

Once defined as the variable will be maximized is necessary to decide who and how to assess their compliance in order to define the cooperation / defecting protocol. This responsibility can be both, overlay and topology layer. Even, several studies [14] [15] [16] shows that cooperation can burn through a judicious combination of game and topology.

2.3.1.3. Basic incentive methods

Basic incentives methods are based on node neighbors evaluation about how collaborative the node is. To do this, each node saves the result of each request realized or received and evaluates individually the variable to maximize of their neighbors. Then, when the node receives a request from one of them decides if cooperates or not based on collected information. This technique have the advantage that each node can't do anything to fool the other nodes, so the information is saved by each node insteads announced. Their readability depends of concrete game and network conditions, but as happens on major part of application layer incentive methods, final decision is make individually.

Other traditional methods of incentives is the monetary system, where a node receives credits when give resources to others and spend its when want to use resources. This is a conceptually simple, but if it is not amended appropriately presents some drawbacks: if exist "credits server" the system is not fully distributed but if not, the credit accounting is only valid locally (one node to their neighbor)

Exists many other complex incentive methods such marketplaces [17], but the alternatives discussed and studied in this project are focused only on topology properties and concrete game (application protocol) uses.

2.3.2. Cooperation games

As mentioned earlier, complexity into incentive algorithms and overlay layer protocols reduce system performance too. So, protocols implemented into grids are usually simple and easy to calculate, like little games. That is the reason why in the literature on development cooperation are used simple games as a common mathematical framework for studying cooperation between unrelated individuals.

Games can be divided into collaborative and competitive games. Collaborative games have the same goals as collaborative grids (Remember section 2.1.1.1.) in contraposition of competitive games where the objectives are individual. Models more complex of competitive games contemplate the use of collaboration among small clusters of nodes to take advantage compared to other teams. However, we only focus on collaborative games.

2.3.2.1. Prisoner's dilemma

Prisonerś dilemma is a fundamental problem in game theory that has been used countless times to study why unrelated people choose to cooperate or not with them based on a common interest. In this game two players choose between cooperation (C) or defection (D), the payoffs for the two actions being as shown in the following (Table 2.1).

Relations between different possible payoffs follow the rule $b > c > \varepsilon \to 0$, that immediately poses the dilemma: if cooperation is costly to the individual and benefits only the interaction partners then Darwinian selection should favor non-cooperating defectors and eliminate cooperation, it leads to a highly inefficient outcome as compared to that obtained

Player decision	Co-player cooperate	Co-player defect
Cooperate	b - c	c
Defect	b	ε

Table 2.1: Payoff matrix of the Prisoner's Dilemma

by two cooperators. In other words, a decision that should be good for the individual leads to a poor result from the global (group, social) viewpoint [18] [19].

As individuals are anonymous within them, the decision of one or other strategy must be based on trust of others and their reciprocity. Is clear that if two players play a unique round of Prisoner's dilemma game (PD), the best strategy is to defect; since the benefit can be obtained is always greater than the risk. The interesting interaction evolution occurs when two or more players plays more than one PD round, so know the intentions of others and all become only partially anonymous.

This game has been used as a *framework* to study some human life perception problems. Not exists a perfect strategy to win the game because it depends of node's objective and particular perception, but the best way to achieve a good punctuation on 99% of cases is the tit-for-tat strategy. The idea is very simple: nodes start cooperating to encourage others and next rounds copies the neighbor strategy. As a result, if other nodes defects everybody defects and anybody came advantaged. But is other nodes are intelligent and understand the tit-for-tat strategy cooperates; so everybody cooperates and whole punctuation will be increased.

Another factor influencing the behavior of the game is the relationship between b and c. In several studies, such as [20], has shown that there is a threshold beyond which regardless of network topology, defecting is more profitable to cooperate. Figure 2.4 shows it in a tit for tat with particularized game protocol conditions over several networks, where b value is 1 and c values is renamed as b.

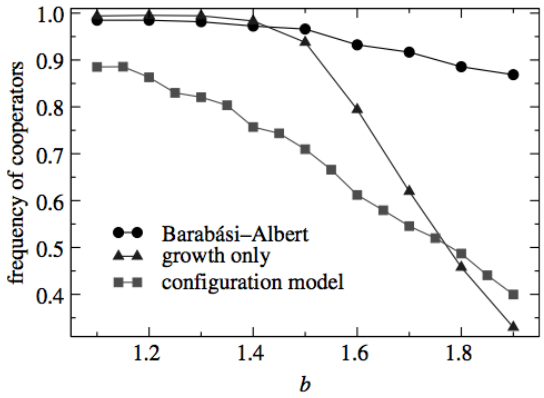

Figure 2.4: Impact of payoff relationship on PD game

This means that when defectors gain approximately 1.85 times defecting and other twice cooperating than cooperating both, is better and more productive, defect always than cooperate.

2.3.3. Influence of the topology over the cooperation

Although it was not the objective, Figure 2.4 also shows that there is a strong difference between behavior of the nodes in the game if the network topology changes. Others authors also published similar studies when cooperation evolution depends of a fixed network topology. These studies are detailed in their publications [18] [19] but, in order to have a comparable results from our simulator, below the most important results will be explained.

First of all, papers describes the payoff matrix as: if both players defect, each one receives ε points and if each one cooperates, both receives 1 point. Other cases, defecting receiving b and cooperating 0 points. The game is played discretely in turns where individuals cooperates or defects as prefixed value (on start, half cooperates and half defects) At end of each turn, nodes study the results of their neighbors and copy the best results strategy.

So, game has been played into 4 different networks until reach an density cooperation equilibrium point. Figure 2.5 shows a graphical resume of the results of their study where cases A and D correspond, respectively, to the synthetic classes of networks. In case A communities have been built as Erdos-Renyi random graphs. Communities in case D are constructed as independent scale-free networks (Barabasi- Albert with $k_0 = 3$). Case B and D has been obtained from D and C increasing the clustering coefficients. IH represents the clustering coefficient and IC the inter-community connectivity.

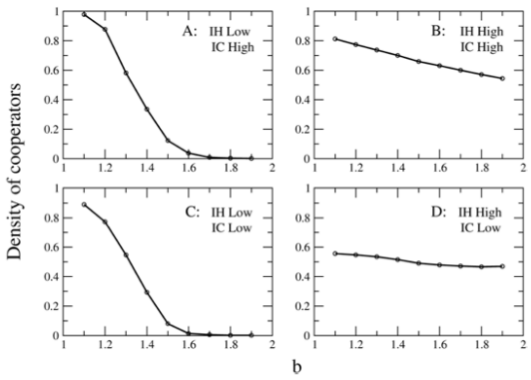

Figure 2.5: Evolution of cooperation in four synthetic networks

The results of this study allow us to draw two important conclusions about how the topology influences to achieve the cooperation of the nodes. At least in a game like tit-for-tat original.

- B and C cases experienced a market increase in cooperation, so it is clear that clustering is a very important requirement to achieve cooperation between nodes.
- High inter-community connectivity, so avoiding power law distribution effects, improve the cooperation density where the payoff relation is closed to 1.

Finally, the simulations of Chapter 5 will focus on comparing and validating the results of the studies referred in this chapter throughout a developed simulator simGRID in two different games (another tit for tat and a proposed game “Àrea-equilibrium”) on different network topologies comparable to these.

CHAPTER 3. TOOLS

This chapter describes all the technologies and tools involved in the realization of this project. Begins with a description of programming language C++, specific tools and libraries used and justifying their selection from a practical point of view of project requirements. The second and largest part focuses on the description of the different tools used for the generation and analysis of the topologies used and evaluated in this project. All tools used are free use and open license. Furthermore, except Pajek, all other can be used through a GNU/Linux platform.

3.1. Development tools

Today, exists many languages and development tools that provide an abstraction level good in of. Since pure objective oriented languages such Java and .NET, to more descriptive programming language as SmallTalk. Select one over another is often a decision based on previous language knowledge. However, in this case the selection of a concrete language has been the result of an assessment of the simulator requirements, previous experience over simulator development and the possibilities and tools of each language.

On design stage, several challenges from a programming point of view that must be detected. Firstly, concurrent programming [21] should be avoided as much as possible to control everything that happens in the simulation and take advantage from full cycle simulation paradigm (see section 4.2.1.). However, this affects adversely the speed of simulator. Second, I/O (Input and output) information through console or file system is more costly of all the simulator; so is imperative use a optimized method to do this kind of operations, such statistics saving. Finally, to support scalability and avoid memory leaks the language selected must be provide a secure garbage collection methods or an equivalent solution. To accomplish all of its conditions, that can be resumed as processing speed and memory control, C++ language has been selected.

3.1.1. C++ language

C++ language is a multi paradigm language based on C generic language. Well as language that maintains some of the paradigms of C (generic and structured programming paradigms) and includes one more, the OOP (*Object Oriented Programming*) paradigm. In comparison, C++ has a performance level somewhat lower to its predecessor, but supplies it perfectly by the use of objects and structured classes.

Two other particularities of this language are its ability to overload operators and create new data types at runtime to behave as basic types. This two techniques, allow the use of more versatile design patterns and better memory optimization [23]. Clear examples that have been used in this project are *smart pointers* and *data pools* [22], that have allowed to implement the famous Java garbage collector, or strategy pattern, which allows at runtime to decide which of the strategies you want to use a node to resolve a sharing request. Finally, the use of C/C++ pointers improves performance of the system, avoiding

the physical movement of data on memory.

