POLITECNICO DI MILANO
SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

TESI DI LAUREA MAGISTRALE IN COMPUTER SCIENCE AND ENGINEERING

GATOR

A GAME THEORY APPROACH TO RESOURCE ALLOCATION FOR HETEROGENEOUS CPUS

Author:
Dott.ssa Lara Premi
Student ID:
841714

Supervisor:
Prof. William Fornaciari

Co-Supervisor (Correlatore):
Dott. Federico Reghenzani

Co-Supervisor (Correlatore):
Ph.D. Giuseppe Massari

A.Y. 2019/2020

A Raffaella,
ovunque tu sia.

Contents

List of Figures

List of Figures

List of Tables

Acknowledgements

T. S. Eliot ha detto *Quello che conta è il percorso del viaggio e non l'arrivo.*

Se qualcuno me lo avesse fatto presente qualche anno fa, in corsa verso la laurea ancora lontana, probabilmente avrei storto il naso perché, diciamocelo, l'obiettivo di chi inizia ad andare all'università è il fantomatico pezzo di carta per cui ogni giorno ci alziamo dal letto, impazziamo sui libri, inseguiamo professori nel campus e cerchiamo di barcamenarci fra lezioni costantemente sovrapposte.

Eppure, a distanza di una vita intera da quando ho varcato la soglia del Politecnico di Milano per la prima volta, sono convinta che, senza il tragitto che ho effettuato, senza tutti gli scossoni che ho dovuto affrontare, senza tutte le vittorie che ho potuto vivere, non sarei dove sono ora, una ragazza diventata donna grazie, in particolar modo, all'appoggio delle persone che mi hanno fatto compagnia in questa avventura.

I primi che cito in questi ringraziamenti sono i miei genitori che, oltre ad avermi garantito le comodità di un tetto e l'accoglienza di una famiglia durante il periodo di studi, sono riusciti a sopportarmi più di quanto avrebbero dovuto fare: grazie che ancora non mi avete cacciata di casa per insofferenza.

A mio fratello Gabriele, per il silenzio che a volte condividiamo, per le risate che non guastano mai e per i passaggi in auto verso la stazione: se non fosse stato per te, manco avrei presieduto all'*Open Day*.

Acknowledgements

A tutti i miei ex compagni di corso, quelli con cui ho legato particolarmente, quelli che non ho avuto l'occasione di conoscere bene, chi è arrivato qualche anno più tardi, chi ha iniziato insieme a me, coloro con i quali ho litigato e coloro che ancora tollerano la mia presenza: grazie, per aver condiviso con me le esperienze in cui siamo inciampati all'ateneo. Fortuna che, cadendo, non ci siamo fatti troppo male!

Alle amiche "poche ma buone" che il destino mi ha fatto incontrare nei modi più disparati, Chiara P., Susy, Erica, Giulia e Chiara T.: mi siete sempre state accanto, mi siete accanto ora e lo sarete anche domani, vi adoro.

A Walter e Marta che accettano i miei isterismi nel pubblico e nel privato senza dire mai niente: so che prima o poi mi tirerete il collo, ma vi ringrazio anche di non averlo ancora fatto.

Al professor William Fornaciari, ai correlatori Giuseppe e Federico, alla famiglia intera dell'*HeapLab*: mi avete adottata quando sono rimasta orfana, a voi devo davvero tutto, grazie.

Da ultimo menziono la persona che sono solita bistrattare, l'unica che critico sempre e in ogni modo possibile, io, che, nonostante tutto e dopo tutto, ce l'ho fatta, da sola, fino alla fine. Grazie.

Abstract

In literature, several state-of-the-art works exploited Game Theory as an approach to solve the resource allocation problem in distributed systems. In this thesis, we propose to use this theory also for heterogeneous platforms with different types of computational units. By creating two resource allocation policies based on congestion games and double auctions games, and by integrating them on *BarbequeRTRM* software, we show their benefits in terms of performance and other metrics, such as energy and power. We check the above-mentioned policies with a simulation and an experimental evaluation phases: in the former, we study their execution time and their solution optimality from a theoretical standpoint; in the latter, a set of metrics, including power and energy, are measured directly on a real system. We run well-known benchmarks to explore a large set of scenarios. From the results of the experimental campaign, we discuss how to tune the parameters of the two policies and their advantages.

Part of this work has been accepted for publication in the *Embedded System Week 2020* proceedings [23].

Abstract (in italiano)

In letteratura, molti lavori dello stato dell'arte hanno sfruttato la teoria dei giochi come approccio per risolvere il problema di allocazione delle risorse nei sistemi distribuiti. In questa tesi, proponiamo l'utilizzo di questa disciplina matematica anche nelle piattaforme eterogenee con differenti tipologie di unità computazionali. Attraverso la creazione di due politiche di allocazione delle risorse basate sui *giochi di congestione* e sui *giochi a doppia asta* e grazie alla loro integrazione sul software di *BarbequeRTRM*, mostriamo i loro vantaggi in termini di performance e di altre metriche, come l'energia e la potenza. Testiamo le suddette politiche con una fase di simulazione e una fase di valutazione sperimentale: nella prima, studiamo il loro tempo d'esecuzione e l'ottimalità della loro soluzione da un punto di vista teorico; nella seconda, un insieme di metriche, tra cui potenza ed energia, vengono misurate direttamente su un sistema reale. Abbiamo esplorato numerosi scenari, utilizzando benchmarks ben conosciuti. Dai risultati della campagna sperimentale, abbiamo discusso della tecnica per regolare i parametri delle due politiche e dei loro vantaggi.

Parte di questo lavoro è stata accettata per la pubblicazione nei proceedings della conferenza *Embedded System Week 2020* [23].

CHAPTER 1

Introduction

1.1 Heterogeneous computing

Heterogeneous computing studies the systems that contain different types of computational units, for example, multi-core *CPU*s, *GPU*s, and *FPGA*s. The interest in this branch of computer science increased because the performance of the single-core reached its limit due to the frequency barrier. It is no more possible to increase the clock frequency while maintaining the power consumption, and consequently, temperatures at acceptable levels. This is also linked to the end of the Moore's and Dennard's laws [9]. So, in the last fifteen years, the architecture moved towards a multi-core structure to raise the performance with more computational units involved, rather than focusing on the single-core performance.

During the most recent years, there was an explosion of heterogeneous architectures (as shown in Figure 1.1). These machines contain processors or computational units of several typologies instead of the classical multi-core that has homogeneous, i.e. identical, cores. With such architectures, diverse applications behave differently depending on the archi-

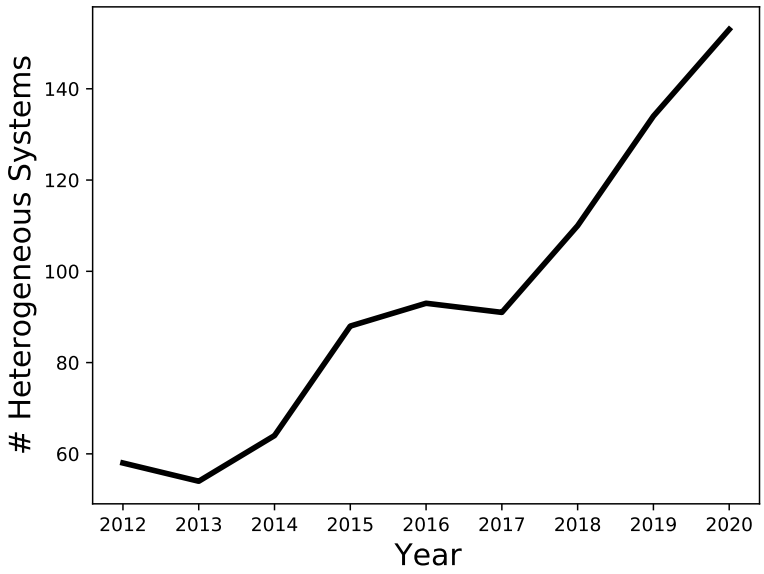

Figure 1.1: *The increasing trend of the use of heterogeneous resources in HPC systems (the data have been retrieved from the TOP500 list).*

tecture type they run on: for example, a highly parallel application can be executed on a *GPU* to achieve better performance because the graphic units have several computational units; instead, in case of an application with a low number of threads, it is preferable the usage of a *CPU* which guarantees high speed on the single-core.

In addition, to reach better performance, the utilization of heterogeneous architectures allows us to explore the trade-off according to non-functional metrics, that are, for example, the power consumption, the energy consumption, and the reliability.

1.1.1 High Performance Computing

Given that the heterogeneous architectures enable efficient management concerning power and energy, they are slowly gaining space also in supercomputing centers. Supercomputers belong to the so-called *High-Performance Computing* that includes all the technologies used to create systems that can provide very high performance, in the order of *petaFLOPS*. For example, in the *TOP500* list[1], the current most powerful supercomputer is *Japanese Fugaku* with a *PEAK* computational power of

[1] `https://www.top500.org/`

415.53 *petaFLOPS*.

During the last years, several scientific projects dealt with the performance of the *HPC* systems and, in particular, the techniques used to reduce the power and the energy consumption that currently represents the obstacle to the growth of the computation capacities of these systems. One of the European projects is the *Horizon 2020 MANGO* that is described in the following subsection as an example of a deeply heterogeneous architecture.

1.1.2 An example: the MANGO architecture

The *Horizon 2020 MANGO* project [3] [10] [11] [12] [26] focused on a hardware architecture that tries to explore deeply heterogeneous accelerators in *HPC* systems running multiple applications with different *Quality of Service* (*QoS*) levels. The main objectives are related to improving the power, the performance, and the predictability of the *HPC* system. With these purposes underlined, the project analysed different and interrelated mechanisms from the hardware architecture to the system software layers.

The infrastructure of *Horizon 2020 MANGO* is a distributed system, featuring computational nodes composed of general-purpose processors, *GN*, linked via *PCI Express* or *Ethernet* with a set of heterogeneous accelerators, *HN*, interconnected through a *Network-on-Chip*, i.e. *NoC*. We can see a simple graphic explanation in Figure 1.2 where a node of the *HPC* is divided into two parts: the homogeneous and the heterogeneous sections. On the left, there is the server (*GN*) in which one or more sockets, i.e. physical processors, are composed of some identical cores, while on the right we have one or more *FPGA*s that are connected to each other with the *PCI Express* bus. The *FPGA*s synthesize different virtual processors with one or more cores, linked, through the external *PCI Express* bus, to the *GN* server. This scenario requires the management and allocation of resources among different applications in a way that maximizes resource usage, while preserving the predictable execution time of critical applications, and the expected power budget [24]. The same architecture is currently used in another *Horizon 2020* project, *RECIPE* [2] [1] [13], with the goal of reaching the exascale performance, i.e. a supercomputer with at least one *exaFLOPS* of computational power.

To explore heterogeneity, *MANGO* project developed different com-

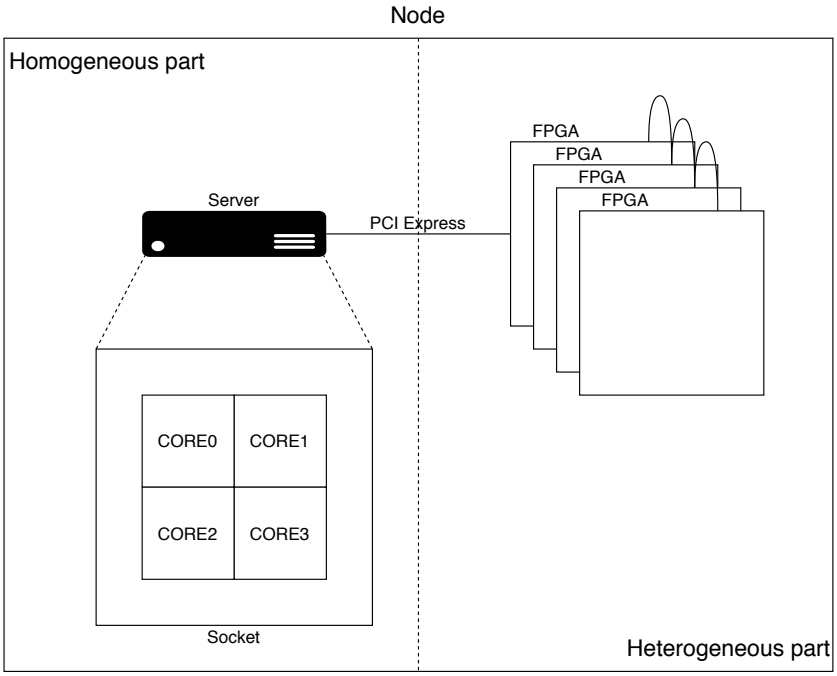

Figure 1.2: *Overall architecture of the Horizon 2020 MANGO platform.*

puting units, referred to as *UNIT*. One of these *UNIT*s is *PEAK* that stands for *Partitioned Enabled Architecture for Kilocores*: it is a research many-core prototype for generic computing. The term "many-core" refers to processors with tens or hundreds of homogeneous cores, rather than the few cores of the traditional multi-core. The main goal of *PEAK* is to offer a configurable processor able to be adapted to different configurations and capabilities, thus enabling exploration of adaptations to the different target applications in the project. *NU+*, another processor architecture used in *MANGO*, is a complex and configurable *GPU*-like accelerator core, allowing flexible customization driven by application requirements. It is designed to support the exploration of advanced architecture features not available in current general-purpose heterogeneous architectures. To clearly explain how the *FPGA* has been configured, the above-mentioned specifications are depicted in Figure 1.3, where it is possible to observe the structure of one *FPGA* in which we have, in addition to *PEAK* and *NU+*, the custom accelerators (in green). These parts of the *FPGA*, differently from the other two processors, are programmed for a specific goal and they perform only a single non-programmable computation. It

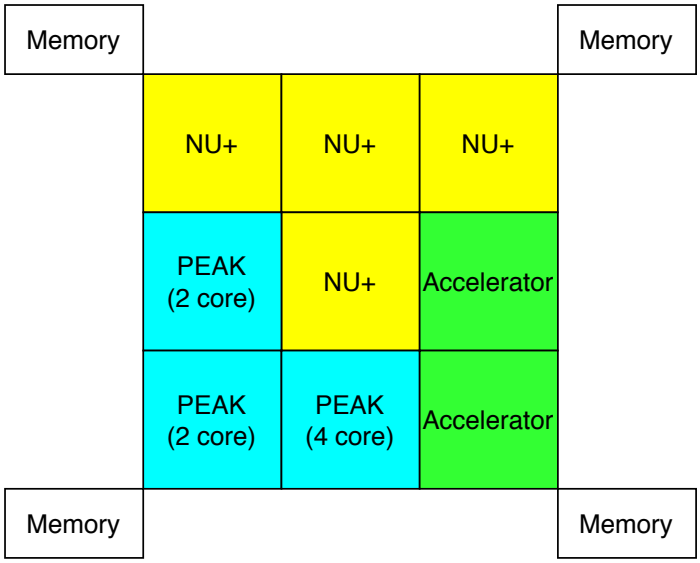

Figure 1.3: *The architecture of one FPGA used in the Horizon 2020 MANGO platform.*

is possible to change it only by reprogramming the entire *FPGA*, but this requires a non-negligible amount of time.

The presence of such heterogeneous and complex architecture raises the following questions: where to allocate the tasks of a given application and how to choose this specific processor. To perform this selection and answer these questions, we need a resource manager, able to take such decisions according to the aforementioned trade-off and to enforce the operating system to schedule the applications on the correct resources.

1.2 The Resource Management problem

As illustrated in 2012, in the paper of Bellasi et al. [4], the growing complexity of computing architectures (e.g. heterogeneous multi-core, several layers of memories, *Network-on-Chip*, etc.) requires a system-wide arbiter that manages all of these computing resources. For example, mobile systems, e.g. smartphones and tablets, have become powerful enough to run computationally-intensive applications like advanced multimedia, augmented reality, and 3D gaming. On a completely different scale, i.e. network and cloud computing, the availability of a large number of resources poses the same problem of mobile platforms: how to manage

such resources. Furthermore, also specialized embedded systems are increasing in complexity to allow the system designers to implement a system feature as a software task instead of a hardware module, gaining in this way flexibility, diminished costs, and reduced time-to-market. However, exploiting these features is non-trivial [25]: to get the maximum performance from parallel architectures requires a substantial design effort. In these complex architectures, there are many objectives and, so, the optimization problem becomes more difficult to manage.

In this whole scenario, the *Run-Time Resource Manager* (*RTRM*) acquires a lot of relevance to reach the performance goal of the system. The resources and the tasks can change, e.g. a node in *HPC* may become not available or a new application requires to be executed; so, there is the need for adaptivity and reconfigurability of the system. This suggests the necessity of having a *RTRM* in modern computing platforms.

1.2.1 Barbeque Run-Time Resource Manager

The problem of allocating a certain number of applications on a certain amount of resources, where both of them have different characteristics, is an allocation problem which can be classified in the *NP-Complete* class of computational complexity, i.e. it is not known if a simple solution computable in polynomial time exists or not. Currently, any algorithm that finds the optimal solution requires an exponential time concerning the input size, making it, by nature, not executable in a short time.

As mentioned in [4], the *BarbequeRTRM* software is a framework for run-time resource management that supports both homogeneous and heterogeneous platforms. Implemented in *C++*, it contains a series of mechanisms to interact with the *Linux* kernel, for example, the allocation of resources is performed via the *cgroup* interface. The "smart" component of *BarbequeRTRM* is the resource allocation policy that decides which application is assigned to which resource. In this thesis, we propose an assignment procedure for resources and tasks implemented by *BarbequeRTRM* and based on the Game Theory.

1.3 Introduction to Game Theory

Game Theory is the branch of applied mathematics that studies the mathematical models concerning the interaction among entities considered as rational decision-makers. In this way, we can represent situations of everyday life, called *games*, in which two or more persons:

- can interact by following some precise rules;
- reach at some point in time a final state that depends on the choices of all the entities, after a finite number of steps;
- have preferences on all the possible outcomes of the given games, i.e. each person usually prefers, for some reason, a set of outcomes with respect to other sets.

The entities taking part in a game are called *players* and they must have two main characteristics to be used in a Game Theory analysis:

- they must be rational in front of, for example, the rules understanding of the expression of logical preferences about the outcomes, i.e. no irrational choices are allowed;
- they must be egoist, i.e. they want to maximize her/his utility function and consider other players' preferences only to analyze their own choices.