Other two libraries has been used: *log4cxx* and *Doxygen*. While first solves soft logging problem, second library provides a simple documentation tool on C++ language. This two tools are essentials to assure a easy development and debugging processes.

3.1.1.1. Log4cxx library

Log4cxx [24] is a port of original log4java implementation created by *Apache Software Foundation* that provides a logging system framework based on value level. Each level, represents one different kind of information. Trace message system, allow the user to configure the set of levels that wants to and receive the information associated. Only three levels has been used in this project: INFO level to show generic progress information, DEBUG level to test the application on development stage and ERROR messages related to file management failures.

Is important note that this logging framework is fully transparent to software execution, so its internal traces and execution not harm the simulator performance.

3.1.1.2. Doxygen library

Doxygen [25] is a documentation system under GNU license for C++, C, Java and more than 10 other languages. It is able to provide full documentation from a set of documented source code. Also, can extract the code structure from undocumented source files showing graphically the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams. Once compiled, all documents with diagrams can be viewed in multiple formats such as HTML, RTF, PostScript, PDF or Unix man pages.

This library has been used with two different purposes: firstly, to provide documentation for future projects based on the simulator, in addition to study and verify the proposed relationships in the design are met by the simulator.

3.1.2. Gawk

Gawk [26] is a GNU implementation of *awk* offering a easy-way and powerful string and file manipulation language performed by *Free Software Foundation*. With this, is possible manipulate text file in order to operate on data from parts of certain lines, make changes in various text files wherever certain patterns appear without writing a complete program.

In this project, Gawk is used to convert different files topologies (such BRITE topology description or Pajek input vector files) between them. The main advantage is that this post and pre processing is performed much faster that if done within the simulator.

3.2. Topology and analysis tools

Earlier already explained some of the difficulties with the generation or treatment of topologies, so as to simplify their treatment three tools have been used: BRITE, Otter and Pajek. The first of it focuses on the generation of synthetic topologies from mathematical models and provides a set of object classes in C++ and Java to expand and used in the simulator. The next two focus on the study of topology and will be used to verify the properties of the topologies created and relate the results extracted from the simulator with its.

3.2.1. BRITE

BRITE [27] [28] (*BRITE: Boston university Representative Internet Topology gEnerator*) is a open source topologies generation tool developed by the University of Boston focused on Internet models. Although is no longer supported by its developers, it remains one of the reference tools relating to topologies generation thanks to their well mathematical models implemented and their full compatibility and interoperability with other topology formats as Otter (See section 3.2.2.) or NS.

Have two identical implementations, one on Java and other in C++ with a common graphical interface only provided through java implementation. As both implementations work equal and the mathematical algorithms implemented in the same way, in our case Java option has been used to generate topologies externally the simulator (and override the configuration file through Java GUI) and C++ option has been used to extend BRITE functionalities (create BRITE based components) inside the simulator.

Topology composition is divided in three areas: topology hierarchy, node placement and nodes interconnection.

- Topology hierarchy: topologies generated can be have different topology level. Each level can be made as a independent topology with a specific node placement and interconnection. Finally, hierarchy model represents how and who interconnects two consecutive layers. In our particular case, all topologies used are *Flat routing model* (See figure 3.1), so only haves one layer.
- Node placement: BRITE allows to define as the nodes are located in an imaginary array of N by N points. The options available for the topologies of type flat routing model are the Barabasi model, extended Barabasi, Waxman and GLP.
- Nodes interconnection: nodes interconnection depends on the bandwidth mathematical model distribution used and the mathematical connectivity model. Bandwidth distribution not have any importance in our simulation because we consider al the links equals.

Results of this software is a text file *.brite* with two parts: first part have a line by node, where describes its properties (node placement coordinates as example) and second have a line by link describing nodes interconnection properties between two nodes connected.

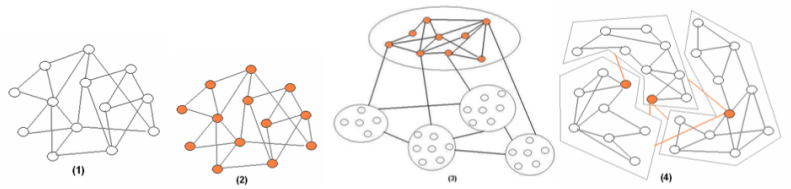

Figure 3.1: BRITE model topologies based on hierarchy

This archive will be processed by the simulator only in order to construct the connection nodes topology, so links.

Then describes the process of generating the two topologies used in the project.

3.2.1.1. Random topology generation

Random topology don't exists in BRITE. However, is possible to generate a statistical random topology using Waxman model definition. Waxman topology is a random node placement mathematical model where the probability of a node n is placed in a concrete (x_n, y_n) is constant for each pair of x and y. Then, the probability for interconnecting the nodes n and u, is given by equation 3.1:

$$P(n,u) = \alpha e^{\frac{-d}{\beta L}} \tag{3.1}$$

where $\beta < \alpha, \beta \leq 1$, d is the Euclidean distance from node n to node u, and L is the maximum distance between any two nodes. All topologies generated in this project have a $\beta = 0.15$ and $\alpha = 0.2$ with a placement matrix of 1000x1000, so $L = \sqrt{10001000} = 1000$.

As equation 3.1 is a negative exponential that slowly decaying and depends by distance and in addition nodes are placed with the same probability one from others, the result is a statistically random topology connection, that is our objective. Note that as discussed above, when equation 2.2 has presented, the statistically random topology is different to full random topology.

3.2.1.2. Power-law topology generation

As BRITE provides a node placement model based on Barabasi model, this is used to create a power law topology. So, this model interconnects the nodes according to the incremental growth approach. This has been described in equation 2.5.

3.2.2. Otter

Otter [29] is a Java application developed by CAIDA (*Cooperative Association for Internet Data Analysis*) in order to create a graphical visualization toolkit of topologies conformed

by nodes, links and arcs. Although its development is also seen standing, as BRITE, is a widely used tool for the scientific community to carry out graphical analysis of topologies.

Its main features focuses on graph the topology using geographic positioning, circle or value based. Also, is able to represent nodes degree, links properties or another configured value with a graphical color scale. Is is used in this project to provide a general image of topology created by BRITE in order to confirm the statistical required properties of the topologies.

3.2.3. Pajek

Pajek [30] is a more complete and complex Windows program for analysis and visualization of large networks having some thousands or even millions of vertices. It is implemented in Delphi (Pascal) and freely available for noncommercial uses. Provide tools for analysis and visualization of such networks: collaboration networks, organic molecule in chemistry, protein-receptor interaction networks, genealogies, Internet networks, citation networks, diffusion (AIDS, news, innovations) networks, data-mining (2-mode networks), etc.

Its operation is based on the decomposition of the topologies in confined structures that can be analyzed within its local context and the use of a global hierarchy that reflects the relationship between each of the decompositions.

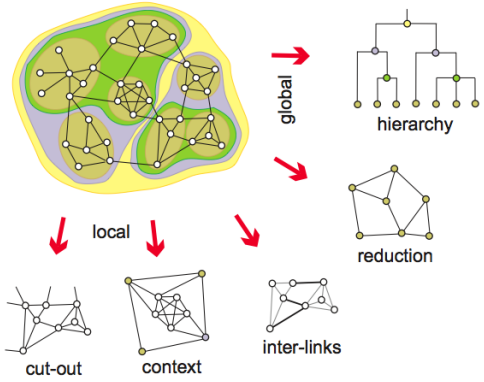

Figure 3.2: Approaches to deal with large networks

As shown in figure 3.2 these decompositions are new graphs, but that need not be part of original topology. Always are smaller topologies from the root topology (cut), some times a new representations of its node relations (context) and other a simplest representation of its more important nodes (reduction) or links (inter-relations). So, the study a large topology can be transform in a tedious work.

From project point of view, Pajek has been used to study the graph characteristics of all the topologies (as average node degree or density of nodes) and the relationship between simulation results and topologies. Chapter 5 describes and analyzes it.

CHAPTER 4. DESIGN AND IMPLEMENTATION

First part of this chapter focuses on a game design that is representative of proposed distributed application *VRIMS* mentioned at Chapter 1 as a long time objective of this project. Also, defines two cooperation strategies based on PD description presented on section 2.3.2.1.: one is an adaptation of Tit-for-Tat strategy and other is a own created cooperation strategy based on CPU occupation.

In second part, a global vision of the simulator, their input-output interfaces expected and requirements will be presented. Next, two simulation paradigms, “step-by-step” and “full cycle”, will be discussed in order to understand the simulation architecture implemented and the core manager state machine.

Finally, last part of this chapter presents the architecture designed and implemented in this project to study the game through the cooperation over different kind o topologies. Also, layers paradigm and basic modules functionalities will be explained in detail, with special emphasis over its modularity and expansion capabilities.

4.1. Game and strategies design

Prisoner’s dilemma presented on previous chapter is a good framework to study the nodes behavior. This is not new, so is demonstrated by many authors and used over different research studies and areas. But the other important feature in PD is the ease with which can be implemented in an environment where multiple nodes interact. Basically, offers two possibilities to each node (Cooperate or Defect) and evaluates one-to-one their results to achieve a punctuation. To make comparable proposed application on chapter 1 has been reduced to a simple game that emulates CPU sharing.

Other simplification applied is substitute the cooperation evolution presented on related articles (See articles [18] [19] [20]) where nodes decision is fixed and strategies can be change after each round or cycle, by a unique strategy for all the nodes that not changes anymore and contemplates both responses: Cooperation (C) and Defection (D). As consequence, the game should not seek an answer to *what topology/game evolutes into cooperation?*. Instead, supposing a fixed topology and game, the question is: *which is the percentage of cooperation and why over this conditions?*. This implies a second important change on the game in comparison to PD: point is not the objective.

Nodes plays a known game mentioned earlier without any particular objective. Independent of the results on each round, the strategies not changes in future, only their variables. Finally, the cooperation and defection rates are compared with other statistical data to explore some conclusions.

4.1.1. CPU sharing behavior

The CPU sharing game implemented intend to replicate whats happens on real world into a resource sharing scenario. On real scenario, each machine have a variable quantity of CPUs with different features (number of cores, frequency or hyper-threading/processing capabilities are some good examples). When a node does not have the necessary resources to perform a task, it request a portion of neighbors node resource to help it make a task. After negotiation some of other nodes decides cooperate, and others to defect. In any case, requesting node knows how much resources have achieved and if is sufficient to make the task uses it, but if not free all the pre-allocated resources. Finally, when the task is performed node free all resources in use.

Mapping all request cases and model the nodes management of CPUs, portions of CPUs and the decision process involved implies a huge state machine, with a decision-making process that would require a complex intelligence level to evaluate all possible combinations and reactions of the nodes. So, not makes the game comparable with basic PD and makes statistical results very complex to process.

In order to simplify the game, all of its possibilities has been reduced. Our game only focuses on CPU resource. Also, transform CPUs properties into a fixed and discrete value for each node, a natural number that represents their maximum available CPU that in a given time can be used by it or temporally transferred to other node of the topology. This variable can be take a minimum value of 1 and a maximum value of *maxCPU*. Moreover, fictitious tasks CPU requirements are represented by a discrete values of CPU. This value is represented by a variable between 1 to *maxRequestCPU*, that must be higher of *maxCPU* value in order to represent a non-realizable process by unique node. As occurs on real grids, some times the task can be performed by only one node (because is a lower cost task or the node have a high value of CPUs slots) but the major part of time it is not possible because task requires more than available node CPU slots.

Finally, all task needs a given time to be completed. This is represented by a third variable from 1 to *maxTimeCPU* that represents, as discussed on section 4.2.1., the complete number of simulation cycles during CPUs slots are occupied making this task.

Next, game CPU sharing process is resumed:

a. Before start the game, CPU slots from 1 to maximum CPU is assigned to each node according to *maxCPU* variable.

b. Each round, nodes that are not performing any task must decide if they have any with a probability represented by the variable *probAct*.

c. If yes, calculate the amount of CPU required (randomly, from 1 CPU slot to *maxResuestsCPU*) and the time required (also randomly from 1 to *maxTimeCPU*).

d. Then, if needs more than their available CPU slots, requests all the neighbors nodes many slots as require.

e. After game algorithms decide if neighbors nodes cooperates or defect, all involved nodes start the task (blocks CPU slots during required time). If a node gets more

resources than it need, automatically release those who left over (picking at random among those who achieved)

Note that in our particular game behavior tasks will be performed although a node has not achieved the necessary resources. This makes the algorithm much less aggressive, ensuring that resources are used more time. Another important detail is related to CPU assignment: on step (*b*), nodes first must be satisfy their requirements and next, if they need more resources, ask to others. This and other related question to discretization process will be discussed on section 4.2.

Finally, table 4.1 shows game variables with their possible minimum values, maximum and default value of maximum variable.

Variable	**mean**	**min.**	**max. (default)**
Node CPU	Available CPU at a given node	1	maxCPU (3)
CPU request	CPU requested by node for one task	1	maxResuestsCPU (10)
Time request	Time during node requests CPU	1	maxTimeCPU (3)
Prob. request	Probability with which a node decides to perform a task	probAct	probAct (50%)

Table 4.1: Proposed game variables

4.1.2. Game algorithms

Game algorithms represents possible strategies that each node can be use to achieve their objectives and can be exchanged between them, or replaced by any other. There are three conditions that all must follow: at first, its response to a node request can only be based on the information collected by them in the past. Second, this information can only come from the requests and responses with neighbor nodes, not can be announced by other nodes. And last one, responses only can get yes or not values.

Information gathered by requests and responses are: sender and receiver identification number, requested CPU, time requested and, in case of response, yes/no value. Obviously, strategies can be use information generated by them as statistical values.

Next strategies descriptions explores the concrete two strategies implemented on the simulator. In order to simplify their explanations, from here supposes a little topology with three nodes connected between them. Each node have a fixed value of CPU slots available and a identification tag A, B or C.

4.1.2.1. Tit for tat non evolutive

In the original tit for tat strategy the nodes respond to a request with the same answer it have received from the other the previous round. The objective is cooperate only if other

node is willing to cooperate and defect if not. Articles discussed on section 2.3.3. explain some adaptations for this strategy if nodes play versus more than one node. Simplifying, the response to a particular node can be based on all nodes related information or only on their.

Tit for tat non evolutive is a strategy adaptation for CPU game. In this case, each node (A) saves the last unitary responses (See section 4.2.1.) received by others when it have been requested resources. And, when have to evaluate a request from a node B:

- If A not has been requested CPU never to B: A cooperate 50% times
- Else if A has been requested CPU to B last time and all B unitary responses are defect, A defect.
- Other cases, A cooperate.

As a result, nodes only cooperate if their neighbor have responded positively to a last request, in whole or in part. If node not have information, their decision is drawn.

4.1.2.2. Area Equilibrium

Area Equilibrium is a new strategy created specially for CPU sharing game, also based on Tit-For-Tat idea, but with a different objective. While previous presented game objective is evaluate two nodes one to one, this strategy tries to evaluate a requested node (A in our examples) with all of their neighbors (B and C) with the objective to equilibrate given CPU slots with archived CPU slots.

To do this, Area Equilibrium strategy keeps the relationship between CPU slots consumed by others and CPU slots assigned to itself. Next, when receives a request from neighbor node calculates a probability p to accept their request, described by formula 4.1.

$$p(n) = 100 - \left| \left(100 \sum_{t=0}^{T-1} \frac{CPU(t,\vec{v})}{CPU(t,\vec{v}) + CPU(t,n)} \right) - 50 \right| \tag{4.1}$$

where p(n) represents the probability of node n to accept a request, $CPU(t,\vec{v})$ the CPU slots consumed at given time t by the neighbors nodes v_x, $CPU(t,n)$ the CPU slots consumed by itself and T the current time. As a result, if both consumed CPUs are equal (owned and others), the probability to response with a Cooperation symbol is 100% but if it is not balanced, the probability goes to 50% in the same proportion. This minimum probability of 50% tries to make both strategies equally aggressive.

Note that this strategy differs from Tit-for-tat on the evaluation of all neighbors instead only request node and try to consider all the cooperation history, not only the last round played. As a consequence, while on the nodes of low connectivity we expect a similar behavior of both algorithms, in nodes with many degrees of connectivity we expect that AreaEquilibrium strategy tends to get more cooperation.

4.2. Simulator implementation

In our case, *simulator* is a program that imitates some real, but not existent yet, distributed application under different initial conditions in order the study and predict their behavior. As it is not a real implementation there are several factors that must be modeled because are unrelated to the study which you want to, or that its influence is not perceptible.

In this case some assumptions have been made in order to simplify the simulator: first consist on suppress all physical effects not really related to cooperation problem as for example networking effects and other machine physical processes. Next, as is discussed on previous section the topology and nodes connectivity no changes during the game. Finally, is assumed that each node starts the game on time 0 with no initial conditions and plays a fixed X game rounds.

Some others factors, as game decision process, should be simplified to adapt a time continuous world functioning with a discrete round system. Section 4.2.1. discusses the alternatives.

4.2.1. Simulator paradigms

One of most important decisions to be taken to create a simulator is *what is a round?* To response this questions exists to opposite simulator programming paradigms: the full cycle system and the step-by-step.

On *full cycle* a round equals to a long period of time in which many things happen. In this case, the number of cycles is reduced enough but the complexity of calculations increases too much. *Step-by-step* paradigm uses a smallest significative time period for represent a round, where only happens one or at most two “things”. On this occasion the number of cycles increases, reducing the simulator performance, but is very easy to control all the variables and values. While first approach is used more on discrete systems, second is used more when when the actions occurs is more important. In our case, as the game proposed is a turn based game when each node makes several things, first approach is used.

As non-related effects has been deleted to the simulator, on simGRID time has been discretized into a big rounds that represents the time where all the nodes, at same time, makes a set of actions: actualize theirs status, decide whether to perform a task, request CPU, solve requests and response, assign CPU slots and save statistical data.

4.2.2. Simulator requirements

As same manner that project objectives has been listed on earlier, concrete simulator requirements are also created in order to evaluate it as tangible result of this project. In this subsection there are presented this requirements with a related architecture solution. To show how concrete solutions are implemented is recommended read architecture section.

4.2.2.1. Archive inputs

Simulator need a well-made topology from a external file as generated with BRITE program to known how nodes are interconnected between them. To make this process more simple as possible any UNIX/Win32 text archive can be used to input the topology if follows next format: UTF-8 file where each link is represented by a line and values (at least three) are separated by tabulation or whitespace. First value represents the line number from 0, second and third number represent two nodes connected by link. Next example construct mesh node presented on figure 2.2 (d).

```
0 0 1
1 1 2
2 1 3
3 2 4
4 3 4
5 3 5
6 4 6
7 5 6
```

As can be seen, this format is completely equivalent to the second part (links) topology file created by BRITE application and presented on section 3.2.1.. So, is important to note that simulator supposes that relationship between nodes is bidirectional.