The options that a player chooses in a setting where the outcome depends not only on her/his actions but also on the actions of others represent the *strategy* of a player: it determines the action which the player takes at a specific stage of the game. The strategy should not be confused with the move the player performs at a given point during the play of the game (called *action*). The strategy is, instead, the complete algorithm for playing the game, composed of all the possible actions that a player can perform for every possible situation throughout the game. The main goal of the Game Theory is to find the so-called *Nash Equilibrium*, that is a situation in which no one of the players has an incentive to play a different strategy if the others are playing the equilibrium strategy. This is not necessarily the global optimum solution, however, it is the solution that does not allow a single player to unilaterally improve its position at the expenses of the other players.

1.3.1 Historical overview

The first known discussion of Game Theory occurred in a letter in **1713** attributed to Charles Waldegrave, where he provides a minimax mixed strategy solution (a classical Game Theory problem formulation) to a two-person version of a card game: this problem later becomes known as Waldegrave problem. However, it was only in the 20th century that the Game Theory studies advance. Without the aim to carefully describe each detail of the research, we summarize in the following paragraphs the main discoveries and algorithms related to Game Theory. A complete history can be found in [19]. The reader who wants to learn more on this topic can read the related textbooks and the specific scientific works.

In **1913**, Ernst Zermelo demonstrated that the optimal chess strategy is strictly determined. In **1927**, Émile Borel showed a minimax theorem for two-person zero-sum matrix games that provides a solution to a non-trivial infinite game and he conjectured the non-existence of mixed-strategy equilibria in finite two-person zero-sum games. This conjecture would have been later proved false. In fact, in **1928**, John Von Neumann published a paper [31] in which he used Brouwer's fixed-point theorem on continuous mappings to show the existence of the mixed-strategy in finite two-person zero-sum games. The second edition of his book [32] co-authored with Oskar Morgenstern provided methods for finding mutually consistent solutions for two-person zero-sum games. These works became the basis for the so-called mathematical economics.

Merrill M. Flood and Melvin Dresher proposed in **1950** the Prisoner's Dilemma experiment that will have become the standard example in Game Theory. In this thesis, it is described in the next Subsection 1.3.2. The next year, in **1951**, John Nash developed the concept of *Nash Equilibrium* as previously defined. He proved that any non-cooperative, finite n-player, non-zero-sum game has a *Nash Equilibrium*.

In the **1950**s and **1960**s, the Game Theory has been extensively developed and many theorems and results have been carried out by the scientific community. From the **1970**s on, the Game Theory has also been applied to other fields, starting from biology and economics. Nowadays, it is used in several fields, including sociology, military, law, computer, and telecommunication sciences.

Table 1.1: *The classical formulation of the Prisoner's Dilemma.*

A/B	B stays silent	B deceives
A stays silent	(-3,-3)	(-5,0)
A deceives	(0,-5)	(-4,-4)

1.3.2 The Prisoner's Dilemma

The Prisoner's Dilemma is the traditional example used in Game Theory to show why two players, when considered as completely rational individuals, do not necessarily decide to cooperate, even if it seems that it is advantageous for them to find a shared solution. The schema of this game is reported in Table 1.1.

Two criminals are arrested and imprisoned separately. They have no way to communicate with each other to, for example, exchange any sort of information. They can decide between two choices: testifying that the other committed the crime or remaining silent. The effects are the following:

- if both A and B decide to testify against the other, both of them serve four years in prison;
- if A deceives B but B remains silent, A will be set free and B will stay five years in prison (and vice versa);
- if both A and B remain silent, both of them will serve only three years in prison.

Another necessary assumption is that the prisoners have no possibility to reward or punish the other player after they have been both set free. Thus, their decision is considered as not affecting outside scenarios not related to the game. Considering that deceiving a partner offers a greater reward than cooperating with them, all purely rational self-interested prisoners would decide to testify that the other committed the crime. In this way, we note that pursuing individual reward logically leads both of the prisoners to deceive when they would get a better individual reward if they both kept silent. This is a perfect example of a *Nash Equilibrium*: it is not a globally optimal solution, but it does not allow any player to unilaterally improve its position.

1.3.3 Classification of games

Several classifications and typologies of games exist. This section presents the most famous ones:

- A **cooperative game** is a game in which players can form groups and take part in it. Coalitions are consequently created. On the other hand, the **non-cooperative game** is characterized by players that play alone, one against the other, without any sort of group.
- In a game with **perfect information**, each player knows the possible moves of the others, before they play them, and (s)he has all the information to compute the utility functions of the other players. However, most of the games have, on the contrary, **imperfect information**.
- A **symmetric game** is a game in which, based on the payoffs for playing a particular strategy, the focus is not on who plays the other strategies, but the real crucial point is on the content of the other strategies themselves.
- A **synchronous game** expects a global timing system which the players follow together during their moves; on the other hand, in an **asynchronous game**, the time does not play a role in the game and the players can take arbitrary time to select the strategy.
- A **repeated game** consists of a fixed repetition of some *stage game* that is a well-studied two-person game: in this case, the consequences of the current action of one player impact on the future moves of the other players.
- The **Stackelberg Competition** is a strategic game in which, firstly, a leader chooses his/her move and, then, the followers, after watching the play of the leader, decide how to deal with it, choosing, so, the better move for them.
- In combinatorial optimization, the **knapsack problem** o **rucksack problem** considers a set of items, each characterized by a weight and a value, in which the goal is to determine the number of each item because it is necessary to include them in a collection where

the total weight is less than or equal to a fixed limit and the total value is as large as possible.

- A **selfish routing game** is represented by a directed connected graph in which a lot of flows traverse it at the same time. In this case, we have a *source* and a *destination*, two nodes where the flows begin and finish respectively; between them, there are also other nodes thanks to which we have some edges identified with a specific non-negative value.

The problem of resource allocation can be both cooperative and non-cooperative, based on the fact if the applications talk to each other or not, or if there is a central entity. Furthermore, our issue is practically convertible into a knapsack problem: this is the cause of its *NP*-completeness. It is usually not perfect, because models of applications and resources are not always perfect, thus making it impossible to accurately compute the utility functions. In the resource management of networks (that is not a subject of this thesis), the problem is easily modeled with the last presented class, i.e. the selfish routing game.

1.3.4 Identification of possible algorithms for resource management

In this thesis, we focused on two games that, in our opinion, best fit our problem of resource assignment. Before presenting them, we describe the subsequent four different games that we identified as possible candidates for the application to our problem.

- **Simultaneous-action games.** As their name suggests, these games imply that the involved players take their actions at the same time: a famous example is the *Rock-Paper-Scissors* game. This is interesting for the resource management problem because the applications have to be allocated all at the same time, thus it is not a game that "progress" in turns (like a card game).

- **Coordination games.** In these particular games, all the players have the same utility function for a given strategy. This can be used in our problem to give to the application a fair share of resources; however, it is difficult to model heterogeneous platforms.

- **Congestion games.** In these games, each player selects one or more resources to use, and his/her expected payoff depends on the resources (s)he and the other players choose.
- **Double auctions.** They are seen as a process that involves some users (buyers) which want to buy an item and some users (sellers) which have the item and want to sell it.

Among them, in this thesis, we considered the last two game classes. Their applicability to the resource management problem is explained later in the relative sections.

1.4 Thesis contributions and structure

1.4.1 Contributions

Differently from the *State of Art* in Chapter 2 where the Game Theory is applied to the allocation problem between resources and tasks in distributed systems, in this thesis we use it in a single and heterogeneous system. During the simulation phase, we check the complexity of the algorithms. We tested them on a real platform that simulates a heterogeneous system, focusing on their side effects on the metrics, i.e. energy, power, and execution time.

1.4.2 Structure

After this chapter that is an overview of Game Theory and the *BarbequeRTRM* software, in Chapter 2 we present the *State of Art* related to the scientific papers about the two main topics of this thesis: Game Theory and resource management. Then, in Chapter 3, first of all, the congestion game policy is described, beginning from a brief introduction of the game, and by continuing with our model that maps the assignment problem as a congestion game. The algorithm is then presented and followed by its complexity computation, the pseudo-code, and the cost function description. The second part of this chapter is dedicated to the double auctions scenario where, like the previous section, we can observe a little introduction of this game, the description of the model with two proposed algorithms, their comparison, and the mechanism thanks to which we decided to calculate the bid of every player involved. In Chapter 4 we

discuss the simulation of the two scenarios chosen, firstly the congestion game and secondly the double auctions, to measure the execution times and compared them. Hereafter, there is Chapter 5 in which we show the implementation of the two scenarios from the Game Theory in the *BarbequeRTRM* software. Finally, before Chapter 7 designated for the conclusions of the thesis, the experimental evaluation is presented in Chapter 6 to learn how to tune their parameters and to judge their effectiveness in solving the resource management problem.

CHAPTER 2

State of the Art

Resource management and Game Theory are not novel concepts, as we have seen in the previous chapter. However, this thesis proposes a novel use of game-theoretical algorithm to the problem of resource allocation. Before presenting the novelty of our approach, we recap in the following two sections the works available in literature regarding the current policies implemented in *BarbequeRTRM*, and the state-of-the-art works for resource management based on Game Theory.

2.1 Available policies in BarbequeRTRM

Before the introduction of the new two policies presented in this thesis based on Game Theory, *BarbequeRTRM* already had several policies implemented. During the years, researchers have implemented different policies with different goals. In this section, we show only the policies currently activable in the *BarbequeRTRM* configuration menu and not the legacy ones. *BarbequeRTRM* implements a set of "toy" policies, i.e. that are used only for testing purposes:

- **Random.** In a random way, it assigns the *CPU* resources to the several tasks.

- **Test.** It assigns a constant resource chosen at compile-time by the developer.

- **TempBalance.** It simply assigns the tasks to the processor with the lowest temperature.

Besides these trivial policies, other smart algorithms are available:

- **Tempura.** Some researchers showed that this policy sets an upper-bound for the *CPU* usage based on the temperature, and partially on the energy, of a *BIG.little* processor, i.e. an heterogeneous processor that has only two typologies of cores [27].

- **Contrex.** This policy has the same name of the European project in which has been developed [14]. It allocates the resources depending on a subdivision already set up: some resources are dedicated to critical applications and others to non-critical ones. *Contrex* has no optimizations relative to the energy, the power or other non-functional metrics.

- **Perdetemp.** Massari et al. [20] showed that this policy allows to follow the run-time variable performance requirements of the applications, saving power consumption and limiting the occurrence of thermal hotspots: in this way, it reduces the aging effects of the computing resources, firstly, by improving the overall system reliability and dependability, and, secondly, by saving also cooling costs.

- **Manga.** Another work [21] shows that, in a heuristic-based way, this policy predicts the best resource mapping solutions for each application, such that the resource manager can quickly pick them at run-time, by also limiting the additional overhead caused by *BarbequeRTRM*. As target, it has a deeply heterogeneous system, with interconnected processors via *Network-on-Chip*, like in the *MANGO* platforms.

2.2 Resource management based on games

In this section, we present the current available works that exploit Game Theory for the resource allocation problem. Please note that is not our goal to discuss the whole State of the Art of Game Theory, but just the works similar to the problem we are trying to solve.

2.2.1 Cooperative games

Dong et al. [8] showed that energy accounting, i.e. how much a software contributes to the total energy consumption of a system, can be formulated as a cooperative game. Using a prototype based on, for example, smartphone workload, the authors showed that their approximation, by using the Shapley value, is better than the existing policies if the final objective is to manage the system energy. Furthermore, another work [22] introduced a two-level hierarchical game to reduce the server load and to enlarge not much the total computational time.

2.2.2 Non-cooperative games

Some researchers modelled with a Stackelberg game the problem of minimizing energy consumption: if the system monitor, the *leader*, maximizes profit, scheduler agents, the *followers*, select resources. It is used a non-cooperative game among decentralized *followers*, considering that some how they play against each other in the resource sharing: the target is to minimize the average response time. In this way, besides the enhancement of energy efficiency, we have the improvement of reliability and robustness of the system [34].

Ye et al. [35] suggested the usage of non-cooperative games amongst *VM*s on cloud computing environments. After the introduction of two different types of resource allocation games, the *Nash Equilibrium* is found: through the calculation of the *Price of Anarchy* and the *Price of Stability*, the authors studied its inefficiency.

Another work [29] introduced a new policy regarding the cost and the allocation of the resources. Following and balancing the constraints about budget and deadline, users can predict the price playing a game with incomplete information. The experimental results showed that there is an equilibrium state where the cost converges in a gradual way.

Some researchers focused on the design of an allocation problem between users, modelled as a potential game. In the end, the authors proved that there is at least one *Nash Equilibrium* and we are able to find it through an allocation mechanism [17].

Vanderster et al. [30] used the knapsack problem as a model for resource allocation, introducing a group of such policies. After the analyses, we can see that the performance of this approach results better.

Finally, concentrating on the selfish routing game, another work [6] allocated tasks by changing the basic architecture and demonstrating the obvious improvement in the response time and the server speed.

2.2.3 Studies on the comparison of cooperative and non-cooperative games

Some researchers proposed a game-theoretic solution to the problem of distributed resource allocation, especially in an emerging cloud platform, by studying two resource allocation games, non cooperative and cooperative games. The explored games include repeated and asynchronous ones. The authors analysed the interaction among the entities in the cloud environment: the virtual machines are allocated according to the output of the Game Theory algorithm. The authors discovered that non-cooperative games are not optimal, for example in terms of total utility and cost effectiveness, compared to cooperative games [16].

Xu et al. [33] proposed a finite extensive game with perfect information to distribute resources with two objectives, honesty among users and employment of entities like *CPU* and memory. Comparing the created algorithm with two existing ones, we can see that, with the proposed method, there is a better and more efficient resource allocation.

In addition to suggest the usage of cooperative and non-cooperative games to model resource allocation in cloud computing, for the same objective another work [18] proposed imperfect, symmetric and asymmetric games: the authors showed that, in this case of study, the issues about security are ignored.

2.3 Our contribution with respect to the State of the Art

All the work previously described in Section 2.2 applied the Game Theory to distributed systems, e.g cloud datacenters. To the best of our knowledge, no works tried to use Game Theory algorithms to the problem of managing heterogeneous resources in a single computing, by modelling the applications as players of a game.

The selection of congestion games and double auctions games (already presented in Section 1.3.4) implies using non-cooperative approaches. Applications are not necessarily ready at the same time, making difficult the use of cooperative games where the players have to take together a decision. Moreover, this would have required the integration of all the applications to a common framework, while a competitive approach allows us to use any application, even not integrated with the *BarbequeRTRM* framework. These are the reasons for the selection of non-cooperative algorithms. The analysis of cooperative policies is left as future work.

CHAPTER 3

Game Theory policies

The goal of this chapter concerns, firstly, the description of the congestion game scenario, the model that we used for the new policy based on this Game Theory approach, and the cost function we purposely designed for this optimization problem. On the other hand, in this chapter, our objective is to present the double auctions scenario, not only with a mere description, but also by describing how we model the resource allocation policy with this Game Theory strategy. Specific expressions for the "bid mechanism" have been derived and discussed.

3.1 Congestion games

In the class of **potential games** where a single global function can express the incentive of all players to change their strategy, there are the so called **congestion games**, used to model a situation in which the users compete for the use of some given resources.

In this game, the possible actions a player can choose are subsets of resources that can include an arbitrary number of elements. The central

component of this type of game is the **congestion**, a value which represents the number of users assigned to a specific resource.

In general, resources have a cost that depends on the congestion: in fact, the cost function of a given resource has the congestion of the resource as input parameter and it returns the related cost. Since the congestion may assume, in the original formulation, a finite set of discrete values, the cost function can be represented as a vector where, in the first position, we find the cost when the congestion is 1, in the second position, we find the cost when the congestion is 2, and so on.

In the end, we can compute the **potential function** ϕ for the whole game that is given by:

$$\phi(a) = \sum_{j=1}^{m} \sum_{k=1}^{cong_j(a)} c_j(k)$$

where a is the strategy of the players, m is the number of resources, $cong_j(a)$ is the congestion of resource j given a, and $c_j(k)$ is the cost of resource j for congestion k. The next section presents the model of our problem as a congestion game, and we will formally specify all of these variables.

3.1.1 The allocation policy

By considering our problem focused on the distribution of resources to applications, i.e. the main purpose of the *BarbequeRTRM* software, we now introduce a resource allocation policy implemented as a congestion game. The objective of our study is to assign computational resources to the applications, by taking into account the optimization of their performance and the achievement of, e.g., the power, the frequency, the energy and the temperature.

3.1.2 Problem modelling

Assuming that all the threads, belonging to the same application or different ones at a time, correspond to several tasks[1], we recognize them with the set $\mathcal{T} = \{t_1, t_2, ..., t_n\}$ that identifies the **players** of a congestion

[1] By depending on the context, *thread*, *task*, and *process* are three terms often used with different meaning. In this thesis we consider *task* the same concept of *thread*, i.e. a set of executable instructions running sequentially on a single computational unit.

game. The set of **resources**, instead, $\mathcal{R} = \{r_1, r_2, ..., r_m\}$, represents the computational units available in the system. The set of possible **actions** is defined as:

$$\mathcal{A} = \{a_1, a_2, ...\} = \{(t_1, r_1, x_1), (t_2, r_2, x_2), ...\} \tag{3.1}$$

where x_i is the percentage of usage of the i-th resource. Please note that the presence of x_i distinguishes our approach from a traditional congestion game, where, usually, the resource is always used at 100% percentage. In our case, a **strategy** $\mathcal{S} = \{a_1, a_2, ...\}$ is composed of a subset of actions that describes the final allocation solution, i.e. which resources are assigned to which task and with what percentage associated. Basing on how many tasks use a precise resource with a specific percentage, that is a number which, by assumption, coincides to an integer value of the congestion related, the congestion will be updated during the development of the game. The congestion function for a given strategy $\mathcal{S}$ is formally defined as:

$$cong_i(\mathcal{S}) = \sum_{\forall j} 1_{\mathcal{S}}((r_i, a_j))$$

where $1_{\mathcal{S}}$ is the indicator function, i.e. $1_{\mathcal{S}}((r_i, a_j)) = 1$ if $(r_i, a_j) \in \mathcal{S}$ or $1_{\mathcal{S}}((r_i, a_j)) = 0$ if $(r_i, a_j) \notin \mathcal{S}$. The cost that a task pays for the usage of a resource, in addition to its dependency on the value of its congestion, is related to the basic characteristics of the resource, e.g. the power, the frequency, the energy and the temperature. The cost function will be later described in Section 3.1.4 and we provide here only the formal abstract function definition:

$$c_i : cong_i(S) \to \mathbb{Q}^+ \tag{3.2}$$

The cost function is then a function parametrized on the i-th resource, with the congestion as domain (input) and a positive rational number as codomain (output).