4.2.2.2. Fidelity

Although it presupposes a series of estimations to improve system performance, to simplify the development or caused by the full cycle implementation, the core must act in the most faithful to reality in order that the results can be considered valid. To ensure this, on statistical module some “variables of control” has been added that checks the well-known expected results and alerts if it not agreed. Also, as explained on section 5.1. a set of experiments are used to improve the simulator results before extract any conclusion and make the set of final simulations.

4.2.2.3. Modularity

In order to profit the simulator and use it as a generic toolkit to evaluate cooperation over different games, strategies and topologies is important ensure that software architecture be modular. It means, that all the configurable elements must be interchangeable by other similar. To do this, all modular elements present a interface definition that list all the function required by this set of elements. So, specific classes can be substituted by other concrete classes that represent a different model but the same pattern. C++ explicit and implicit interfaces make this job.

4.2.2.4. Performance

One of the requirements, related to the programming language selected, is the optimization of resources consumption and maximize the performance of the application in order to provide a framework that can be used to evaluate large topologies in a limited time without requiring expensive machines or cloud computers.

Related solutions are the use of smart memory allocation thought pool templates and shared C++ pointers.

4.2.2.5. Simulation results

The outcome of each simulation is returned in a plain text file, similar to the topology input file. Each line represents a simulation time (round) and each column corresponds to a concrete statistical result. Table 4.2 shows all the variables and its meaning.

Value	**Description**
Round	Round number iD
requestedCPU	Total requested CPU in a given round (external variable of control)
receivedCPU	Total received CPU in a given round (external variable of control)
givenCPU	Total CPU that nodes accept to share or use
reallyOwnUsedCPU	Total of CPU shared by node to itself
reallyOtherUsedCPU	Total of CPU shared with others
requestedCPUbyNode	Average of requestedCPU by nodes that request CPU slots in this round
receivedCPUbyNode	Average of receivedCPU by nodes that receives CPU slots in this round
givenCPUbyNode	Average of CPU that nodes share or use divided by participant nodes
reallyOwnUsedCPUbyNode	Average of CPU slots that nodes use on our own benefit
reallyOtherUsedCPUbyNode	Average of CPU slots that nodes share with others
satisfiedNodes	Number of nodes that has satisfied 100% their request
askNodes	Number of nodes that perform a request

Table 4.2: Output simulator variables

Also, a second result archive is created by the application with the average percentage of satisfaction of each node individually computing only the last 50 rounds. Each line represents a node and have only three columns: the node iD, the total of satisfied requests and the number of requests. As first archive allows to study the cooperation of the whole system, with the second archive is possible to study the effect of the nodes placement over topology on their individual result.

Another important feature to implement related to statistical point of view is the cutoff system similar to Pajek presented that allows users to select what and how variables will be saved on configuration file.

4.2.2.6. Logger system

Logger system allows to users and developers to control the status of the application, monitoring the percentage of computation finished, input/output errors on files management and internal variables of control. Also, this feature is used on debugging process to fine control the simulator operation.

4.2.2.7. Parameters

All simulation parameter must be configurable in order to launch different simulations without compile the program each time. Table 4.3 shows parameters description:

Value	**Description**
InputFile	Topology description input file
Output file	Generic output results file
Nodes	Number of nodes used
maxCPU	Maximum CPU that a node can have (*maxCPU* variable)
Game	Concrete game to play (See section 4.3.)
Strategy	Concrete strategy used by each node (global or individually, see section 4.3.)

Table 4.3: Configurable simulator parameters

Finally, simulator is distributed as GNU/Linux library, along with several sample programs that show the configuration of the simulator for one or more simulations concurrently. Also, is important take into account that news games and strategies need a concrete implementation and definition inside the source code.

4.3. Architecture

In order to simplify the design, architecture has been divided into different layers of management that are responsible for a particular aspect of the simulator. Another advantage of this distribution is for exchanging each of the layers independently, without requiring a redefinition of other, making the simulator a much more flexible and powerful toolkit.

As can be show in figure 4.1 simulator is based on two heigh level layers (core manager and physical) and two support layers that allows to attach or remove game and strategy capabilities to each node and save the manage statistical data of topology. Finally, tools

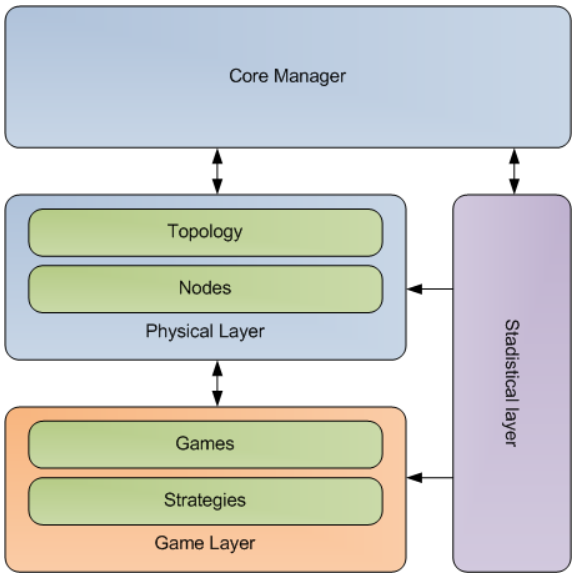

Figure 4.1: simGRID global architecture

infrastructure provides classes to manage common aspects of all the classes and can be used over all the layers.

4.3.1. Core manager

Core manager is responsible to control the flow of simulation process through a simple circular state machine represented by figure 4.2. First of all, core manager configures the simulation variables (as number of simulation rounds) and receives a topology object represents a concrete physical representation layer. Next, starts a new singleton instance of statistical modules and begin a new start-up configuration that ensures that not exists any pre initial condition.

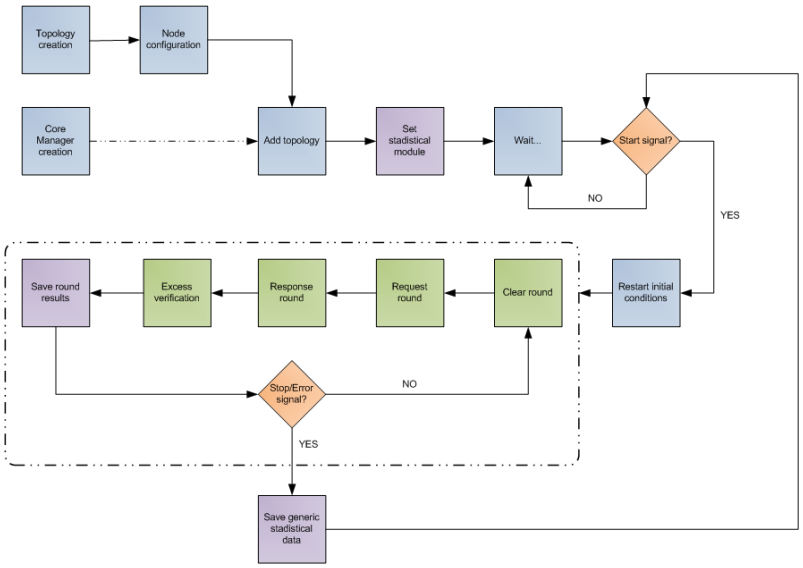

Figure 4.2: Core manager state machine

Once configured, the manager wait a start signal and then plays the simulation until the end or an error/stop signal interrupts the process. Note that simulation process has to be divided into five parts, in order to ensure that every action is performed synchronously by all nodes. Second and third steps, request and provisional responses, has been sliced to assure the same probabilities to respond a request to each node, independently of why simulation asks it.

Four, and no defined step, assures that requesting nodes leaves the non necessary achieved CPU slots and confirm the necessary slots achieved. This decision has been made randomly. As a result, the order of execution or placement of nodes not has importance because all occurs discretely into a simulation time and only necessary.

4.3.2. Physical representation layer

Physical representation is the most complex layer. It represents the infrastructure of a network and all of its elements and capabilities. Is responsible for building network infrastructure (through the topology file), configuring each node, manage communication between them and send reports of all that happens to statistical layer.

4.3.2.1. Topology classes

Topology classes are the responsible to construct and place the nodes into a defined array. Also, manages all the communication out-of-game, so all calls to functions and parameters not related to the game, but with the simulation. More important example is the statistics collection.

4.3.2.2. Node classes

Node classes represent a node, as a configurable object that provide abstract methods to access to it game and strategy defined. It transforms this class into a complex prototyping class represented by figure 4.3

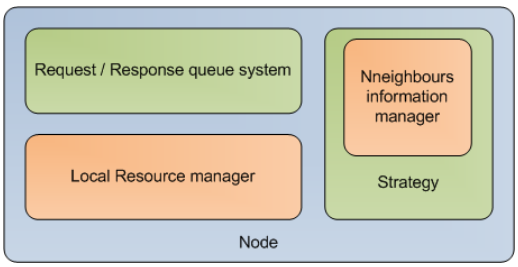

Figure 4.3: Class Node model

Also, implements a message infrastructure to provide a communication with their neighbors. This communication infrastructure has been developed using a internal messaging protocol composed by a generic message class (with sender, receiver and generic data

description) and two inherited classes that represents request message and response message.

4.3.3. Game layer

Game layer has two differentiated parts: the sharing CPU game sharing and Strategy described interfaces.

Class	Methods to implement
IGame	virtual request play()
IStrategy	virtual bool resolveRequest(RequestMessage* msg)
IStrategy	virtual void setCooperationThreshold(int cooperationLimit)
IStrategy	virtual int getCooperationThreshold()
IStrategy	virtual void addResponseData(list<ResponseMessage*> responseMessageList)

Table 4.4: Game layer interfaces

Any new game or strategy can be included in the simulator always inheriting from these classes and implement their functions, that assures that each game can be play a game (obviously) and each strategy can get messages data and return a decision when receives a request. In addition, it must be registered in the list of games and strategies enumerated in the core manager.

4.3.4. Statistical layer

Statistical is a counter infrastructure that is able to remember all the requests realized through the simulator. Is implemented through a POJO data collector structure and singleton template. Also, to avoid memory leaks or simulation failure the data results is recalculated and saved into result file each round. Is perfectly justifiable this performance loss to improve the safety of results.

4.3.5. Tools infrastructure

Tools infrastructure have three basic patterns implementations to simplify the management of data.

- Singleton: is a creational pattern that ensure a class only has one instance, and provide a global point of access to it. Is used to create the core management and statistical architectures.
- Shared pointers: it ensures all messages are destroyed once they are no longer used (referenced) by any other object.

- Pool data management: gives a shared pointer object of the required object and saves the original object into a memory block. With this technique is able to assure global access to all of objects used inside the simulator from statistical modules.

CHAPTER 5. SIMULATION RESULTS

Our simulator has enabled us to understand the behavior of multiple nodes within a distributed network while playing a modified version of prisoner's dilemma game. These results were contrasted with different topologies with values of clustering variables (Heavy tailed and Random), cooperative strategies (Tit-For-Tat and Equilibrium Area) and some related papers.

Using the tools presented on Chapter 3 (Otter and Pajek) are able to analyze the results of these simulations and draw some conclusions, some directly from the charts others by deduction, which will allow us to create a rules of cooperation in the p2p application to share resources discussed in Chapter 1. This chapter presents these results.

Other critic point in our scenario is the topology creation. As is demonstrated on section 3.2.1. the generation of topologies performed by BRITE are based on mathematical and statistical models, so we should also verify that the topologies used correspond to the desired.

First, the entire set on simulations, conditions and variables used will be presented in order to be clear to what extent the results presented are valid. Next, discussion about the method used to verify the simulator and how control variables has been helped will be made. The influenced detected from strategies over the PD game will be discussed on second part. Third part exposes the opposite point of view, trying to discover when is important the topology and its effects over the results. Finally, last part exposes a algorithm proposal to be implemented over a real CPU sharing distributed scenario as a validated result from these experiments.

5.1. Set of topology/game test

Some of parameters presented on design and implementation chapter can be configured on fly each simulation in order to emulate a set of different scenarios quickly. But, some other conditions must be fixed on source code. Law of conservation of energy commented above is also applicable here: to make the simulations comparable some of conditions must be remain constants. All of it conditions will be listed on table 5.1.

Note that on theoretical articles [18] [19] the number of simulations gone to very high values the order of thousands. It have sense when the objective is a equilibrium point, but in our simulation the evolution is avoided; so the number of simulations only must become upper to transient time (about 65 rounds in extremely cases). Also, some simulations has been tested with 10 times plus nodes, but results achieved the same statistical distributions, so 1000 nodes has used in order to reduce the simulation time.

Moreover, to assure that results achieved in simulations not will be constricted to a concrete topology created, all the simulations has been made with three different topologies of each type. Table 5.2 shows all the combination of simulations realized:

Is important take into account that neighbor value used is perfectly respected on random

Parameter	Area	Value
Number of nodes	Topology	1000
Area	Topology	1000x1000
Number of neighbors	Topology	Fixed on each topology
Strategy	Game	Fixed the same to all nodes
CPU slots	Game	Equally, random distributed
Rounds	Game	250
Statistics rounds	Game	Average of last 50
Slots request	Game	From 1 to 10 slots
Time request	Game	From 1 to 3 rounds

Table 5.1: Global settings for simulations

Strategy	Topology	Neighbors
Tit-For-Tat	Random	From 1 to 5, then from 10 to 50 in steps of 5
Tit-For-Tat	Heavy Tailed	From 1 to 5, then from 10 to 50 in steps of 5
Area Equilibrium	Random	From 1 to 5, then from 10 to 50 in steps of 5
Area Equilibrium	Heavy Tailed	From 1 to 5, then from 10 to 50 in steps of 5

Table 5.2: Simulations

topologies where all nodes have the same number of degree, but as will be demonstrated on section 2.2.3.2. is not mandatory in heavy tailed topologies.

Then, a second set of simulations was repeated with all combination with 15 and 35 nodes (as samples of poor and high neighbor connectivity) but modifying the maxCPU number from 3 to 10 CPU slots.

5.2. Verification of simulations

Before use the simulator, as occur with each unknown tool, is important to assure that it is working well. It can be done in two ways: on execution time using a set of verification variables that test some critical parameters and make a report about their status, or perform a series of tests with a set of entries whose result is widely known. First approach is necessary in order to detect some misstating between design and implementation or critical parts of code, while second test is also necessary to test that design is correct and whole functioning of program.

5.2.1. Control variables

One of main characteristics of non evolutive simulators, those that their physical modeled representations not changes their behavior on time, is that each round is simulated in same manner. Only change accepted is the initial conditions of the turn, that is inherited from last round, that can be changed them. As a result, two kind of conditions, so variables, appears in source code: round variables and simulation variables.

First group, also called invariants in theoretical computer, follows the “law of conservation of energy”, so can be verified on execution time. The law of conservation of energy is an empirical law of physics. It states that the total amount of energy in an isolated system remains constant over time (is said to be conserved over time). A consequence of this law is that energy can neither be created nor destroyed, it can only be transformed from one state to another. It can be applied to round named variables and objects and means that the amount of round variables remains constant (invariables) from round to round over all the simulation in quantity and value. Compliance with this ensures that the rounds are independent, although the physical representations of the nodes are not. For example configurations variables are, in fact, statics. Next list shows the most significant:

- Internal pool messages: during each round, all the messages must be attended and deleted before next round starts. So, messages are a temporal representation of requests and responses.
- Requests and responses: statistics module counts all the requests (see section 4.2.2.5.) has been generated a pair response from receiver to sender with same original data plus the response value.
- Statistical results: the number of statistical requests and responses must be equal, as same manner that quantity of CPU shared and consumed. Also, this is comparable through the requests and responses by nodes, that not are equals but allows to asura that the number of participants nodes (requesters plus responder) are equals.

Last one represents conditions that can be change on time and only is possible verify those through simple simulations. Their proof will be commented on section 5.2.2.1.

5.2.2. Verifications

The special set of tests explained next verifies the operation of simulator and the correct construction of topologies.

5.2.2.1. Simulator verification

To verify simulation process and internal variables a tit-for-tat strategy with a special combination of topology/variables has been used. In its case, each node has been provided with 10 CPU slots and time request has been set to 1. So that the CPU reserve only lasts

one round, so that the response to each request only depends on the strategy and not of available resources. Debug logger messages has been also activated to show the one node decision chain in order to control internal variables.

The expected results of it experiment, verifies that all topologies simulated (See section 5.1.) finish with 50% of cooperators and defectors. This is logic because as first round the nodes can choose with 50% Defect or Cooperate, next round neighbor nodes make the same, and chain of decisions come invariably to 50% results. Force all nodes C or all nodes D on first round, also come to 100% cooperations or 100% defections.

Other simulations has been used to make a idea about how much information exists on a given time or what are the behavior of nodes round to round. Chart is a example that shows the quantity of information flows thought simulations.

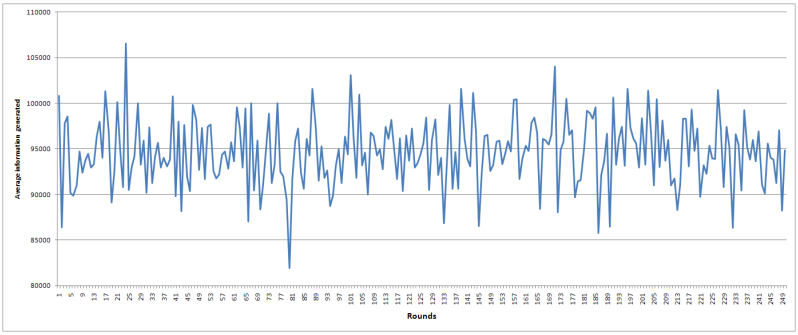

Figure 5.1: Evolution of information shared on a given simulation

5.2.2.2. Topologies verification

Topologies verification consists on transform BRITE topologies used into Otter and Pajek valid topologies and compare the expected properties with the achieved properties calculated. On next graphic (See 5.2) is possible to compare the distribution of nodes degree over two topologies with 35 degree value fixed, one random and other heavy tailed, both generated as described in section 3.2.1.. Color values represents degrees and bars the number of nodes with these degree.

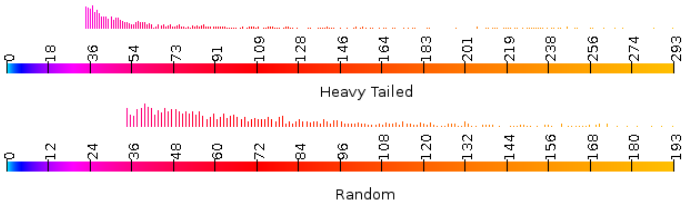

Figure 5.2: Degrees distribution on Heavy Tailed and Random topologies

As can be see, Barabasi modeled topology follows exactly as we has expected, generating a Power law degrees distribution from 35 (the number of nodes fixed) to some final value. Random, however, have a more constants nodes distributions along close values of fixed nodes degree. Also, the dispersion of degrees are much lower.