3.1.3 The algorithms and their complexity

The congestion game problem can be solved by using two different algorithms, an exact and an approximated.

Algorithm 1 The exact algorithm.

Require: Congestion Game G

Ensure: Nash Equilibrium $\mathbf{s}$ of G

1: $\mathbf{s}^0 \in \mathbf{S}$
2: $k \Leftarrow 0$
3: **while** $\mathbf{s}^k$ is not a Nash Equilibrium **do**
4: **if** a_i of player $t_i \in S$ and $\pi_{t_i}(t_i, \mathbf{s}_{-t_i}) < \pi_{t_i}(\mathbf{s})$ **then**
5: $\mathbf{s}^{k+1} \Leftarrow (a_i, \mathbf{s}_{-t_i})$
6: $k \Leftarrow k + 1$
7: **end if**
8: **end while**
9: **return** $\mathbf{s}^k$

The exact version mentioned is shown in Algorithm 1. In Line 1 a strategy $\mathbf{s}^0$ is chosen in a randomized way to initialize the game. Then, in Line 3, a **while** cycle starts if the considered $\mathbf{s}^0$ is not a *Nash Equilibrium*: in our code, to verify if the *Nash Equilibrium* has been reached, we control the loop with a boolean variable that checks if the strategy has been modified or not in the current cycle. If the strategy has not been changed, then a *Nash Equilibrium* has been found and the algorithm terminates. Please note that a game does not necessarily have only one *Nash Equilibrium*: it may be possible to have more than one equilibrium, that are equivalent with respect to the cost function of the congestion games. For this reason, only the first encountered equilibrium is returned by our algorithm.

The **if** condition at Line 4 checks the cost of the current game and the one in which we take an action a_i of the t_i player and we put it in the initial strategy, substituting the old one of that player and creating, in this way, a new strategy. The cost of the newly created game is computed: if the cost of the new game is smaller than the other, we change $\mathbf{s}^{k+1}$ with the new game's strategy and the **while** cycle is repeated.

Instead, the approximated version of the algorithm is different from the exact one because of the presence of a variable, i.e. ε, that is subtracted to the cost of the current game in the **if** condition: this is shown in Algorithm 2. The ε coefficient is a positive value that represents how much we move away from the exact solution: in fact, the larger it is, the

Algorithm 2 The approximated algorithm.

Require: Congestion game G

Ensure: Nash Equilibrium $\mathbf{s}$ of G

1: $\mathbf{s}^0 \in \mathbf{S}$
2: $k \Leftarrow 0$
3: **while** $\mathbf{s}^k$ is not a Nash Equilibrium **do**
4: **if** a_i of player $t_i \in S$ and $\pi_{t_i}(a_i, \mathbf{s}_{-t_i}) < \pi_i(\mathbf{s}) - \varepsilon$ **then**
5: $\mathbf{s}^{k+1} \Leftarrow (a_i, \mathbf{s}_{-t_i})$
6: $k \Leftarrow k + 1$
7: **end if**
8: **end while**
9: **return** $\mathbf{s}^k$

more the solution is approximated.

By taking into account the exact procedure shown in Algorithm 1, we calculate its complexity starting from the **while** cycle which, based on the fact that we identified with n the amount of tasks and with m the number of resources, costs $\mathcal{O}(m^n)$. This is because, in the worst-case, it is the same of exploring a tree with n levels and m branches.

Given that we expanded the block of code defined by the **if** lettering with a lot of functions and methods used to simplify the structure of the algorithm, we obtained as complexity a value of $\mathcal{O}(n^3m^3log(n))$. Multiplying the above-mentioned amount with the cost of the external **while** cycle, we had $\mathcal{O}(n^3m^nlog(n))$, that is not different from the complexity of the approximated Algorithm 2: in fact, this version is diverse from the exact one only for the presence of ε. However, ε does not affect the whole complexity because it only reduces the number of cycles by a constant value. Consequently, $\mathcal{O}(n^3m^nlog(n))$ is the complexity of the worst cases in both scenarios. The study on the real amount of execution time required to execute this algorithm will be presented in Chapter 4, where a simulation of different scenarios is performed.

3.1.4 Cost function design

Taking care of Equation 3.2, we have designed a specific cost function for our problem of assigning resources to tasks. Its expression for the i-th resource is:

$$c_i(g) = \sum_{j=1}^{g} \bar{c}_i(j) \cdot \eta_{i,j}$$

where g is the congestion, $\bar{c}_i(j)$ is the cost vector subsequently defined, and $\eta_{i,j}$ is the percentage of the i-th resource used by the j-th task. By using the same notation of Equation 3.1, we can write:

$$\eta_{i,j} = \{x_k | (t_j, r_i, x_k) \in S\}$$

where S is the current strategy. Each element of the cost vector $\bar{c}_i$ is defined as:

$$\bar{c}_i(g) = \begin{cases} \alpha_i + \beta_i + {\gamma_i}^2 & \text{if } g = 1 \\ \alpha_i[n + \alpha_{i,0}(g-1)] + \beta_i[1 + \beta_{i,0}(g-1)] & \text{if } g > 1 \end{cases} \tag{3.3}$$

where

- g is the position in the cost vector that corresponds to the usage percentage of the relative task;
- α_i is the term related to the performance of resource i and it is equal to

  $$\alpha_i = \hat{\alpha}_i \cdot \frac{1}{f_i} \tag{3.4}$$

  in which $\hat{\alpha}_i$ is a coefficient chosen by the user based on his/her need related to the performance and f_i is the frequency of the *i*-th processor. The higher the frequency, the lower the cost because, in this way, the task throughput improves;
- β_i is the term related to the energy of resource i and it is equal to

  $$\beta_i = \hat{\beta}_i \cdot PDP_i \tag{3.5}$$

  in which $\hat{\beta}_i$ is a coefficient chosen by the user based on his/her need related to the energy. *PDP* is the *Power Delay Product*, also

called *Energy Per Operation* (*EPO*): this value represents the average amount of energy required to perform a single computation [28]. The *PDP* of resource i is computed as follows [15]:

$$PDP_i = \frac{P_i}{f_i} \tag{3.6}$$

where P_i is the power consumption of the *i*-th processor and f_i is its frequency;

- γ_i is the term related to the power referred to resource i:

$$\gamma_i = \hat{\gamma}_i \cdot P_i \tag{3.7}$$

in which $\hat{\gamma}_i$ is a coefficient chosen by the user based on his/her needs related to the power and P_i is the power consumption of the *i*-th processor. This coefficient exists only when $g = 1$ (i.e. the cost for the first task) because allocating one or more tasks does not affect the power consumption: the processor is already running even with just one task, thus the instantaneous power consumption is already considered;

- $\alpha_{i,0}$ is the overhead correlated to α_i: if we have more than one task, in order that all of them work with the *CPU*, it is necessary that the operating system executes some context switches (one at a time, a task works a bit, before it is stopped and substituted by the successive task, and so on), which is an operation with its overhead that increases with the number of tasks (in our cost function, it is represented exactly by $\alpha_{i,0}$). Otherwise, with just one task, we have no context switches and, consequently, no overhead;

- $\beta_{i,0}$, similarly to $\alpha_{i,0}$, is the overhead correlated to β_i.

3.2 Double auctions

A double auction is a process in which we have a set of **buyers** and a set of **sellers** with a potential different cardinality: the first ones are interested in buying a single item that all the other buyers are interested in; the second ones have to sell one item that all the other sellers want to sell.

A single trader, the so called **auctioneer**, matches, if possible, every buyer to every seller: a buyer can be matched with, at most, one seller and vice versa because each buyer is interested to obtain a single item and each seller is selling exactly one single item.

Once the auctioneer has matched the buyers and the sellers, the auctioneer must define the amount of money to be transferred from the matched buyer to the corresponding matched seller.

3.2.1 The allocation policy

Considering our problem focused on the allocation of resources to applications, we introduce an optimization policy which models the resource management problem as a double auctions game. The objective of our study is to assign computational resources to the applications, by considering the optimization of their performance and the achievement of, e.g., the power, the frequency, the energy and the temperature goals.

3.2.2 Problem modelling

The specific game studied in this section expects a set of buyers interested to buy an item that is, in our case, a portion of the computational power. We chose the tasks to fill the role of the buyers. On the other hand, as sellers, which have to sell an item, we selected the resources that, someway, sell their computational capabilities to the tasks. To informally summarize, the resources sell their computational power capabilities to the tasks that are looking for a resource to run on it, and, consequently, consume its computational power. Finally, we assumed that the *BarbequeRTRM* software is the auctioneer which matches every buyer to every seller. If we have an initial situation in which the tasks number is greater than the resources number, the existing resources will be doubled or tripled, and so on, based on the amount of tasks (in the end, the resources will equal or surpass the tasks). The duplicated resources will have:

- a second identifier that is an unique number;
- as bid, the result of the product between the bid of the copied resource and a multiplier calculated specifically each time.

Algorithm 3 The McAfee mechanism.

Require: Double auctions game DA

Ensure: Best Assignment **ba** of DA

1: **Order** the n bids of the buyers by decreasing order
2: **Order** the m bids of the sellers by increasing order
3: **for** $k \in [N, 1]$ **do**
4: **if** $b_k \geq s_k$ **then**
5: **break**
6: **end if**
7: **end for**
8: **Calculate** $p = (b_{k+1} + s_{k+1})/2$
9: **if** $b_k \geq p \geq s_k$ **then**
10: The first k buyers and sellers trade the good in price p
11: **else**
12: The first $k-1$ sellers trade for s_k and the first $k-1$ buyers trade for b_k
13: **end if**

3.2.3 The algorithms and their complexity

We can solve this problem by using two different procedures: the *McAfee* algorithm and the *VCG* mechanism. The first method is shown in Algorithm 3. After sorting the resources by increasing order regarding the bids and the tasks by decreasing order regarding the bids, in Line 2 we find the so called **breakeven index**. This value is the largest position in the vectors buyer/seller such that the buyer bid in that k position is greater than the seller bid in the same position. After the calculation of the price as the average of the bids of the subsequent buyer/seller pair (Line 8), at Line 9 there is a simple check that this price is within the current buyer and seller bids: if yes, the first k buyers and sellers buy and sell respectively the item with a price of p. Otherwise (Line 12), the first $k-1$ sellers sell the item with a price of s_k and the first $k-1$ buyers buy the item with a price of b_k.

In Algorithm 4 the second method, i.e. the *VCG* mechanism, is described and it is, in the first part, very similar to the *McAfee* algorithm. At Line 7 there is a sequence of checks between the bids of buyers and sellers, identifying, in this way, the new values defined for the trading: if $b_{k+1} < s_k$ and $s_{k+1} > b_k$, each buyer buys the item with a price of s_k and

Algorithm 4 The VCG mechanism.

Require: Double auctions game DA

Ensure: Best Assignment **ba** of DA

Order the n bids of the buyers by decreasing order
Order the m bids of the sellers by increasing order
for $k \in [N, 1]$ **do**
    if $b_k \geq s_k$ **then**
        break
    end if
end for
if $b_{k+1} < s_k$ and $s_{k+1} > b_k$ **then**
    each buyer pays s_k and each seller gets b_k
end if
if $b_{k+1} < s_k$ and $s_{k+1} \leq b_k$ **then**
    each buyer pays s_k and each seller gets s_{k+1}
end if
if $b_{k+1} \geq s_k$ and $s_{k+1} > b_k$ **then**
    each buyer pays b_{k+1} and each seller gets b_k
end if
if $b_{k+1} \geq s_k$ and $s_{k+1} \leq b_k$ **then**
    each buyer pays b_{k+1} and each seller gets s_{k+1}
end if

each seller sells the item with a price of b_k; if $b_{k+1} < s_k$ and $s_{k+1} \leq b_k$, each buyer buys the item with a price of s_k and each seller sells the item with a price of s_{k+1}; if $b_{k+1} \geq s_k$ and $s_{k+1} > b_k$, each buyer buys the item with a price of b_{k+1} and each seller sells the item with a price of b_k; and, if $b_{k+1} \geq s_k$ and $s_{k+1} \leq b_k$, each buyer buys the item with a price of b_{k+1} and each seller sells the item with a price of s_{k+1}.

From a complexity point of view, both the *McAfee* algorithm and the *VCG* mechanism cost $\mathcal{O}(n \cdot log(n) + m \cdot log(m))$ that is caused especially by the two ordering involved, i.e. increasing ($\mathcal{O}(n \cdot log(n))$) and decreasing ($\mathcal{O}(m \cdot log(m))$), and the worst case in which the number of resources, that is m, is smaller than the amount of tasks, that is n. Instead, on the other hand, if $n = m$, the complexity of both the methods is $\mathcal{O}(n \cdot log(n))$. Finally, if we want to calculate the social welfare (this is not necessary for the finding of the optimal solution), their total complexities

become $\mathcal{O}(n^2)$.

3.2.4 How to choose the bid

To determine the specific bid for the group of the sellers and the whole buyers, we defined two different expressions, according to some metrics strictly connected to the performance goals of our problem.

Sellers

From this point of view, the bid increases in three different cases inspired by the economic model of supply and demand. First of all, it increases when the execution time decreases because better performance is offered: the more the performance, the more the buyers are willing to pay. Secondly, it increases when the power consumption increases because, concerning the non-functional requirements, it is necessary to pay more to use a resource that consumes more power. In the end, if the PDP value decreases, we can have a better energy efficiency, so the amount of the bid reduces for the same reason of the power. Then, our bid for the resources is:

$$b_{S_i} = \alpha_i + \beta_i + \gamma_i \tag{3.8}$$

where α_i, β_i and γ_i are the same parameters explained in Equations 3.4, 3.5 and 3.7. If we consider the case in which there are duplicated resources thanks to the `copyandpaste` method of the double auctions algorithm, we have to introduce a new calculus for the bid. In this case, we have to multiply the 3.8 to two constants, k, that is the second *ID* of the resource, and p, which is the penalty related to the fact that the resource is already used:

$${b_{S_i}}^k = (k+1) \cdot p \cdot b_S \tag{3.9}$$

In this way, we introduced a penalty for selecting the duplicated resources.

Buyers

To calculate the buyers bid, we consider the quantity of work that the task has to execute, by assigning a larger bid to those tasks which have less

work to do. In this way, the tasks with less work will end sooner, by freeing the resources for the other tasks:

$$b_{B_i} = \frac{1}{WL_i} \tag{3.10}$$

where WL_i is the amount of work related to the i-th task that varies by depending on the user choices, e.g. in this thesis we considered the total execution time by calculating it as the average of these execution times measured multiple times for each task.

CHAPTER 4

Simulation of the policies

To check the performance of costs and execution times of the policies from a theoretical standpoint, in this chapter we study the two games. We check how the above-mentioned metrics change for the four different algorithms presented: the perfect and the approximated versions in the congestion games, the *McAfee* procedure, and the *VCG* method in the double auctions.

4.1 Simulation setup

To perform the required simulation we implemented a *C++* program that runs several times the algorithm, measures its execution times, and writes them to a file. We explored for 40 times each couple (n, m) for $n \in [2, 20]$ and $m \in [2, 20]$, where n is the number of tasks and m the resources amount. Therefore, the total number of scenarios tested are 14440[1] for each algorithm. Then, with a *Python* script, we calculated the average of the execution times and plotted the graphs subsequently presented. The

[1] It is the result of: $19 \cdot 19 \cdot 40$.

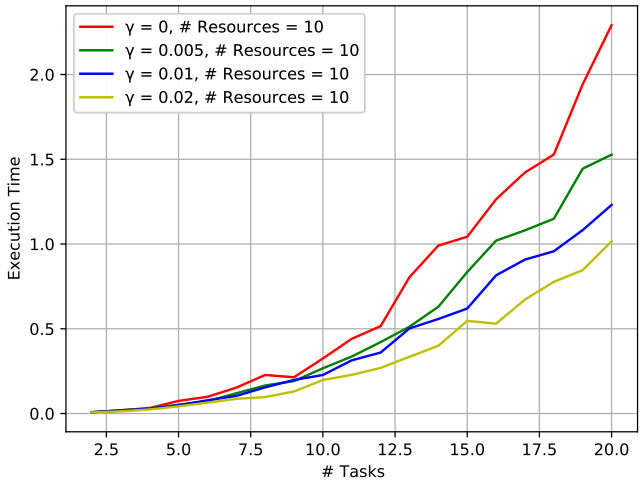

Figure 4.1: *The representation of the policy execution times by varying the number of tasks and the γ values, with a fixed amount of resources, in the case of congestion games.*

plot makes it possible to analyse the time evolution with respect to the number of resources and tasks. From the point of view of the congestion games, we tested the perfect and the approximated versions because we wanted to understand how the execution time and the cost behave. Accordingly, in the double auctions scenario we compared the *McAfee* algorithm and the *VCG* mechanism to see the differences in execution time and the social welfare.

4.2 Congestion Games

Figure 4.1 and Figure 4.2 depict how the execution time changes by varying, firstly, tasks and, secondly, resources. Given that the γ value is a number that we multiply to the cost game of the initial random solution, which is used to find the ε approximation factor, described in Subsection 3.1.3, independently from the problem data. The value of ε is then computed as:

$$\varepsilon = costgame \cdot \gamma$$

We tested four different cases, 0, 0.005, 0.01, and 0.02. The figures show the case with $(10, 10)$ as number of resources and tasks. The growth

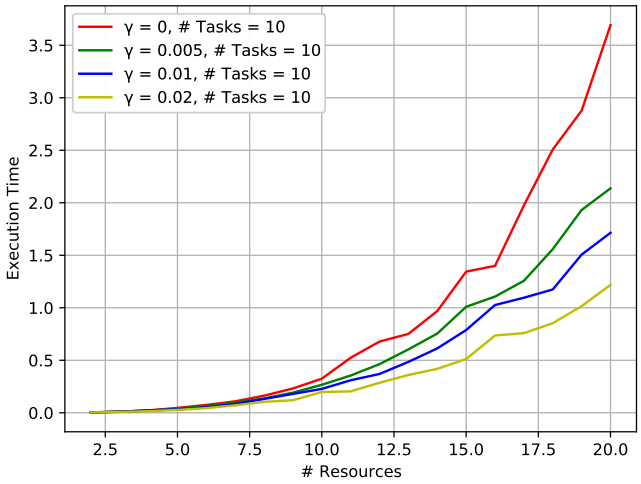

Figure 4.2: *The representation of the policy execution times by varying the number of resources and the γ values, with a fixed amount of tasks, in the case of congestion games.*

of the lines is clearly exponential, according to the tasks in the first image and according to the resources in the second figure. Also in the approximated version, the execution time trend of the congestion game policy remains exponential anyway, but, even with small values of γ, the execution times decrease considerably.

The more γ is far from the 0 value, the more the corresponding line in the graph is far from the one that characterizes the situation in which we reach the perfect *Nash Equilibrium*.

If we look at the *3D* plot (Figure 4.3), we can observe the evolution with tasks, resources, and execution times involved together. By considering the approximated version of the algorithm, if the number of tasks, the amount of resources, and the execution times grow at the same time, the height of the columns increases: the above-mentioned picture explains how they work. Regarding the results on the cost variations, we decided to provide the data in tabular form. The following formula has been used to fill Table 4.1:

$$\frac{cost_{is} - cost_{fs}}{cost_{is}} \cdot 100$$

where $cost_{is}$ is the cost game considering the initial random situation and

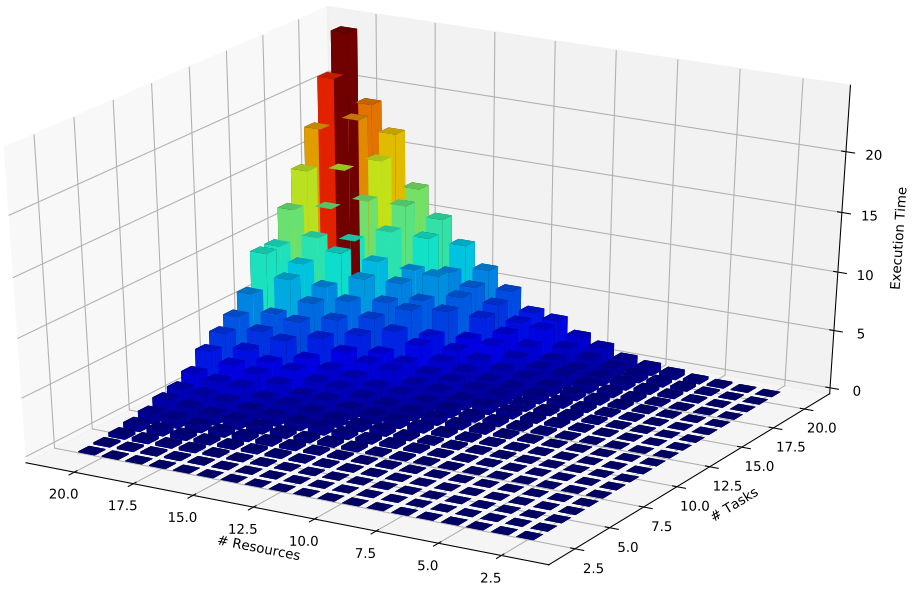

Figure 4.3: *The 3D representation of the policy execution times by varying both the number of resources and the number of tasks, in the case of congestion games.*

$cost_{fs}$ is the cost game considering the new resource assignment given by the policy. The table shows the average percentage calculated considering the four different cases in which γ changes from 0 to 0.02: a part from the couple with 5 resources and 5 tasks that can be considered a sort of outlier, it is possible to observe that, the more γ grows, the less the cost improves, with respect to the random solution, as expected. For example, in the case of 20 resources and 5 tasks, the perfect solution improves the random guess by 21.62%, while, when $\gamma = 0.02$, the improvement is just 6.81%.

4.2.1 Modifications required for the experiments

In order to run this simulation, we had to modify the implementation as follows:

# Resources	# Tasks	Cost with $\gamma = 0$	Cost with $\gamma = 0.005$	Cost with $\gamma = 0.01$	Cost with $\gamma = 0.02$
5	5	$\approx 15.24\%$	$\approx 16.44\%$	$\approx 16.32\%$	$\approx 14.22\%$
5	10	$\approx 16.14\%$	$\approx 15.55\%$	$\approx 13.55\%$	$\approx 10.55\%$
5	20	$\approx 14.92\%$	$\approx 13.69\%$	$\approx 9.95\%$	$\approx 2.23\%$
10	5	$\approx 18.49\%$	$\approx 18.48\%$	$\approx 18.55\%$	$\approx 15.03\%$
10	10	$\approx 19.13\%$	$\approx 18.40\%$	$\approx 15.23\%$	$\approx 5.36\%$
10	20	$\approx 18.95\%$	$\approx 14.77\%$	$\approx 6.43\%$	$\approx 0.16\%$
20	5	$\approx 21.62\%$	$\approx 20.22\%$	$\approx 17.59\%$	$\approx 6.81\%$
20	10	$\approx 21.63\%$	$\approx 17.94\%$	$\approx 6.6\%$	$\approx 0.16\%$
20	20	$\approx 21.70\%$	$\approx 7.07\%$	$\approx 0.11\%$	0.0%

Table 4.1: *Improvement percentages of the costs by using the algorithms, based on congestion games and compared to the initial random solution.*

- We implemented the generation of initial strategies based on a defined number of resources and tasks, that changes according to the needs for all the simulations studied.
- The costs and the percentages are calculated in a random way by using the `rand` method. For this reason, we adopted `srand` at the beginning of the `main`: it extracts random numbers always different for every simulation.
- To create a correct initial strategy, we randomly select the task-resource assignments. We chose a probability of 0.3, so that a task is assigned to a resource with this probability: in this way, we have the guarantee that all the tasks are using at least one resource in the initial strategy.

4.3 Double auctions

In Figure 4.4 we can observe the evolution of the execution times in relation to the growth of the tasks, by keeping the amount of resources fixed. It is clear that, in case of the *McAfee* algorithm, when we have the same number of resources and tasks or when the total amount of resources is greater than the number of tasks, the execution time increment is linear. Instead, every time we have less resources than tasks, considering that, in this case, the double auctions algorithm duplicates the number of resources to cover the gap with the tasks amount, we can notice a step in the lines of the graph coloured by red, blue, and green.

By looking Figure 4.5, we can see the graph related to the *VCG* mech-

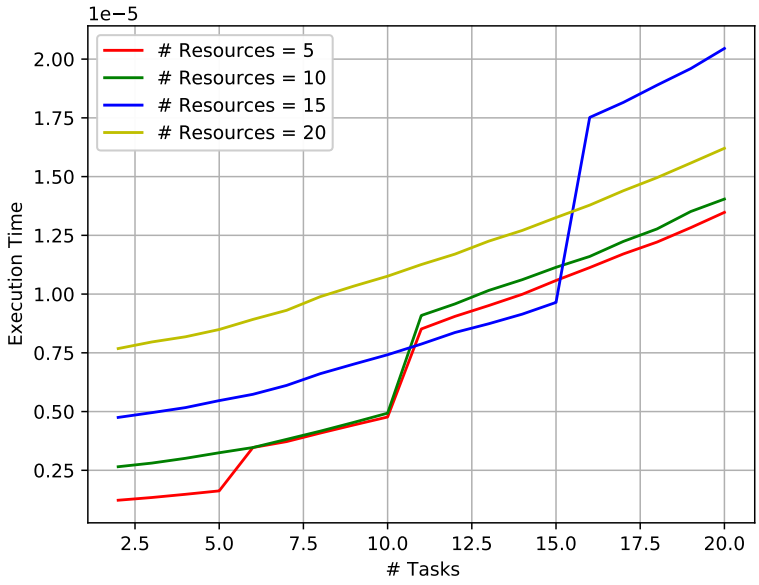

Figure 4.4: *The representation of the policy execution times by varying the number of tasks, with a fixed amount of resources, in the case of McAfee algorithm.*

anism that is very similar to the plot of Figure 4.4, maybe a bit more variable than the representation of *McAfee* algorithm. Regarding the similarity between the execution times, none of the two methods is more convenient than the other. In Table 4.2 we compared the two strategies and highlighted the two social welfare values: it is clear that, with little differences between their resulting values, the *VCG* procedure is slightly better than the *McAfee* one, considering that, in theory, the first method is more economically efficient than the second.

4.4 Discussion of the results

With the exploration of all the possible couples from $(2, 2)$ to $(20, 20)$ we obtained the execution time of the four algorithms that we chose to study, by representing their values on graphs in which it is possible to see their trends with respect to the number of tasks and resources.

In the case of the congestion games, if the amount of resources or the number of tasks increases, we can observe an exponential growth in

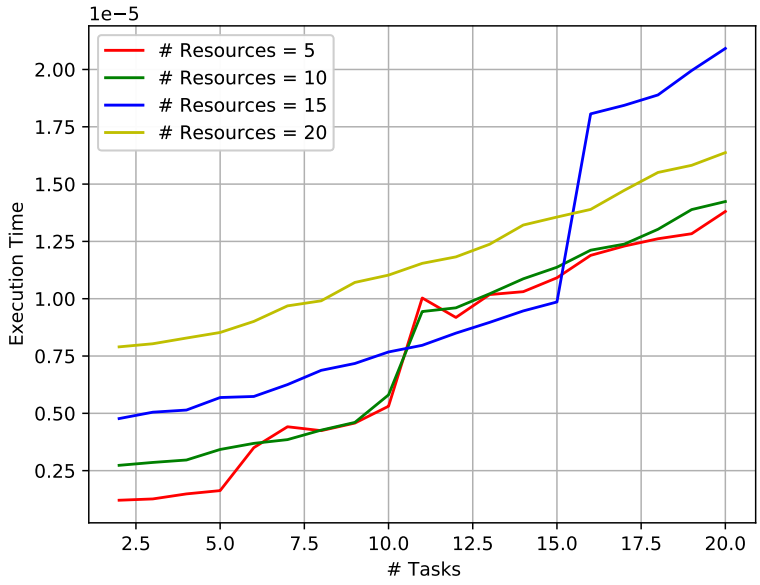

Figure 4.5: *The representation of the policy execution times by varying the number of tasks, with a fixed amount of resources, in the case of VCG mechanism.*

both the execution time and the cost. This is clearly expected because, when we have more resources or tasks, we need more time to execute the optimization algorithm. Instead, if we introduce an approximation thanks to the γ parameter, the cost improvement is reduced more and more, as well as the execution time. If the number of tasks and resources involved increase, first of all the gaps between the lines in the graphs are larger than the ones representing the exact method and, secondly, the costs tend to reduce more when the γ variable is farther from 0 because of the decreasing of the execution time. However, even if there is a big and clear modification in the several shapes, with different values of the above-mentioned metrics the lines in the graphs follow an exponential trend in any case.

By considering the point of view of the double auctions scenario, in particular the *McAfee* algorithm, when the tasks are more than the resources, the lines follow a linear trend and grow according to the execution time and the increasing of the number of tasks themselves: in fact, if we focus on the case with 20 resources and 20 tasks, we can see a linear evolution in terms of execution time and tasks. Nevertheless, when we

# Resources	# Tasks	Social Welfare with McAfee algorithm	Social Welfare with VCG mechanism
5	5	$1.6244019999999997 \cdot 10^{-06}$	$1.6292750000000004 \cdot 10^{-06}$
5	10	$4.7681130000000016 \cdot 10^{-06}$	$5.3102 \cdot 10^{-06}$
5	20	$1.3477735999999983 \cdot 10^{-05}$	$1.3805700000000002 \cdot 10^{-05}$
10	5	$3.2472720000000016 \cdot 10^{-06}$	$3.4242 \cdot 10^{-06}$
10	10	$4.9311070000000035 \cdot 10^{-06}$	$5.804974999999999 \cdot 10^{-06}$
10	20	$1.4043570000000019 \cdot 10^{-05}$	$1.4238974999999997 \cdot 10^{-05}$
20	5	$8.492335999999993 \cdot 10^{-06}$	$8.527600000000003 \cdot 10^{-06}$
20	10	$1.0761819000000008 \cdot 10^{-05}$	$1.1030125 \cdot 10^{-05}$
20	20	$1.6204474 \cdot 10^{-05}$	$1.6373324999999996 \cdot 10^{-05}$

Table 4.2: *The comparison of social welfare of the two algorithms based on double auctions scenario: the larger, the better.*

have the number of tasks that is smaller than the amount of the resources, given that the algorithm fills this gap duplicating the resources, we can notice a stepping-curve before maintaining the classical linear shape. On the other hand, if we take in consideration the graph of *VCG* procedure, we can see a slightly noisy evolution very similar to the one of the *McAfee* algorithm. With the comparison of the two above-mentioned mechanisms, it is clear that, like the theory suggested, *VCG* gains a larger value for the social welfare[2] than the *McAfee* because of its greater economic efficiency.

To summarize, the simulation showed how the execution time is different in the two considered games, like we expected because of the dissimilarity of the algorithm complexity: while the double auctions game presents a more linear growth in the time spent to execute the policy, the congestion situation has an exponential evolution and, so, requires more time to carry out the resource allocation.

[2]The larger the social welfare, the better the result.

CHAPTER 5

Implementation of the algorithms

In this chapter, we talk about the implementation of the two policies described in the thesis, starting from the description of the implementation of a generic policy on *BarbequeRTRM* software. Then, the implementation of the policies specific to this thesis is showed including the details of the new classes methods inserted.

5.1 Designing the BarbequeRTRM policies

By default, when a new policy is created via the *BarbequeRTRM* scripts, the resulting code structure is an empty skeleton that can be seen in the *UML* scheme of Figure 5.1.

From `SchedulerPolicyIF` that is the parent class, `GenericSchedPol`, i.e. its child, is characterized by three particular and private variables, a `ConfigurationManager` object (used to retrieve the configuration values), a `ResourceAccounter` object (that provides the access to the resource information) and, finally, a logger that creates the system logger instance. From the point of view of

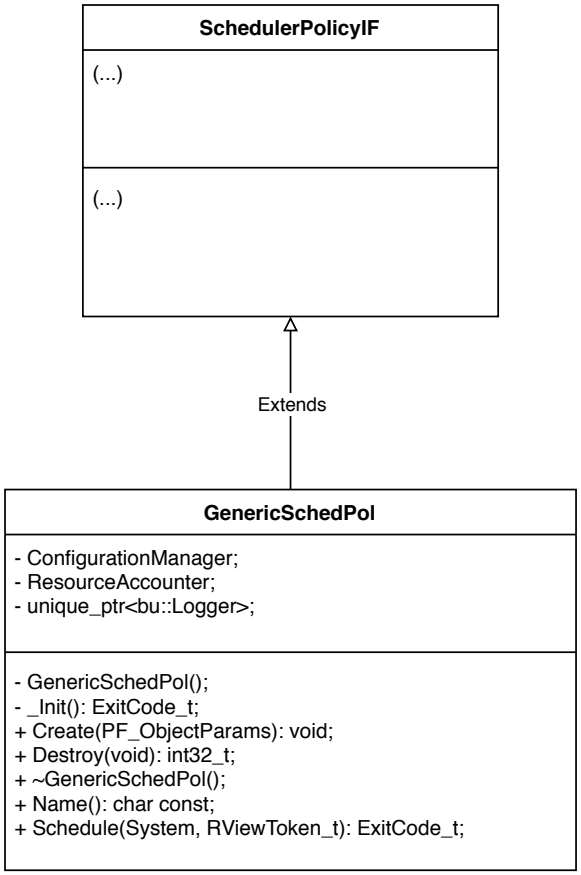

Figure 5.1: *The UML class diagram for a generic BarbequeRTRM policy.*

the functions, we have a `Create` and a `Destroy` methods that, respectively, generates and eliminates the plugin of the policy. Instead, the `~GenericSchedPol` method is the so called destructor that is the counterpart of the constructor and is necessary to clean the memory when the object is removed. The other methods are the `Name` function which returns the name of the policy plugin and the `Schedule` function which performs a new scheduling or a new resource allocation. In the end, the `_Init` method is an optional initialization member function. Before the examination of the two policies created, it is important to remark that the policy is triggered by the resource manager when some specific events occur, e.g. a new task starts to run or a task terminates its execution. Consequently, our policies run every time a change in the game occurs, by finding a new optimal solution when a new game situation appears.

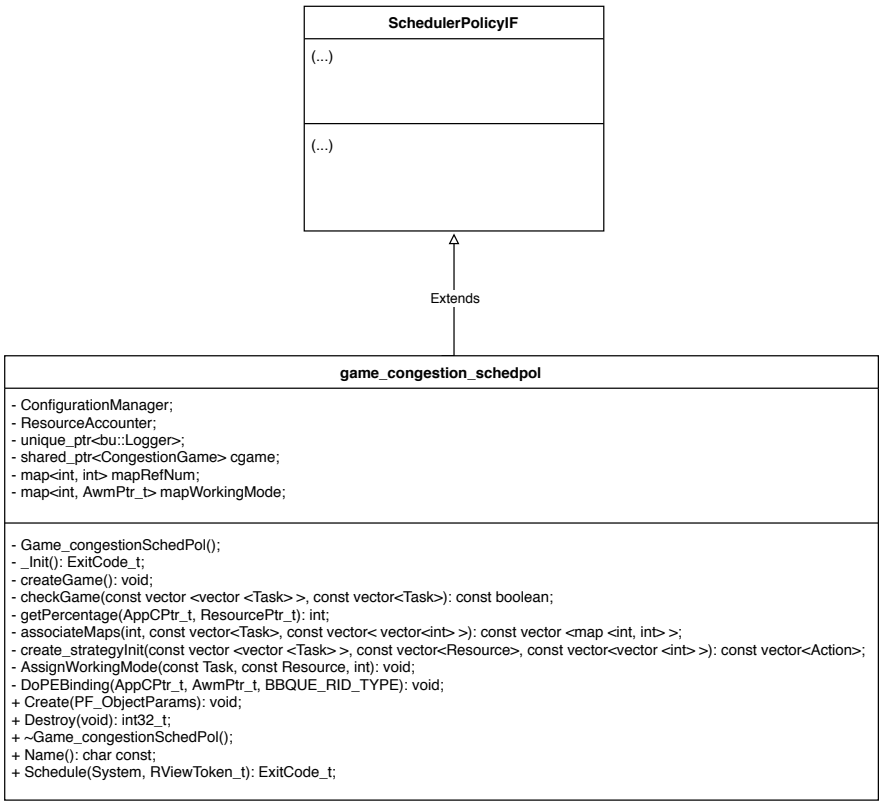

Figure 5.2: *The skeleton of the UML class diagram for the policy based on congestion games.*

5.1.1 The congestion game extensions

For the congestion game policy (its *UML* is shown in Figure 5.2), in the `Schedule` method we introduced the function `createGame` thanks to which we can initialize the game by taking the needed data from *BarbequeRTRM* itself. In fact, thanks to a pointer to the applications, we can bring the names and the *id* of the tasks, firstly the ones in the *Ready* status and, secondly, the ones that are already in execution, i.e. in the *Running* status. After a cycle used to create the initial strategy, we build the vector of resources by taking all the information from *BarbequeRTRM* and, in particular, from the `ResourceAccounter` class.