5.3. Strategy influence

Game strategy used by nodes influence the result of cooperation in different manner that mades the topology. Next results cannot be comparable to other papers or studies presented, as each author uses a different definition of Tit-for-tat and Area Equilibrium strategy has been invented explicitly for this project. But both are able to compare a memoryless one-to-one strategy with a one-to-many strategy over non played versus memory nodes.

Graphic 5.3 represents the percentage of requests full achieved on 250 iterations (last 50 average data) on 1000 nodes topologies with 1 to 3 CPUs slots function of the topologies degrees comparing two different games and topologies. Graphic 5.4 shows the same simulations on 1 to 6 CPUs slots by node.

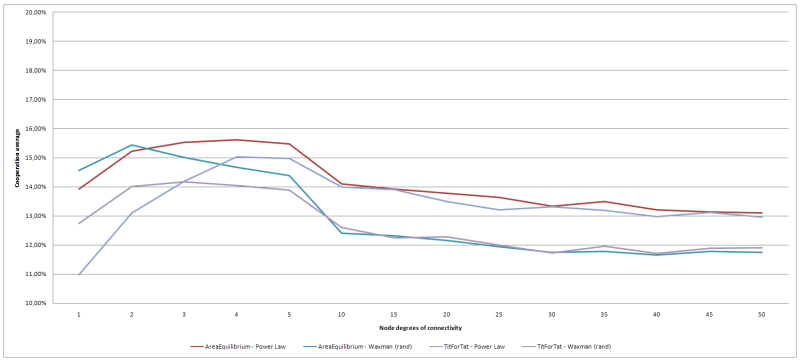

Figure 5.3: Percentage of cooperation using 1 to 3 CPU slots function of nodes degree

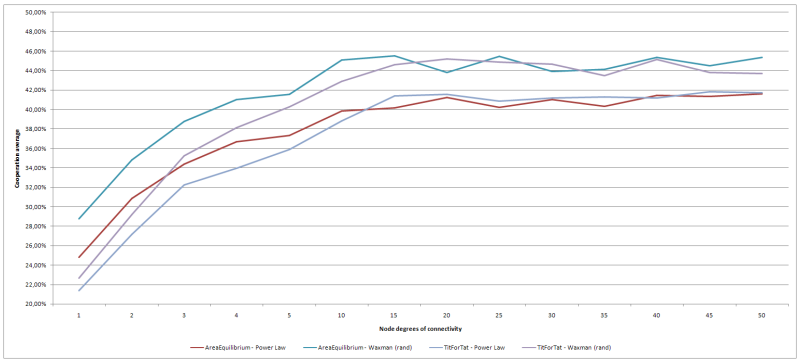

Figure 5.4: Percentage of cooperation using 1 to 6 CPU slots function of nodes degree

The most important conclusion that can be draw from these graphs is that cooperation percentage achieved depends of both parameters. For topologies degree below some number of nodes, the trend is marked by game, contrary as occurs upper value. It is the expected result because area equilibrium game takes into account all of its linked node to make a decision, when tit for tat decides one to one. As a result, the behavior of both strategies are more similar when the number of degrees per nodes are more limited.

5.3.1. CPU slots importance

But, one of interesting unexpected result is that strategy/topology cutoff not are equal on both graphs. While with a 6 maximum CPU slots, the importance of topology beat the game importance immediately 3 maximum CPU slots simulations shows a strategy superiority since almost 5 connection degrees. In this way, the importance of maximum CPU slots has been studied.

Other important information give by previous commented graphs is the performance achieved between them related to the number of maximum CPU slots configured by node. On figure 5.3, that uses values from 1 to 3 the number of requests achieved has been reduced in comparison of figure 5.4, where the maximum value has been doubled. On graph 5.5, where relation between 35 degree topologies, games and maximum CPU slots has been drawn; is possible observe that the number of slots of nodes is closely related to their performance. This is not directly related to strategies, but yet with our particular game: if node have more CPU, is more possible that can be share slots with others.

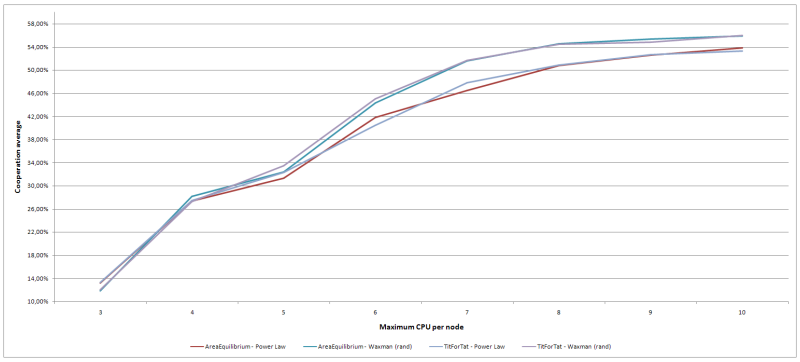

Figure 5.5: Percentage of nodes cooperation function of maximum CPU slots assigned to each node

5.3.2. High CPU slots drawbacks

But increase the number of slots of CPU seems to be beneficial for cooperation, really is rather good for collaboration. So, more CPU implies that all nodes reduces their requests to other since fewer times are not satisfied with their own resources. As a result, the sacrifice nodes make is lower, so nodes do not need a relationship as close as they would if the amount of total CPU was more limited.

Figures 5.6 and 5.7 are good examples of this relation: on theirs, a simulation of 250 rounds with different topologies and strategies has been realized over 1000 nodes topology changing the topologies degree over 3 and 6 maximum CPU slots. Y axis represents the relationship between the amount of resources that a node shares to himself and others achieve from it, so upper 100% percentage implies that node uses more CPU by itself than are sharing. Although both graphics shows the same tendency to goes to 50% division of resources where the number of neighbors of the topologies increases, exists two important differences on there.

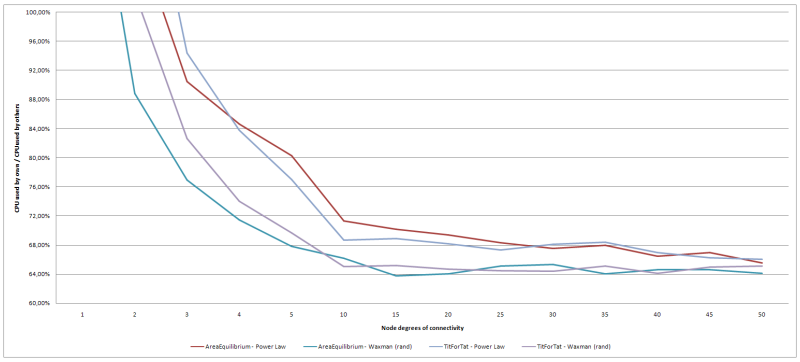

Figure 5.6: Relationship between own used and others used CPU using 1 to 3 CPU slots function of nodes degree

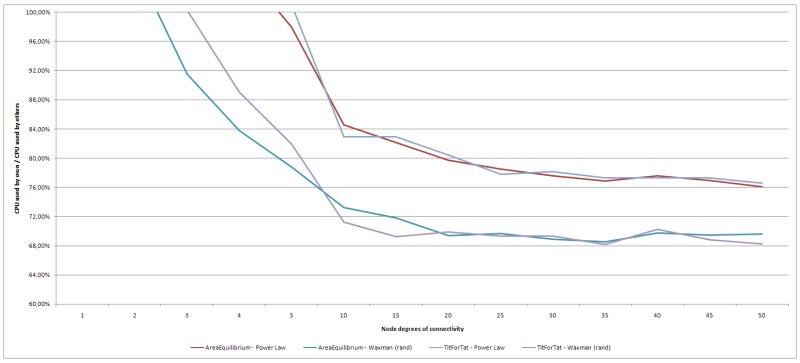

Figure 5.7: Relationship between own used and others used CPU using 1 to 6 CPU slots function of nodes degree

First all, when nodes have more CPU slots the difference of requests achieved between topologies are doubled (from 3.5% to 7%) and on second term, the average of CPU relation (CPU slots used by own divided by CPU slots used by others) moves away the equilibrium point when nodes CPU slots increases. So, exist a trade off in CPU selection between cooperation and fairness. In any case, seems more realistic that in real world the number of maximum CPU slots is wider; so second graphic is more significant.

5.4. Topology influence

As already shown up from a value of connectivity, the amount of resources obtained from the nodes no longer depends on the strategy used, but the network topology (See 5.3 5.4). This is important because it allows us to corroborate the results discussed by Pacheco and her colleagues, according to which it is possible to ensure a certain level of cooperation using the topology as an incentive method.

5.4.1. Random topology vs. free-scale

Try to respond why random topology is better of free-scale on a concrete game is a easy job using the designed simulator. Results of graphic 5.5, the comparison of a concrete nodes degree topology under different CPU slots configured tell that exists a high dependency between resources achieved and topology used. If the number of resources decreases, heavy tailed topology goes below tit for tat, and if number of total resources increases random achieve best results.

This difference is easy to show comparing 5.3 and 5.4, where below some degree value topologies become more important that strategies, but in different order. While the number of CPUs are from 1 to 3 power law distribution achieve more resources and, on the other case (when CPU are from 1 to 6) randomly. But, in any case, results of second set of figures (See 5.6 and 5.7) tit-for-tat demonstrated that achieves a more fairness result.