We inserted the necessary parameters of the cost function in a *XML* file that a parser, invoked by the policy, reads. Given that, we obtained the exact usage percentages of the tasks and, so, we saved these relationships by creating maps with the identifier of the tasks and the above-mentioned

usage percentages. By setting these maps to the corresponding resources, we defined the initial strategy and, then, created the game. The ε coefficient is selected to be greater or equal than 0, i.e. the default value, by modifying the *BBQUE_SCHEDPOL_GAME_CONGESTION_EPS* constant inserted in the *Kconfig* file.

To fill the cost vectors, we took the values of α_i, β_i and γ_i from the *XML* file, calculated with the formulas explained in Section 3.1.4. After the determination of the costs using the function defined in Section 3.1.4, we launched the `findNE` method to locate the *Nash Equilibrium* and we normalized the percentages thanks to some **for** cycles to scale them. To assign the tasks to the resources that represent the new relationships after the calculus of the *Nash Equilibrium*, we called the `AssignWorkingMode` method by passing the task, the relative resource, and the respective usage percentage. Then, we informed *BarbequeRTRM* about the new allocations, by asking it to validate the schedule request and apply the new resource allocations.

The algorithms implementation

Thanks to the *UML* scheme (Figure 5.3), we can describe the implementation of the algorithm in an easy way, by observing, once a time, each *C++* class that composes the algorithm and their related functions.

The class `Task` is depicted in Figure 5.4. The one and only variable of a `Task` is an integer which has the objective of identifying: with the `getID` method it is possible to obtain that unique number.

The class `Action`, showed in Figure 5.5, has two private variables and, in addition to the constructor and the re-definition of the `equal` operator, three distinct public functions: a getter for the task, a getter for the resource, and a setter for the resource.

Figure 5.6 shows the class `Resource` where we can obtain its `char` identifier, the list of tasks that use it and the `std::map` element in which there are specified the percentages of usage. We can also get its congestion thanks to the `getCongestion` method.

Finally, in Figure 5.7 there is the representation of the class `CongestionGame` which inherits from the abstract one `Game`. It is the core of the entire algorithm, so where the *Nash Equilibrium* is computed and, if it exists, found. In addition to the constructor, the function

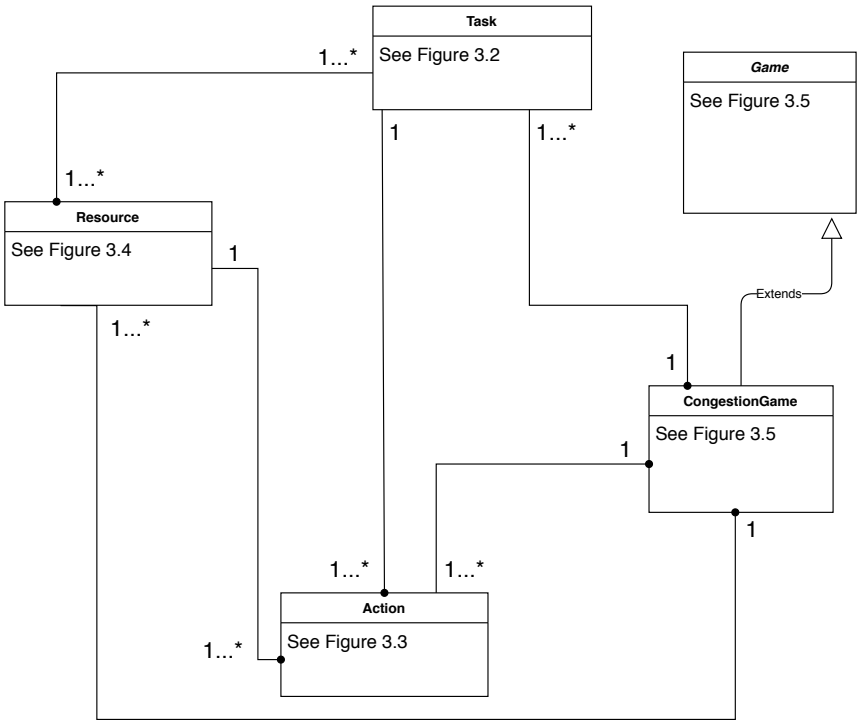

Figure 5.3: *The skeleton of the UML class diagram of the congestion game algorithm implementation.*

`setBuffCosts` sets the buffer of the costs related to the usage of a specific resource, and the `addCostsToBuffer` method adds the cost vectors to the above-mentioned buffer. Finally, we have the public method `getCostGame` that, by calculating the cost of the game, permits the computation of the *Nash Equilibrium* in `findNE`. The other functions are the private methods that the algorithm uses to check if there is, at least, one equilibrium.

5.1.2 The double auctions extensions

For the double auctions policy (its *UML* is shown in Figure 5.8), in the `Schedule` method we introduced the function `createGame` thanks to which we can initialize the game by taking the needed data from *BarbequeRTRM* itself. In fact, with a pointer to the applications, we can bring the names and the *id*s of the tasks, firstly the ones in the *Ready* status and, secondly, the ones that are already in execution, i.e. in the *Running* status. We inserted the necessary parameters of the cost function in a *XML* file

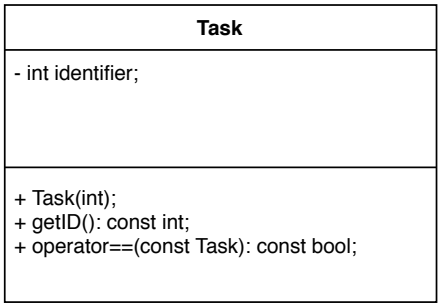

Figure 5.4: *The UML class diagram for* `Task` *(congestion game).*

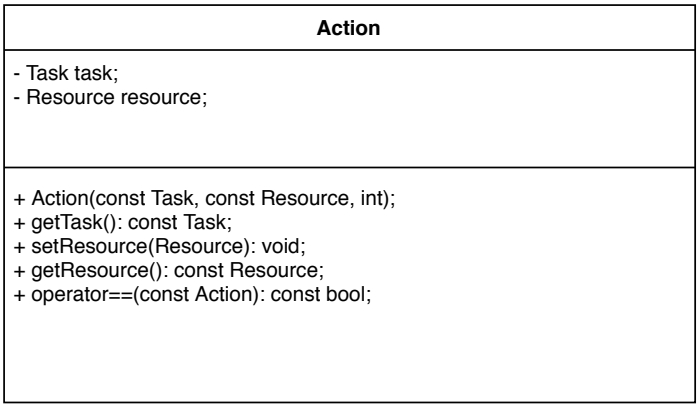

Figure 5.5: *UML class diagram for* `Action` *(congestion game).*

that a parser, invoked by the policy, reads. Given that, we had the resource bids used with their identifier to create the vector of the resources: in this way, we defined the game.

According to the *Kconfig* file, given that the choice is between two constants:

- *CONFIG_BBQUE_SCHEDPOL_GAME_AUCTIONS_VCG*
- *CONFIG_BBQUE_SCHEDPOL_GAME_AUCTIONS_MCAFEE*

respectively we called either the `find_BA_VCG` method or the `find_BA_McAfee` function. To assign the tasks to the resources that represent the new relationships after the end of the double auctions game, we called the `AssignWorkingMode` method by passing the task and the relative resource. Then, we informed *BarbequeRTRM*, via the `ScheduleRequest` methods, to apply the modifications to the task allocations.

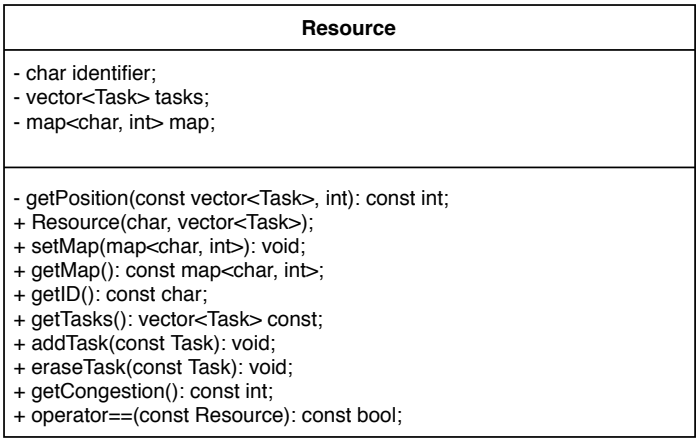

Figure 5.6: *UML class diagram for* `Resource` *(congestion game).*

The algorithms implementation

Thanks to the *UML* scheme (Figure 5.9), we can easily describe these algorithms by observing, once a time, the classes and, specifically, their interactions and functions.

As private variables of the class `Task` (Figure 5.10), there are the identifier, that is an integer with the purpose of identifying, and the bid, i.e. a float value of how much the task is inclined to offer for having the "item" from the resource.

The class `Resource` (Figure 5.11) has an identifier in the form of integer that determines the object, a second identifier, another integer, used to identify it only in the case if the resource is duplicated, a bid i.e. a float value that specifies the offer of the resource with which sells the "item" to the buyer, and, finally, the task to which the resource is going to sell the "item".

In the end, in Figure 5.12, we have the `DoubleAuctions` class that has a vector of tasks and a vector of resources as private variables. In addition to the obvious `getResources` and the methods strictly related to the *McAfee* algorithm and the *VCG* mechanism, we have to mention the `copyandpaste` function thanks to which the resources are duplicated if their amount is smaller than the tasks number.

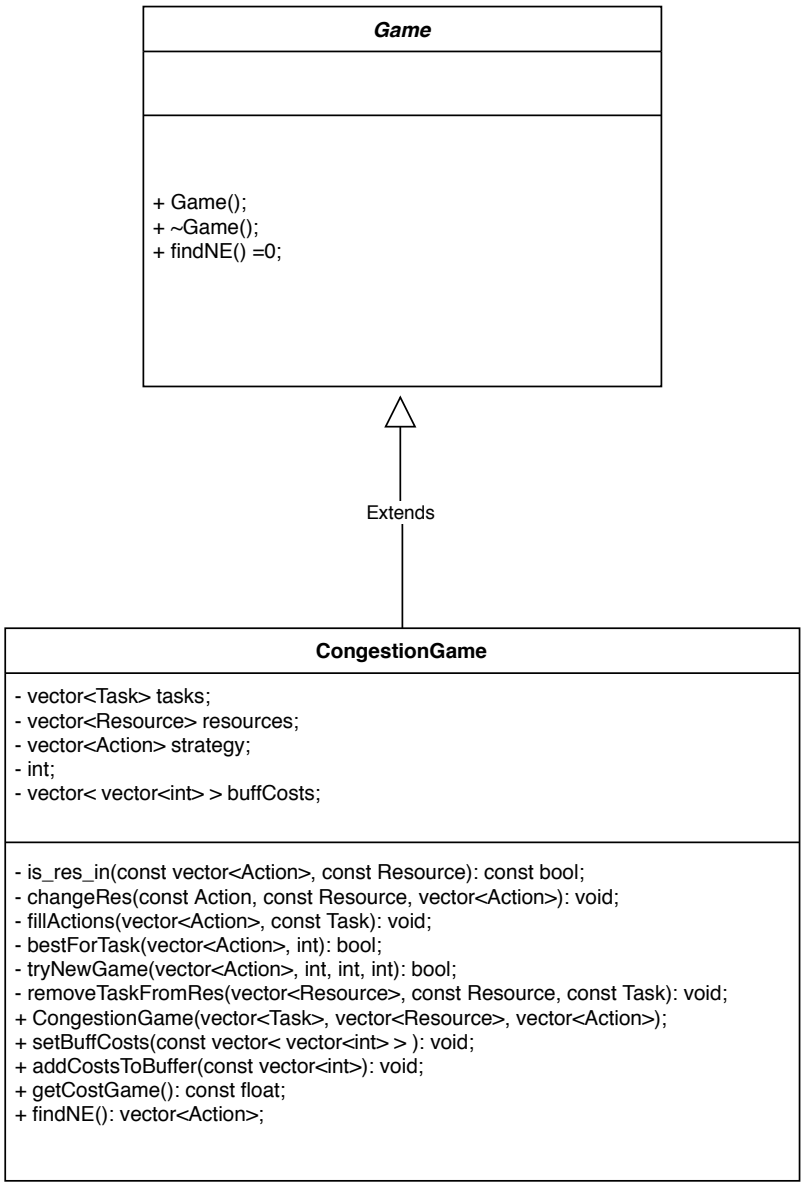

Figure 5.7: *UML class diagram for* `Game` *and* `CongestionGame` *(congestion game).*

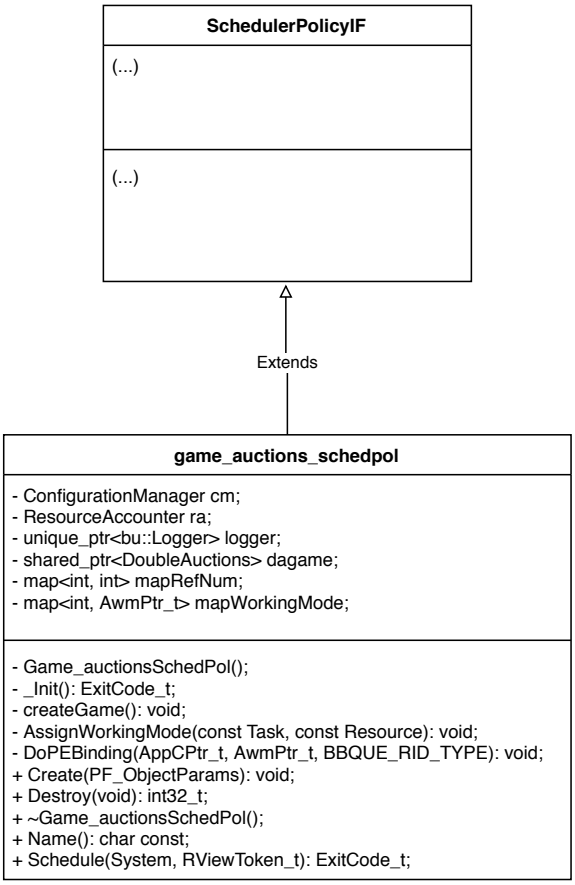

Figure 5.8: *The skeleton of the UML class diagram for the policy based on double auctions games.*

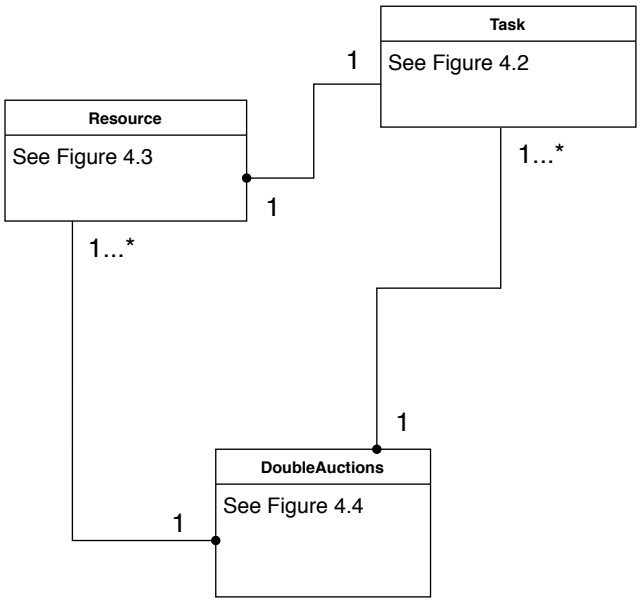

Figure 5.9: *The skeleton of the UML class diagram of the double auctions algorithm implementation.*

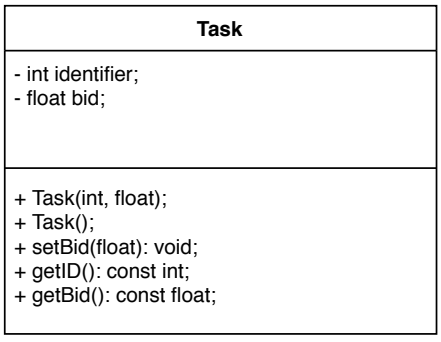

Figure 5.10: *The UML class diagram for* `Task` *(double auctions).*

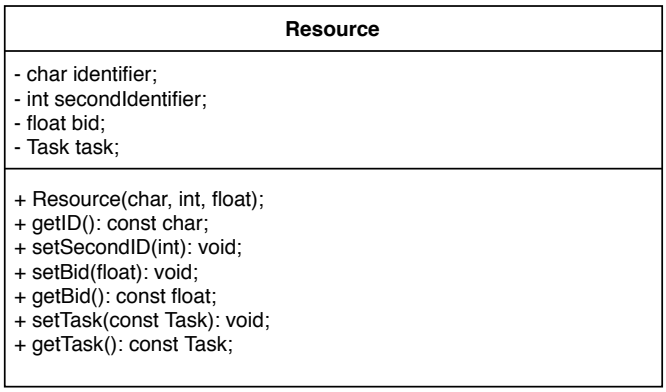

Figure 5.11: *The UML class diagram for* `Resource` *(double auctions).*

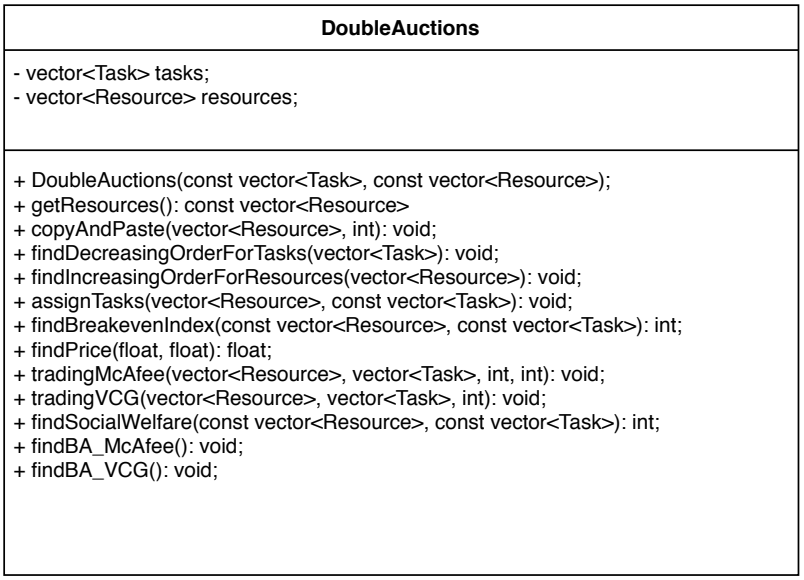

Figure 5.12: *The UML class diagram for* `DoubleAuctions` *(double auctions).*

```
<?xml version="1.0" encoding="utf-8"?>
<entities>
    <resources>
        <resource id="...">
            <alpha>...</alpha>
            <alpha0>...</alpha0>
            <beta>...</beta>
            <beta0>...</beta0>
            <gamma>...</gamma>
        </resource>
        (...)
    </resources>
    <tasks>
        <task name="...">
            <percentage id="...">...</percentage>
            (...)
        </task>
        (...)
    </tasks>
</entities>
```

Figure 5.13: *The skeleton of the XML file for congestion game policy.*

5.2 The configuration parameters files

As explained in the previous sections, we created two *XML* files that contain the parameters necessary to the calculus of the cost function. A custom parser based on the *RapidXML* library has been developed for both the congestion games and the double auctions policies. In the following subsections the content of these two files is briefly described and shown in Figure 5.13 and Figure 5.14.

The congestion game case

In the tag *entities*, we can find the sub-tags *resources* and *tasks*. In our experimental evaluation they respectively contain four *resource*s, representing `cpu0`, `cpu1`, `cpu2` and `cpu3`, and two *task*s, i.e. `facesim` and `blackscholes`. For each tag *resource*, we have an *id* attribute and five different sub-tags, to define α, $\alpha 0$, β, $\beta 0$ and γ. On the other hand, in each tag *task*, in addition to the *name* attribute, we write the *percentage* sub-tag which has the *id* attribute that determines the resource and its

```
<?xml version="1.0" encoding="utf-8"?>
<entities>
    <resources>
        <resource id="..." bid="..."/>
        (...)
    </resources>
    <tasks>
        <task name="..." bid="..."/>
        (...)
    </tasks>
</entities>
```

Figure 5.14: *The skeleton of the XML file for double auctions policy.*

content that represents the usage percentage.

The double auctions case

As in the congestion game case, in the tag *entities*, we can find the sub-tags *resources* and *tasks*. In our experimental evaluation they respectively, contain four *resource*s, by representing `cpu0`, `cpu1`, `cpu2` and `cpu3`, and two *task*s, by identifying `facesim` and `blackscholes`. For each tag *resource*, we have two attributes, the *id* and the *bid*. On the other hand, in each tag *task*, we have the *name* and the *bid* attributes that determine the task and the value that is offered to buy the "item".

CHAPTER 6

Experimental evaluation

In this section, we aim to test the two proposed policies, the first related to the congestion game scenario and the second associated to the double auctions scenario, by evaluating their effectiveness compared to not only each other but also the default *Linux* and *BarbequeRTRM* policies.

6.1 Experimental setup

To perform the above-mentioned experiments, we used an high-end workstation that has an *Intel i7-3770* quad-core processor and a *x86_64* architecture. With a *RAM* of 8 GB and a cache of 8 MB, its maximum frequency is 3.4 GHz. Each core has the capability to run two hardware-threads (for a total of 8 in the machine). The system runs a *Linux* operative system with a kernel of version $4.15.0$.

As workload we used some *benchmarks* to understand, in an experimental way, the advantages and the disadvantages in using the new policy. The benchmark suite has been developed to take into account the subsequent goals:

- the applications should stress the various classes of architectural characteristics of the machine;
- it should depict the important applications on the target processors;
- its workloads should differentiate enough to exhibit the field of behaviours of the target applications;
- it should use state-of-the-art algorithms and techniques.

We selected *PARSEC* suite [5] (already integrated in *BarbequeRTRM*) and, in particular, the `blackscholes` and `facesim` benchmarks. They have been enabled in the *BarbequeRTRM Kconfig* system. From the *PARSEC*'s original paper [5], we knew that one of the goals of this particular suite of benchmarks, written in *C* and *C++*, is to select a set of programs that is large and diverse enough to be sufficiently representative for scientific studies. It consists, in total, of 9 applications and 3 kernels, chosen from wide options of application domains with different combinations of parallel models, machine requirements and runtime behaviours.

Although we had not a machine with heterogeneous architecture, that is with different core architectures, we instead used a server machine with one package (i.e. a single physical processor) configured as an homogeneous quad-cores. To simulate the heterogeneity, we assigned to each core a different constant frequency, by using a module of the *Linux* kernel that is called *cpufreq*. We set the governor (i.e. the *cpufreq* behavioural policy) to *userspace* because, in this way, the user setting can not autonomously be modified by *Linux* itself. We manually adjusted the core frequencies to four different values, $3.4GHz$ for the `cpu0`, $2.9GHz$ for the `cpu1`, $2.2GHz$ for the `cpu2` and $1.6GHz$ for the `cpu3`.

To avoid the interference in the same core which would reduce performance and experiment reproducibility, we disabled the hyperthreading. This is usual in the *HPC* scenario: a logic core is deactivated in every physical one, having, in this way, only one hardware thread for each core. To measure the power and the energy, we used *Intel RAPL*, a driver of *Linux* kernel called *Running Average Power Limit*, which allows us to read the power and energy consumption data.

6.2 Estimation of power and energy coefficients

The *Intel RAPL* allowed us to measure the total energy consumed by the whole processor (i.e. the socket). By considering that we modified the frequencies of the four internal cores, to calculate the single energy consumed by each core, it is necessary to switch off one core at a time, measuring the energy consumed by the remaining three together, and, then, calculate the difference. This procedure is repeated for the other three cores.

IntelRAPL offers the possibility to read also the instantaneous power consumption. However, the machine used for the tests do not support this feature. Consequently, to find the power, we calculated the total energy consumed by the processor at two different timing instants separated by a small interval. After the calculation of the delta of these two values, the difference is divided by the size of the time interval, by determining, in this way, the derivative of the energy, i.e. the instantaneous power.

$$\Delta_E = E(t_1) - E(t_0)$$

$$P = \frac{\Delta_E}{\Delta_T}$$

Once P has been computed (the results are shown in Table 6.1), the γ_i coefficient is found according to Equation 3.7.

Starting from `cpu0` to `cpu3`, one at a time, we spawned only one task pinned on the core under analysis, while the others remain in the *idle* state (so, their power consumption is minimum). For each processor, we measured the power 100 times: this was necessary because the *Linux* kernel and system services introduce noises in the measurements, thus we computed the average value. The total power measured from *Intel RAPL* is composed of:

$$P_{tot} = k + P_0 + P_1 + P_2 + P_3$$

where k is the static energy and P_i is the energy of the i-th core.

When all the cores are turned on and no task is in execution (so, we are calculating the constant k, that is the static power, i.e. the parasitic power consumption present even if all the cores are in *idle*). The measured value is $k = 0.487W$. With `cpu1`, `cpu2` and `cpu3` in the *idle*

# Core	Dynamic Power
0	9.255 W
1	6.776 W
2	5.152 W
3	3.811 W

Table 6.1: *The estimated dynamic power for the cores of the processor under analysis.*

# Core	PDP
0	$\frac{9.255W}{3.4 \cdot 10^9 GHz} = 2.722 \cdot 10^{-}9\ W \cdot s$
1	$\frac{6.776W}{2.9 \cdot 10^9 GHz} = 2.337 \cdot 10^{-}9\ W \cdot s$
2	$\frac{5.152W}{2.2 \cdot 10^9 GHz} = 2.342 \cdot 10^{-}9\ W \cdot s$
3	$\frac{3.811W}{1.6 \cdot 10^9 GHz} = 2.382 \cdot 10^{-}9\ W \cdot s$

Table 6.2: *The estimated Power Delay Product (PDP) for the cores of the processor under analysis.*

state, we obtained $k + P_0 = 9.742W$. Then, by maintaining processors `0`, `2` and `3` in the *idle* state, we read $k + P_1 = 7.263W$. After that, by keeping `cpu0`, `cpu1` and `cpu3` in the *idle* state, we found $k + P_2 = 5.640W$. Finally, with the processors `0`, `1` and `2` in the *idle* state, we measured $k + P_3 = 4.298W$. By subtracting the value of k from the four sums, we calculated the values presented by Table 6.1.

Table 6.2 shows the four values of PDP as defined in Equation 3.6.

It is possible to notice that the PDP value is similar for all the cores, especially the intermediate values, PDP_1 and PDP_2. Among the four considered frequencies, on the basis of PDP, the most efficient processor[1] is `cpu1` which frequency is $2.9GHz$. We can also notice that it does not exist a linear relationship between the values as also noticed by previous works [36].

To ease the selection of the user coefficients and to get rid of the different units of measurement, we decided to normalize P_i, PDP_i, and f_i values, by mapping the larger of each coefficient to 1. By starting from

[1] If we increase the performance by maintaining the same power, we gain more efficiency.

this new value, for the other three cores, the numbers are proportionally computed.

$$\widehat{PDP}_0 = 2.722/2.722 = 1$$

$$\widehat{PDP}_1 = 2.337/2.722 = 0.859$$

$$\widehat{PDP}_2 = 2.342/2.722 = 0.860$$

$$\widehat{PDP}_3 = 2.382/2.722 = 0.875$$

We followed the same procedure of normalization for the frequencies: in this case, we set $\frac{1}{f_3}$ to 1 because it resulted as the biggest value.

$$\hat{f}_0 = 0.294/0.625 = 0.470$$

$$\hat{f}_1 = 0.345/0.625 = 0.552$$

$$\hat{f}_2 = 0.455/0.625 = 0.728$$

$$\hat{f}_3 = 0.625/0.625 = 1$$

Eventually, the new P_i values (identified by $\hat{P}_i$) are:

$$\hat{P}_0 = 9.255/9.255 = 1$$

$$\hat{P}_1 = 6.776/9.255 = 0.732$$

$$\hat{P}_2 = 5.152/9.255 = 0.557$$

$$\hat{P}_3 = 3.811/9.255 = 0.412$$

As expected, the highest power corresponds to the highest frequency.

Regarding $\alpha_{i,0}$ and $\beta_{i,0}$, we employed the data of the scientific article [7] where the authors calculated the context switch overhead in the range $[0.17\%, 0.25\%]$. To be conservative, we chosen the worst case scenario:

$$\alpha_{i,0} = \beta_{i,0} = 0.25\%$$

All the coefficients presented in the previous paragraphs are used by both the congestion game and the double auctions policies.

# Core	Percentage	Average execution time in *ms*
0	47%	5896.294
1	58%	5909.547
2	77%	5647.083
3	100%	5929.859

Table 6.3: *The measured execution times and the computed percentages for benchmark* `facesim`.

# Core	Percentage	Average execution time in *ms*
0	41%	14.050
1	50%	13.961
2	70%	13.438
3	100%	13.702

Table 6.4: *The measured execution times and the computed percentages for benchmark* `blackscholes`.

6.3 Congestion games

In this section, we explain how the task percentages needed for the congestion games are computed, by presenting also the result of the experimental evaluation for these games.

6.3.1 Selecting the CPU share based on QoS

The congestion games policy requires to pre-compute the `cpu` percentage (share) needed to reach the performance goal (or *Quality of Service - QoS*) of each task. Since we are using benchmarks as workload, we took into account the core characterized by the lowest frequency and we assigned to it the 100% percentage. Then, we tested the other *CPU*s with different percentages (by varying the share of 1% at a time): when the similar[2] performance of the core with 100% percentage of usage is reached, the new percentage value is saved. The performance metrics considered is the average execution time over 100 runs. The results are showed in Table 6.3 and Table 6.4 for, respectively, the `facesim` and `blackscholes` benchmarks.

Due to an internal limitation of the *BarbequeRTRM* software, we have

[2]The percentage that leads to the minimum absolute difference has been chosen.

to limit the sum of the usage percentages linked to a resource that must be, at most, 100% because, in this way, the tasks which use that resource are scheduled simultaneously. This allows the *Linux* scheduler to decide the task in real execution during a precise instant of time. This is critical if the system is *oversubscribed*, i.e. there are more tasks than `cpus`. From the percentages previously computed for the congestion game, we can pass to the just described representation of *BarbequeRTRM*:

$$\hat{X}_{i,j} = \frac{X_{i,j} \cdot 100}{\sum_k X_{k,j}}$$

where $X_{i,j}$ is the i-th task usage percentage of the j-th resource and $\sum_k X_{k,j}$ is the usage percentages sum of the j-th resource from all the k tasks that use it.

6.3.2 Results

To study the evolution of the wall execution time[3], energy and power in the congestion games scenario, we chose to vary $\hat{\alpha}$, $\hat{\gamma}$ and $\hat{\beta}$, according to a vector composed by following values: $[0, 0.1, 0.25, 0.3, 0.5, 0.75, 1, 1.5, 3, 5, 7]$.

In this way, regarding $\hat{\alpha}$ which, in theory, affects the execution time, with lower values of it, we discovered that there is a considerable increment of the time spent by the tasks to run. This is because, obviously, $\hat{\alpha}$ is not too much influential in the cost function. On the other hand, if it is larger, it is possible to obtain a better execution time which is much shorter than in the previous situation. In Figure 6.1, by focusing on the blue line, the above-mentioned behaviour is clearly visible.

With reference to $\hat{\gamma}$, when low values are used, it is visible that the power, i.e. the metrics influenced by this variable, increases considerably, while, in the opposite case, that is the one in which $\hat{\gamma}$ is large, as expected by the theory, we obtain a lower value for the power consumption as shown by Figure 6.2 (red line).

Then, by considering the $\hat{\beta}$ variable, we noticed that this cost function coefficient is not effective as expected as regard the energy evolution. By observing the Figure 6.3 and, in particular, the green line, it is visible that,

[3]The wall execution time is the time interval between the moment in which all the benchmarks are activated and the moment in which all of them terminate their execution.

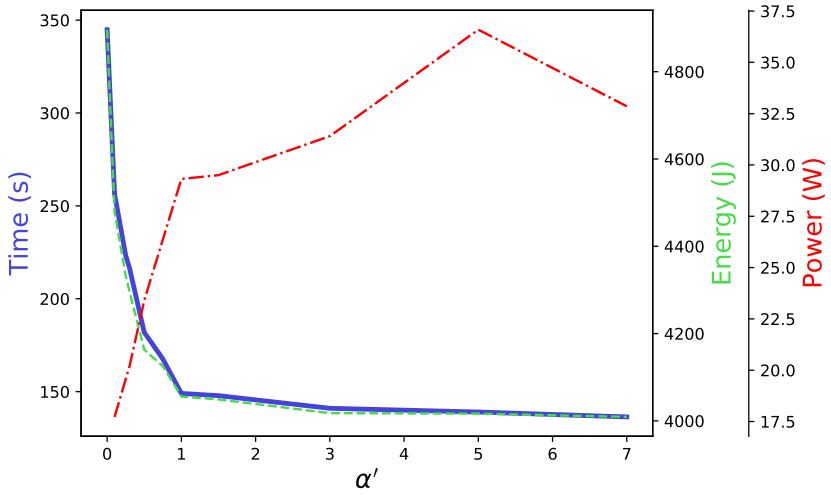

Figure 6.1: *The measured values of time, energy and power, according to the variation of $\hat{\alpha}$ with $\hat{\gamma}$ and $\hat{\beta}$ equal to 1.*

although we used bigger values for $\hat{\beta}$, it did not cause an improvement of the amount of energy. This is because, knowing that it depends a lot on the execution time of the tasks, we must remember that the congestion game scenario does not have any information about it. This analysis is really problematic in our experiments because the chosen benchmarks have an execution time very different among each other: indeed, `facesim` takes 500x more time than `blackscholes`.

Finally, in Figure 6.4, while the amount of tasks varies on the x-axis, we can see the differences between the situation in which *BarbequeRTRM* is used[4], and the one where the resource manager is not employed. In the first above-mentioned case, the execution time is, in general, higher than in the second one, but, by omitting the situation concerning the energy, we can see that the system consumed less power during the experiments involving *BarbequeRTRM*.

6.4 Double auctions games

To study the evolution of the wall execution time[5], energy and power in the double auctions situation, we chose to vary $\hat{\alpha}$, $\hat{\gamma}$ and $\hat{\beta}$, according to a

[4] In this case, we considered: $\hat{\alpha} = 1$, $\hat{\beta} = 1$, and $\hat{\gamma} = 1$.

[5] See Footnote 3 for the definition of wall execution time.

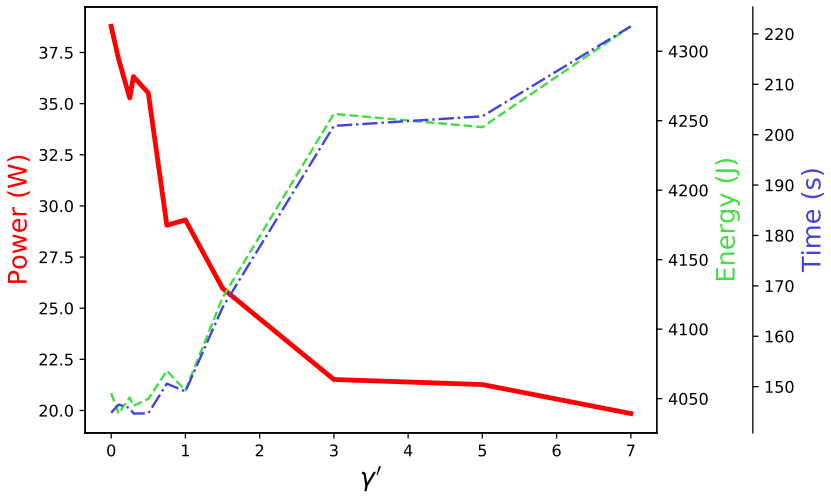

Figure 6.2: *The measured values of time, energy and power, according to the variation of $\hat{\gamma}$ with $\hat{\alpha}$ and $\hat{\beta}$ equal to 1.*

vector composed of the values $[0, 0.25, 0.5, 0.75, 1, 1.25, 1.5]$.