5.4.2. Node placement influence

Other important cut that is possible to make with the simulator is the relationship between initial topology used and statistical results by node. This allows to study not only the whole tendency of the system, but also the influence of nodes positioning over the game and strategy. Figure5.8 shows the percentage of cooperation achieved by each node ordered by clustering coefficient defined earlier.

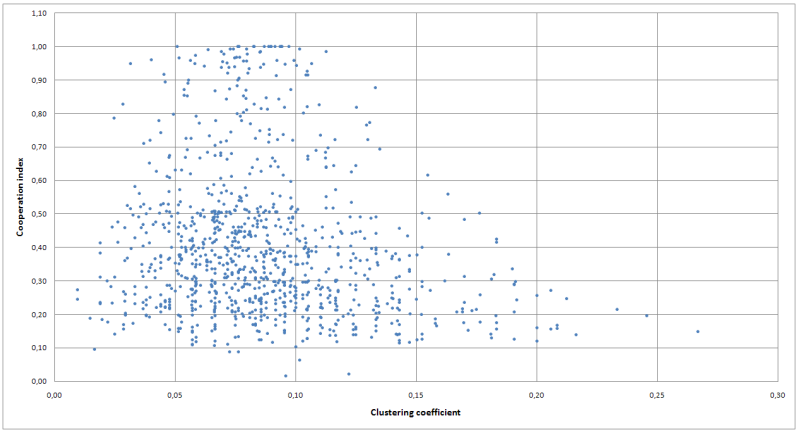

Figure 5.8: Cooperation achieved vs. clustering index

The study of similar graphs as the presented with different configuration of nodes degree, games and topology shows seamless results with it. Major part of nodes that achieve a good cooperation index, in it case upper to 50% are located on cluster coefficients between 0.05 and 0.1, same values where are located the major part of nodes. This is not casual, a too high clustering coefficient implies that the cooperation achieved depends on many nodes, while lower clustering coefficients not assure sufficient resources to connected nodes. Finally, these findings have added importance since they are perfectly comparable to the studies described in the chapter Basic Concepts, that compare cooperation and topologies over tit-for-tat strategies played in PD framework scenario.

5.5. Protocol specification

One of the most costly aspects of this project was the design of the algorithm of the game. In return, once satisfied that the game works correctly and that different strategies can be applied without implementation restrictions, it is possible to extrapolate a negotiation protocol from simulator algorithm. Based on ContractNet [31] [32], next communication protocol has been proposed for the implementation of real application.

5.5.1. XML Messages

As defines negotiation protocol “contract net” each action that two persons make on a simple negotiation must be implemented as a unique message. Even if the use of XML standard for defining messages is not in any way optimal and their use in a real implementation will be require serialization/encoding, it is used in order to simplify the message structures. Also, XML is a descriptive language, so their interpretation easily to do.

Messages has been divided into two types: negotiation and informational messages. The protocol will be described in section 5.5.2. but next, all messages are described:

- Call for proposal (*Negotiation message*): a call for proposal is a announcement message send by a node that need one or more resources by a given time. It message includes their contact information and requirements detail.
- Propose and accept proposal (*Negotiation message*): propose and accept proposal are, in fact, the same message send in different part of the conversation. Both message indicates the part of a agreement partially satisfactory and the part that not. The difference is that while first (Propose) is a participant response to initiator, the other (Accept proposal) is a finally negotiation terms send by initiation at the end of negotiation process.
- Refuse and reject (*Negotiation message*): there are equivalent full disinformation messages send one by participant, other by initiator.
- Failure and communication error (*Informational message*): this information message reports to other that exists some problems on communication. First message (Failure) means that the sender node tries to perform an action and this has launched a unexpected error, while second (Communication error) inform that last received message are incomplete, corrupted or understood. This kind of messages are important in order to avoid communication misunderstandings.
- Done and cancel (*Informationa messagel*): done and cancel messages are negative informational message that communicates to receiver that: or a job has finished ok (Done) or a job must be stopped by some reason (Cancel) the interpretation of their information corresponds to concrete algorithm implemented, but the intention is to separate a logical and expected finishing actions to non expected.

5.5.2. Communication process

Finally, next figure 5.9 and 5.10 shows a draft of negotiation process proposed in VRIMS scenario from learned lessons of simulator. Their functioning, explained below, is based on internal structure of developed simulator.

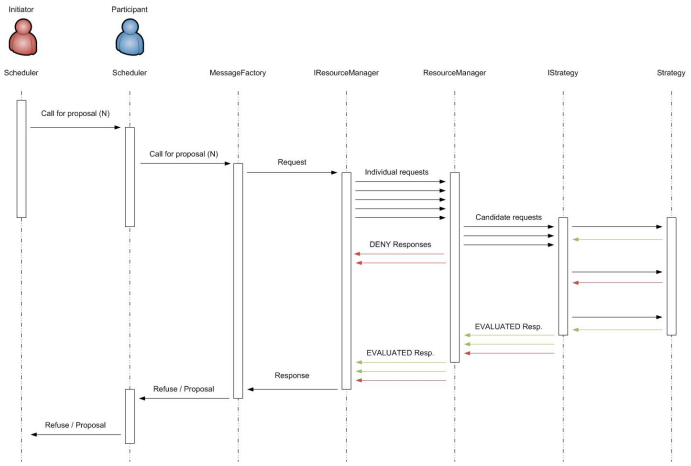

Figure 5.9: Communication request process based proposal for VRIMS scenario

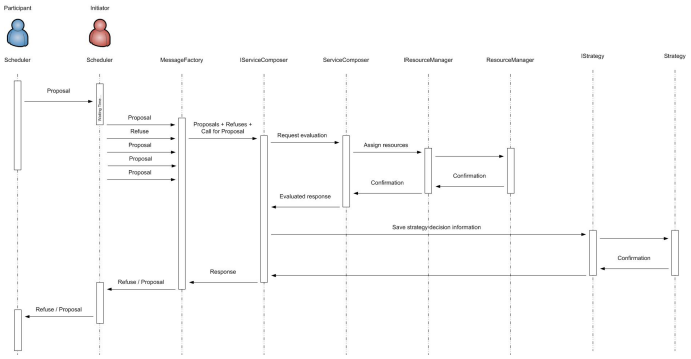

Figure 5.10: Communication response process based proposal for VRIMS scenario

Request of a concrete set of resources is attended individually by a *IResourceManager* that set to refuse impossible to achieve request (for example because the participant not have this kind of resources) and send the rest of requests to *Strategy interface implementation*, the intelligent module responsible to evaluate the prisoner's dilemma game and respond C or D to each unitary requests. Finally, response message will be constructed by resource manager interface implementation and send to initiator.

When initiator received the response, makes similar action: split the response into unitary yes/no responses and *Service composer implementation* tries to known is possible to perform the requested action or not with all the resources achieved. In function of positive and negative responses and if the minimum requirements are achieved response to all nodes with the final conditions.

Finally, is important to note that proposed algorithm is non-aggressive algorithm, so all the responses always have the same or less conditions to previus request/proposal message.

CHAPTER 6. CONCLUSIONS

Although many of the results that are intended to explore in this project have been treated exhaustively in the previous chapter, this final chapter will make a summary of the objectives achieved during the master thesis. Concepts, skills and lessons learned will be identified in order to evaluate the whole results. Also, the environmental impact, not directly but yet indirectly, of this project and its derivatives will be discussed.

Finally, future work related directly to this project, conclusions that take into account, and the simulator development and VRIMS new objectives will be explained, in order to trace a road map for the next years.

6.1. Simulation lessons learned

The aim objective of this project, make a simulator to improve the cooperation between different topologies and strategies over a fixed defined game, has been achieved. So, as a result, we have a simulation toolkit that accomplishes all the requirements proposed in section 4.2.2., but exists some aspects that can be improved and other, better than we expect. Next two subsections explain these.

6.1.1. Improvements

Application developed have a high performance tested over a virtual machines with 10.000 nodes and 50 neighbor by node simulations. But, in order to simulate huge networks, one of initial conditions will be changed: simulator must be use multiprocessing. This allows not only a better performance, also allows it to run simulations over distributed computing system.

However design process has been a very important decision point and I consider that although the time dedicated to it has been more than adequate, maybe spend some more time on research about how other simulation operates and some simulation theory the time spend into design and debugging would have been reduced. Anyway, the knowledge acquired will be very useful in future improvements or developments.

6.1.2. New communication protocol

Not taking into account cooperation game and topology results, surely the best result of this project is the design and realization of a negotiation protocol usable with multiple games and scenarios. Draft of the proposal only explain the request/response process of resources by a concrete node, but is sufficient to verify their functioning on the simulator. In fact, all internal algorithms of internal message queering has been developed using a simplest version of it protocol.

So, using created protocol we are able to create a proof of concept application in order to

test it over real scenario.

6.2. Cooperation game and topology

Previous section demonstrates that exist a tradeoff between topologies, nodes placement, concrete game and strategies decided to use in order to achieve the best cooperation results. But, some times the improvement in one item (as example fairness point) implies directly some drawbacks on other (as average cooperation) Listed below, are all the conclusions extracted from Chapter 5.

- If nodes topology has few links, the cooperation results depends only of the strategy played, but in contraposition if nodes topology increases the cooperation depends only of the concrete network topology. So, in conclusion is possible encourage the cooperation without incentives, only selecting an appropriated topology.
- Cooperation behavior are directly related to the number of resources that each node share to the rest. As a result, so important are large nodes as small. But, after some quantity of CPU slots per node the percentage of CPU achieved by each node decreases because all are self-served. This is no the problem, because in this scenario a grid computing sense not is necessary.
- In same way, on proposed game scenario the topology that achieve greater results (of fairness and cooperation) when depends of their are Tit-for-tat.
- Nodes needs some clustering index to achieve good results. Sufficient to have connectivity to 5% - 10% of nodes, but not much in order to not depends of too others. Is important to note that any loan must be returned if nodes want re-order resources

6.3. environmental

Direct environmental impact not exists on this present project, but is possible justify the advantages of simulation applications in order to reduce the impact of deploying a non verified infrastructures unnecessary. Their environmental costs (energy consumption) not is imputable since will be really deployed and at that time, the verification process (so, the time of system is working and consuming) will be lower.

Moreover, is true that exists a environmental indirect cost related to thesis writing and software implementation, but is not comparable to the savings achieved.

6.4. Future work

This work is only the beginning of a series of research works related to the project underlying VRIMS. So, future work can be related to many topics.

6.4.1. Middle term work: simulator next steps

At middle term, the simulator must be transform on a powerful toolkit to study cooperation over topologies and strategies. Once this goal, several features to directly transform simulations in real applications will be included. Also, a new GUI (*Graphical User Interface*) to control and manage many of variables, simulations conditions and new statistical process also will be made. But, the more important new feature to develop are a internal statistical module that avoid Excel (or other spreadsheet program).

6.4.2. Long term work: VRIMS

As long term work related to it, the new simulator must be tested over thousand of conditions (networks, games and strategies) in order to transform the outlined protocol into a complete negotiation algorithm implementable on VRIMS project. Of course, the final objective to create a grid resource sharing tool will be accomplished.

6.5. Personal conclusions

Personally, this project has meant an enriching experience both academically and professionally. All new foreground, related to applications on distributed networks, methods of cooperation, network topologies characterization and the different tools used, I value them very highly. In same way, facing the design of a non typical software application as a simulator from scratch has been a major challenge.

Finally, some other skills have improved as UML design or the use of information sources academic (research articles and specialized databases are some examples).

BIBLIOGRAPHY

[1] Rodríguez-Haro, Fernando; Freitag, Felix; Navarro, Leandro; Brunner, René. Exploring the Behaviour of Fine-Grain Management for Virtual Resource Provisioning. 7th International Conference on Parallel Processing and Applied Mathematics. *PPAM' 2007, Poland* Gdansk, Poland: Ed. 2007.

[2] DMIS. (2010, February 18) In: *DMIS Wiki project page*. [On line] Project description [Retrieved: February 13, 2010]. Available at: <https://code.ac.upc.edu/projects/dmis/>

[3] Gordon E, More. Cramming more components onto integrated circuits. *Electronics*. 1965, vol. 38, no. 8, pp. 3-7.

[4] Intel Corporation. Excerpts from A Conversation with Gordon Moore: Moore's Law. In: *Intel Corporation* [On line] USA: Intel Museum, 2005 [Consult: May 29, 2010]. Available at: <ftp://download.intel.com/museum/Moores_Law/Video-Transcripts/Excepts_A_Conversation_with_Gordon_Moore.pdf>

[5] Tang, Jia; Zhang, Minjie. An Agent-based Peer-to-Peer Grid Computing Architecture: Convergence of Grid and Peer-to-Peer Computing. In: In Proc. Fourth Australasian Symposium on Grid Computing and e-Research. *AusGrid 2006, Hobart, Australia*. Australia: Buyya, R [et al.] Ed., 2006, pp. 33-39.

[6] Collaboration. (2010, May 26) In: *Wikipedia, the free encyclopedia* [On line] Wikipedia article [Retrieved: May 30, 2010]. Available at: <http://en.wikipedia.org/wiki/Collaboration>

[7] Cooperation. (2010, May 26) In: *Wikipedia, the free encyclopedia* [On line] Wikipedia article [Retrieved: May 30, 2010]. Available at: <http://en.wikipedia.org/wiki/Co-operation_(evolution)>

[8] Jacob, Bart; Brown, Michael; Fukui, Ketaro; Trivedi, Nihar. *Introduction to Grid Computing*. 1st ed. USA: Redbooks, 2005. ISBN 0738494003.

[9] Stallings, William. *Data and computer communications*. 8th ed. USA: Prentince Hall, 2007. ISBN 0-13-243310-9.

[10] Stoica, Ion; Morris, Robert; Karger, David [et al.] Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: ACM. *SIGCOMM 2001, San Diego, USA*. USA, 2001. pp. 149-160.

[11] Barabási, Albert-László; Bonabeau, Eric. Scale-Free Networks. *Scientific American*. 2003, vol. 288, no. 2, pp. 50-59.

[12] Fass, Craig; Turtle, Brian; Ginelli, Mike. *Six degrees of Kevin Bacon*. 1st ed. NY: Plume, 1996. ISBN 9780452278448.

[13] Réka, Albert; Barabasi, Albert. Statistical mechanics of complex networks. *Reviews of modern physiscs*. 2002, vol. 74. n. 1. pp. 47-97.

[14] Lieberman, Erez; Hauert, Christoph; Nowak, Martin A. Evolutionary dynamics on graphs. *Nature*. 2004, vol. 433. n. 1. pp. 312-316.

[15] Santos, F. C.; Pacheco, J. M. Scale-Free Networks Provide a Unifying Framework for the Emergence of Cooperation. *Physical Review Letters*. 2006, vol. 95. e. 098104.

[16] Levin, Dave; LaCurts, Katrina; Spring, Neil; Bhattacharjee, Bobby. BitTorrent is an Auction: Analyzing and Improving BitTorrent's Incentives. In: ACM SIGCOMM Computer Communication Review. *ACM 2008, Seattle*. Washington, USA. Ed 2008.

[17] León, Xavier; Vilajosana, Xavier; Brunner, René [et al]. Information and Regulation in Decentralized Marketplaces for P2P-Grids. In: 16th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises. *WETICE 2008, Rome*. Italy. Ed. 2008.

[18] Santos, F. C.; Rodrigues, J. F.; Pacheco, J. M. Graph topology plays a determinant role in the evolution of cooperation. In: Procedings in the Royal Society. *Royal Society 2005, UK* Australia: FirstCite e-publishsing, 2005.

[19] Lozano, Sergi; Arenas, Alex; Sánchez, Angel. Mesoscopic Structure Conditions the Emergence of Cooperation on Social Networks. *PLoS ONE*. 2008, vol. 3. n. 4. e. 1892.

[20] Langera, Philipp; Nowaka, Martin A.; Hauerta, Cristoph. Spatial invasion of cooperation. *Journal of Theoretical Biology*. 2008, vol. 250. n. 1. pp. 634-641 .

[21] Concurrent computing. (2010, May 30) In: *Wikipedia, the free encyclopedia* [On line] Wikipedia article [Retrieved: June 6, 2010]. Available at: <http://en.wikipedia.org/wiki/Concurrent_computing>

[22] Gamma, Erich; Helm, Ralph; Johnson, Ralph; Vlissides, John M. *Design Patterns: Elements of Reusable Object-Oriented Software*. 2nd ed. USA: Addison-Wesley, 2000. ISBN 0-20-163361-2.

[23] Alexandrescu, Andrei. *Modern C++ Design: Generic Programming and Design Patterns Applied*. 9th ed. USA: Addison-Wesley, 2004. ISBN 0-20-170431-5.

[24] What is log4cxx. (2008, March 31) In: *Apache Software Foundation* [On line] Library description [Retrieved: June 6, 2010]. Available at: <http://logging.apache.org/log4cxx/index.html>

[25] Doxygen. (2010, April 24) In: *Doxygen homepage* [On line] Dimitri van Heesch description [Retrieved: June 6, 2010]. Available at: <http://www.stack.nl/~dimitri/doxygen/>

[26] Gawk. (2009, July 24) In: *Gawk homepage* [On line] README information [Retrieved: June 6, 2010]. Available at: <http://www.gnu.org/software/gawk/>

[27] Medina, Alberto; Lakhina, Anukool; Matta, Ibrahim; Byers, John. *BRITE: Universal Topology Generation from a User's Perspective*. User Manual. USA: 2001.

[28] Medina, Alberto; Matta, Ibrahim; Byers, John. On the Origin of Power Laws in Internet Topologies. In: ACM Computer Communications Review. *ACMCC 2000, San Diego, USA* USA, 2000. pp. 30-42.

[29] Huffaker, Bradley; Nemeth, Evi [et al] Otter: A general-purpose network visualization tool. In: ISOC Inet. *ISOC 1999, USA* USA, CAIDA: Cooperative Association for Internet Data Analysis, 1999.

[30] Visualization of Large Networks. Reference Manual List of commands with short explanation (2010, May 18) In: *Wikipedia, the free encyclopedia* [On line] Reference manual [Retrieved: June 6, 2010]. Available at: <http://vlado.fmf.uni-lj.si/pub/networks/pajek/doc/pajekman.pdf>

[31] FIPA Contract Net Interaction Protocol Specification. FIPA (2002, March 12) In: *Foundation for Intelligent Physical Agents* [On line] Standard SC00029H [Retrieved: June 14, 2010]. Available at: <http://www.fipa.org/specs/fipa00029/SC00029H.pdf>

[32] FIPA Communicative Act Library Specification. FIBA (2010, March 12) In: *Foundation for Intelligent Physical Agents* [On line] Standard SC00037J [Retrieved: June 6, 2010]. Available at: <http://www.fipa.org/specs/fipa00037/SC00037J.pdf>