6.4.1 Results

We proceed similarly to the previous section of the congestion games. In this way, we discovered that, regarding $\hat{\alpha}$ which, in theory, affects the evolution time, with lower values of this cost function coefficient, we have a considerable increment of the time spent by the resource manager to run the tasks because, obviously, $\hat{\alpha}$ is not too much influential in the cost function. On the other hand, the larger the $\hat{\alpha}$ value, the smaller the execution time which is considerably shorter than in the previous situation. In Figure 6.5 and Figure 6.6, by observing, in particular, the blue lines, we can see the above-mentioned behaviour with one `facesim` task and one `blackscholes` task, when scheduled by, respectively, the *McAfee* algorithm and the *VCG* mechanism.

Then, in Figure 6.7 and Figure 6.8, by observing, in particular, the blue lines, we can see the above-mentioned behaviour with two `facesim` tasks and two `blackscholes` tasks, when scheduled by, respectively, the *McAfee* algorithm and the *VCG* mechanism.

Finally, in Figure 6.9 and Figure 6.10, by observing, in particular, the blue lines, we can see the above-mentioned behaviour with four `facesim` tasks and four `blackscholes` tasks, when scheduled by, respectively,

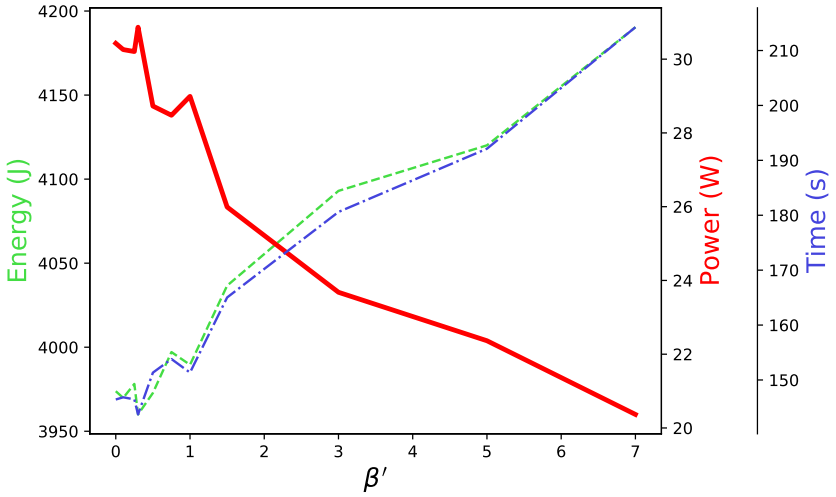

Figure 6.3: *The measured values of time, energy and power, according to the variation of $\hat{\beta}$ with $\hat{\alpha}$ and $\hat{\gamma}$ equal to 1.*

the *McAfee* algorithm and the *VCG* mechanism: in these cases, the evolution of the time has larger variations compared to the previous examples. This is the first symptom that the double auctions game does not perform very well when the number of tasks is larger than the amount of resources.

By considering, instead, the $\hat{\beta}$ variable, with the same setup of the $\hat{\alpha}$ variable, we noticed that this cost function coefficient is not effective as expected regarding the energy evolution: by observing the green lines in Figures 6.11, 6.12, 6.13, 6.14, 6.15 and 6.16, it is not possible to claim a given trend or clear relation. We suspected that this is caused by the lack of heterogeneity of the processors because we modified only the frequency to differentiate the four cores in our experimental setup. In fact, if we consider the $\hat{\beta}$ variable singularly, i.e. with $\hat{\alpha} = 0$ and $\hat{\gamma} = 0$, we obtain a decreasing energy function when $\hat{\beta}$ increases.

Regarding the $\hat{\gamma}$ coefficient by using lower values, it is visible that the power, i.e. the metrics influenced by this variable, increases larger, while, in the opposite case, that is the one in which $\hat{\gamma}$ is equal to bigger numbers, as expected, we obtain a lower power consumption as depicted by Figures 6.17, 6.18, 6.19, 6.20, and 6.21 (red lines).

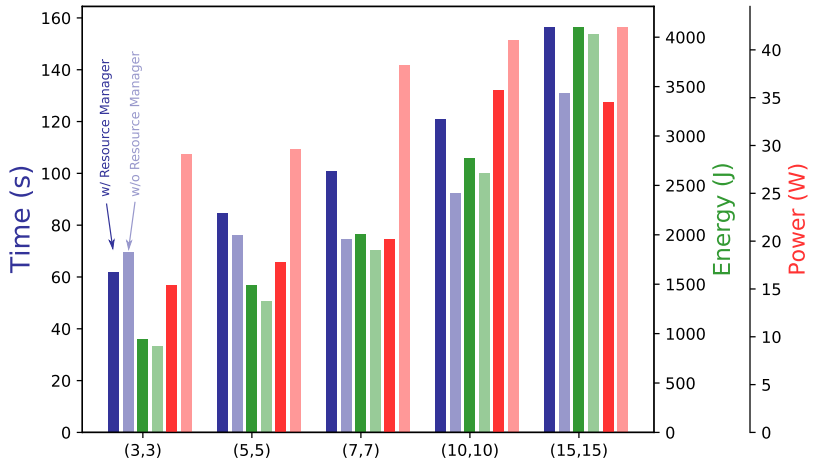

Figure 6.4: *The comparison of the "with resource manager" situation and the "without resource manager" situation according to time, energy and power. The values of $\hat{\alpha}$, $\hat{\beta}$ and $\hat{\gamma}$ are 1. The x-axis represents the number of tasks: (# `facesim`, # `blackscholes`).*

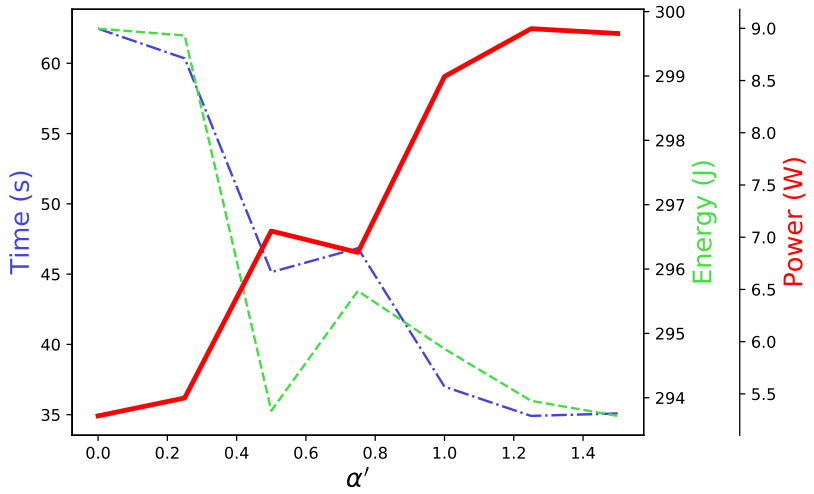

Figure 6.5: *The analysis of time, energy and power, by varying $\hat{\alpha}$ and maintaining $\hat{\beta}$ and $\hat{\gamma}$ equal to 1. In this case, we run one `facesim` task and one `blackscholes` task, using the McAfee algorithm.*

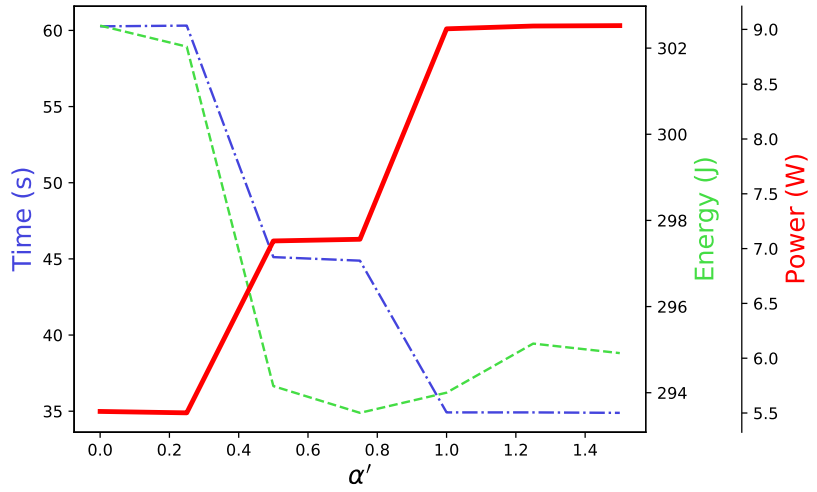

Figure 6.6: *The analysis of time, energy and power, by varying $\hat{\alpha}$ and maintaining $\hat{\beta}$ and $\hat{\gamma}$ equal to 1. In this case, we run one* `facesim` *task and one* `blackscholes` *task, using the VCG mechanism.*

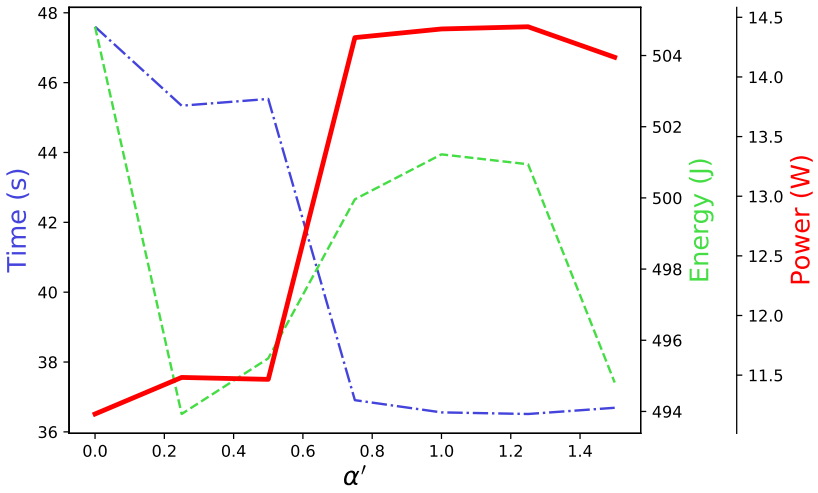

Figure 6.7: *The analysis of time, energy and power, by varying $\hat{\alpha}$ and maintaining $\hat{\beta}$ and $\hat{\gamma}$ equal to 1. In this case, we run two* `facesim` *tasks and two* `blackscholes` *tasks, using the McAfee algorithm.*

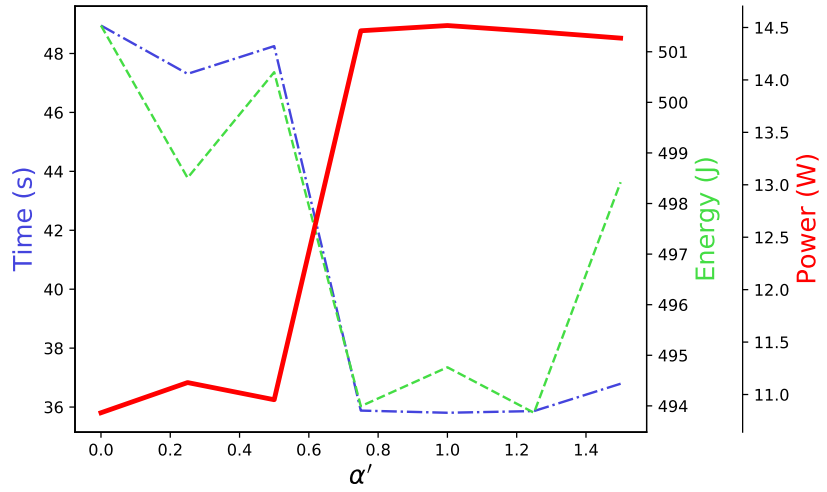

Figure 6.8: *The analysis of time, energy and power, by varying $\hat{\alpha}$ and maintaining $\hat{\beta}$ and $\hat{\gamma}$ equal to 1. In this case, we run two* `facesim` *tasks and two* `blackscholes` *tasks, using the VCG mechanism.*

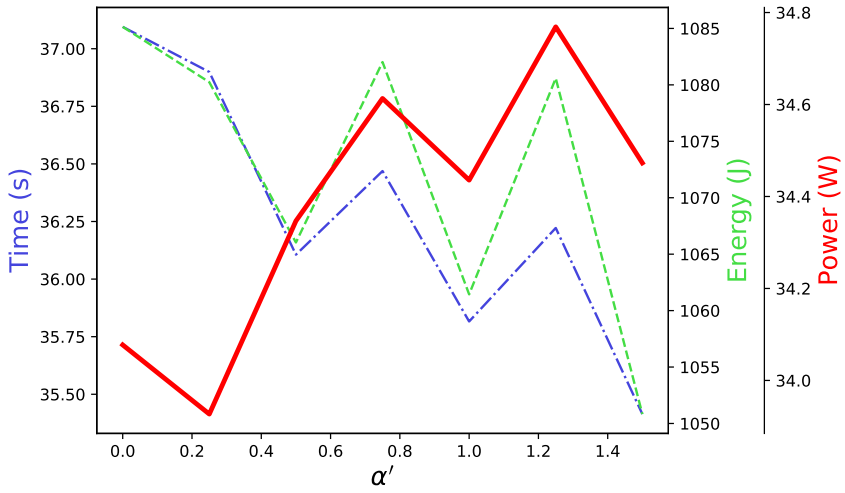

Figure 6.9: *The analysis of time, energy and power, by varying $\hat{\alpha}$ and maintaining $\hat{\beta}$ and $\hat{\gamma}$ equal to 1. In this case, we run four* `facesim` *tasks and four* `blackscholes` *tasks, using the McAfee algorithm.*

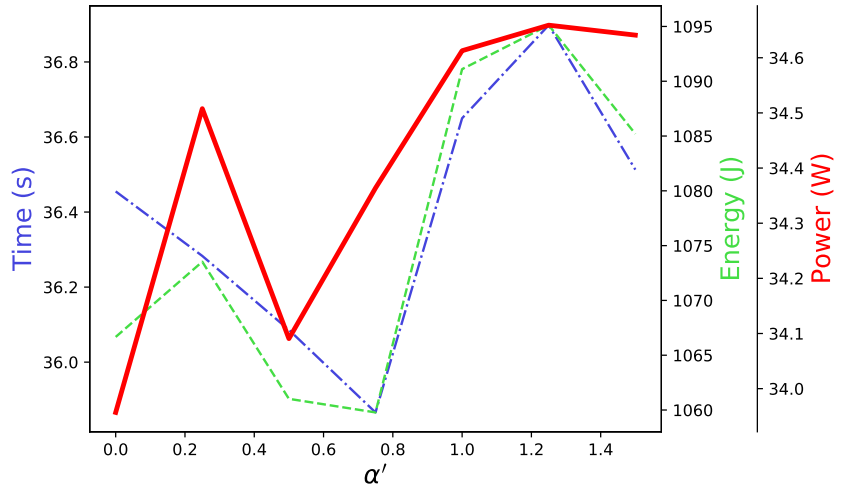

Figure 6.10: *The analysis of time, energy and power, by varying $\hat{\alpha}$ and maintaining $\hat{\beta}$ and $\hat{\gamma}$ equal to 1. In this case, we run four* `facesim` *tasks and four* `blackscholes` *tasks, using the VCG mechanism.*

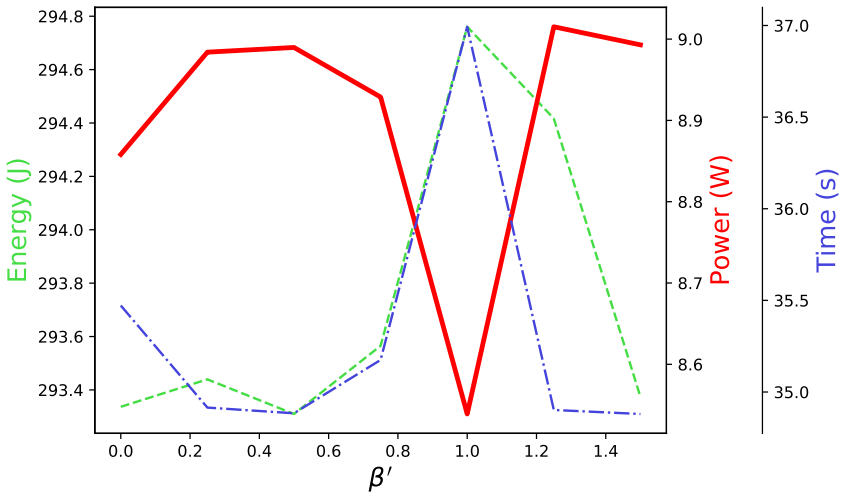

Figure 6.11: *The analysis of time, energy and power, by varying $\hat{\beta}$ and maintaining $\hat{\alpha}$ and $\hat{\gamma}$ equal to 1. In this case, we run one* `facesim` *task and one* `blackscholes` *task, using the McAfee algorithm.*

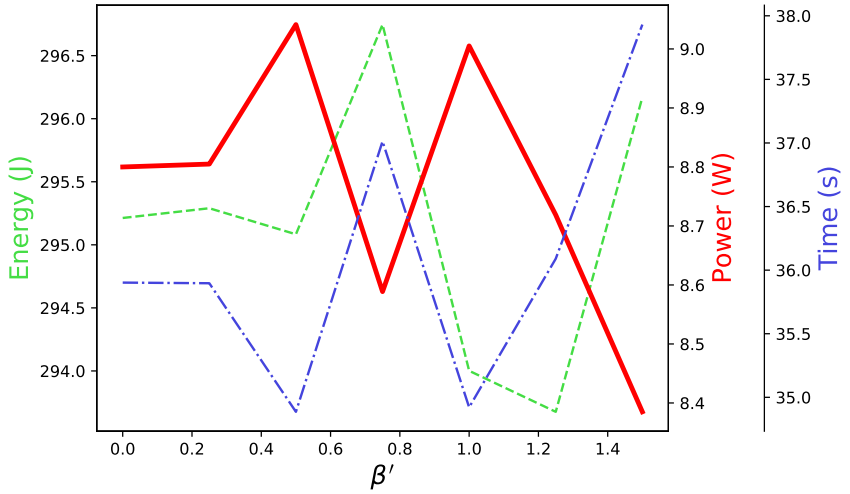

Figure 6.12: *The analysis of time, energy and power, by varying $\hat{\beta}$ and maintaining $\hat{\alpha}$ and $\hat{\gamma}$ equal to 1. In this case, we run one* `facesim` *task and one* `blackscholes` *task, using the VCG mechanism.*

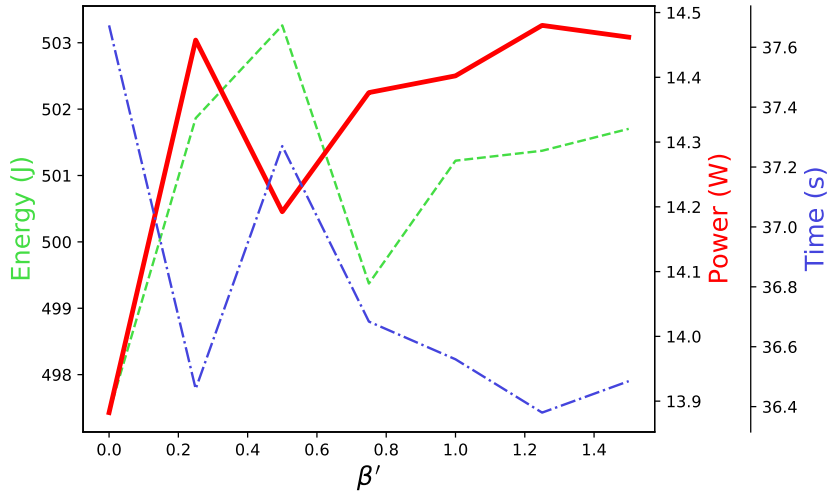

Figure 6.13: *The analysis of time, energy and power, by varying $\hat{\beta}$ and maintaining $\hat{\alpha}$ and $\hat{\gamma}$ equal to 1. In this case, we run two* `facesim` *tasks and two* `blackscholes` *tasks, using the McAfee algorithm.*

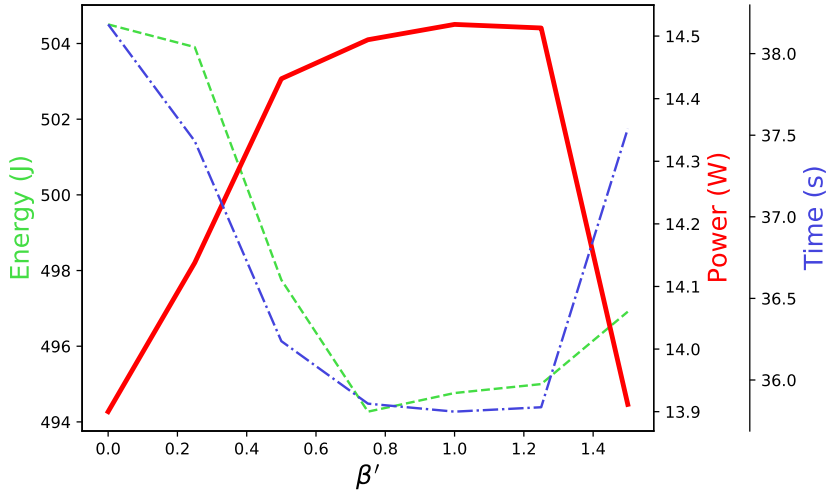

Figure 6.14: *The analysis of time, energy and power, by varying $\hat{\beta}$ and maintaining $\hat{\alpha}$ and $\hat{\gamma}$ equal to 1. In this case, we run two* `facesim` *tasks and two* `blackscholes` *tasks, using the VCG mechanism.*

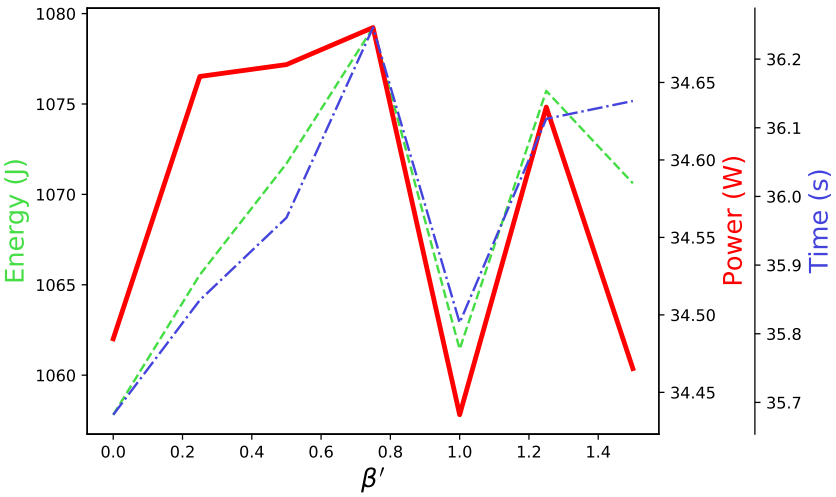

Figure 6.15: *The analysis of time, energy and power, by varying $\hat{\beta}$ and maintaining $\hat{\alpha}$ and $\hat{\gamma}$ equal to 1. In this case, we run four* `facesim` *tasks and four* `blackscholes` *tasks, using the McAfee algorithm.*

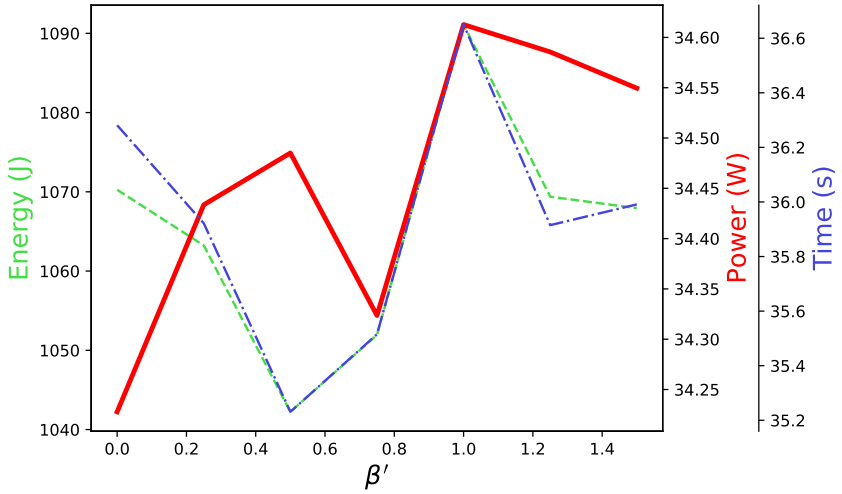

Figure 6.16: *The analysis of time, energy and power, by varying $\hat{\beta}$ and maintaining $\hat{\alpha}$ and $\hat{\gamma}$ equal to 1. In this case, we run four* `facesim` *tasks and four* `blackscholes` *tasks, using the VCG mechanism.*

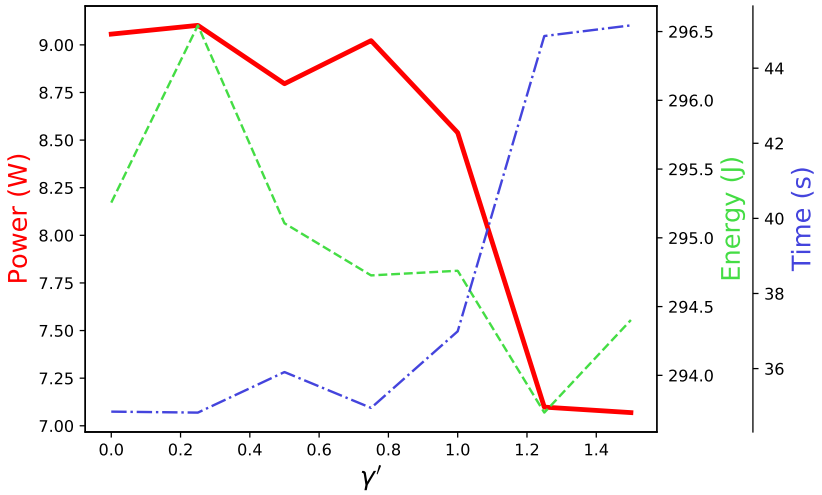

Figure 6.17: *The analysis of time, energy and power, by varying $\hat{\gamma}$ and maintaining $\hat{\alpha}$ and $\hat{\beta}$ equal to 1. In this case, we run one* `facesim` *task and one* `blackscholes` *task, using the McAfee algorithm.*

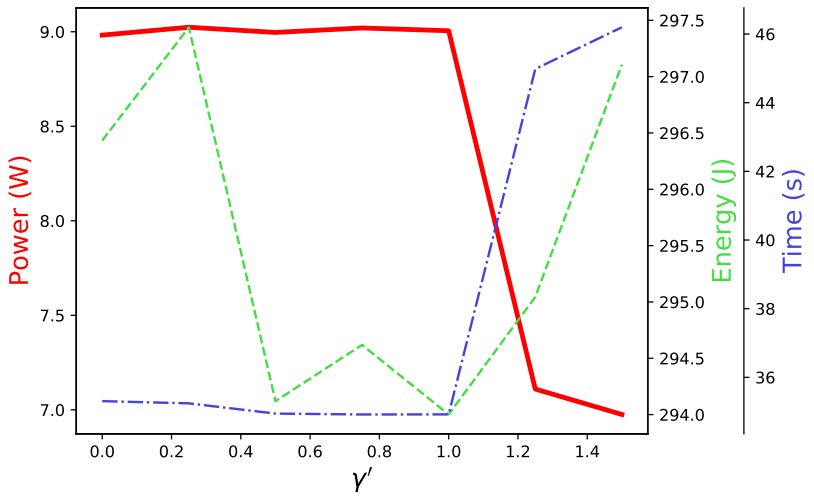

Figure 6.18: *The analysis of time, energy and power, by varying $\hat{\gamma}$ and maintaining $\hat{\alpha}$ and $\hat{\beta}$ equal to 1. In this case, we run one* `facesim` *task and one* `blackscholes` *task, using the VCG mechanism.*

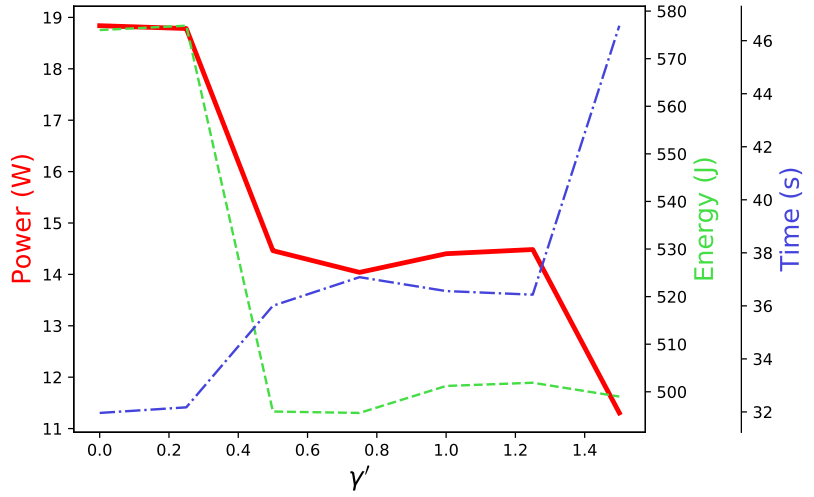

Figure 6.19: *The analysis of time, energy and power, by varying $\hat{\gamma}$ and maintaining $\hat{\alpha}$ and $\hat{\beta}$ equal to 1. In this case, we run two* `facesim` *tasks and two* `blackscholes` *tasks, using the McAfee algorithm.*

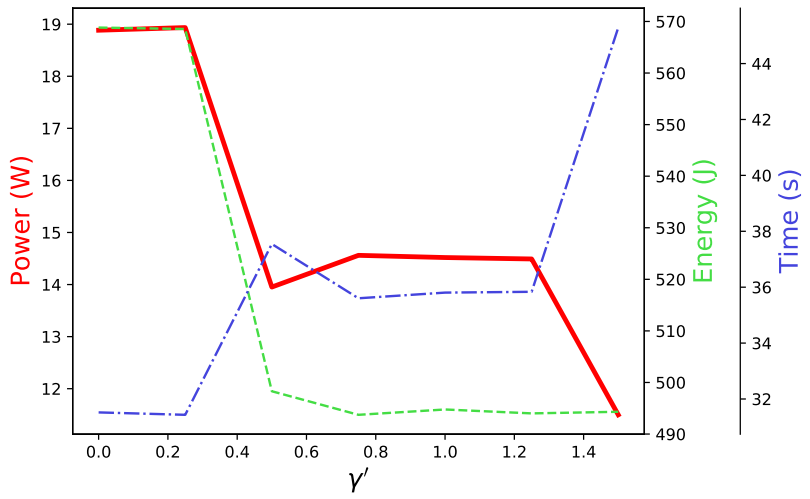

Figure 6.20: *The analysis of time, energy and power, by varying $\hat{\gamma}$ and maintaining $\hat{\alpha}$ and $\hat{\beta}$ equal to 1. In this case, we run two* `facesim` *tasks and two* `blackscholes` *tasks, using the VCG mechanism.*

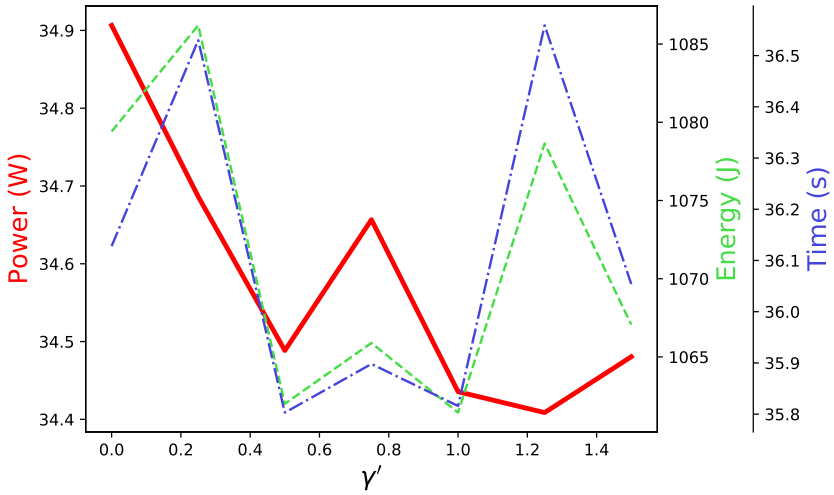

Figure 6.21: *The analysis of time, energy and power, by varying $\hat{\gamma}$ and maintaining $\hat{\alpha}$ and $\hat{\beta}$ equal to 1. In this case, we run four* `facesim` *tasks and four* `blackscholes` *tasks, using the McAfee algorithm.*

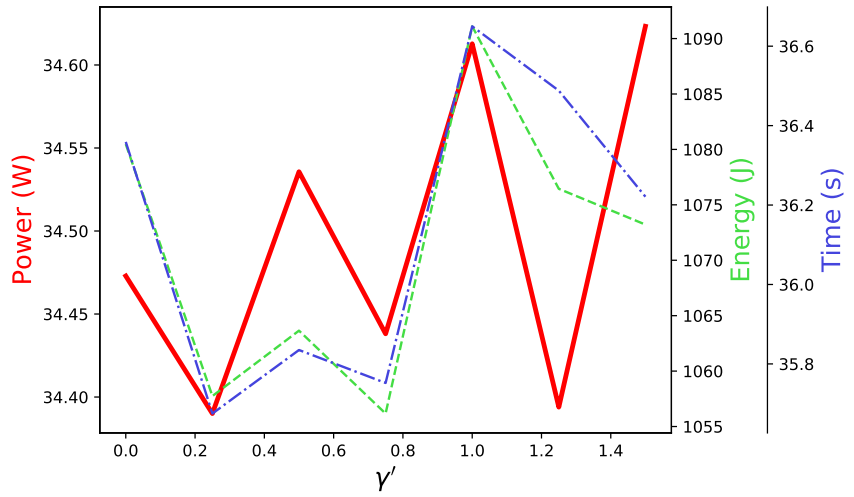

Figure 6.22: *The analysis of time, energy and power, by varying $\hat{\gamma}$ and maintaining $\hat{\alpha}$ and $\hat{\beta}$ equal to 1. In this case, we run four* `facesim` *tasks and four* `blackscholes` *tasks, using the VCG mechanism.*

The only outlier is the scenario depicted in Figure 6.22, where we can see a not expected evolution of the power, with remarkable variations and no clear trends, which do not respond to our assumptions. Similar to the other cost function coefficients, the cause is the too large number of tasks compared to the amount of resources.

In Figures 6.23, 6.24, 6.25 and 6.26, we can see how the *McAfee* algorithm works with and without *BarbequeRTRM* software. We studied the cases in which we have, as tasks, four different couples: one `blackscholes` task with zero `facesim` task, one `facesim` task with zero `blackscholes` task, two `blackscholes` tasks with zero `facesim` task, and two `facesim` tasks with zero `blackscholes` task. It is obvious that in general, as compared to the situation where the resource manager is involved, both the cases without *BarbequeRTRM*, i.e. the ones that in the graphs are represented by the green and red bars, expect a worst behaviour in terms of total energy, dynamic energy, time, average power and maximum power. In these figures, the case without *BarbequeRTRM* is shown both with the metrics average (*AVG*) and with the metrics maximum (*MAX*), because the *Linux* default allocation varies each time we launched the benchmarks.

Eventually, like the previous results, in Figures 6.27, 6.28, 6.29, and

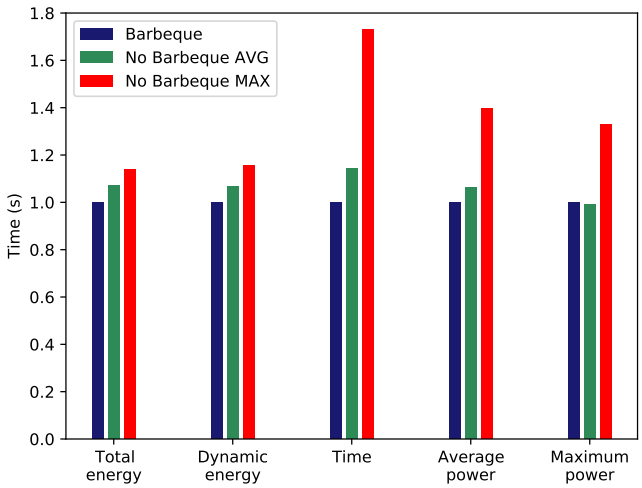

Figure 6.23: *The comparison of total energy, dynamic energy, time, average power and maximum power with and without BarbequeRTRM, in both the AVG and MAX situations, in the case of the McAfee algorithm and one* `blackscholes` *task.*

6.30, we can see how the *VCG* mechanism works with and without *BarbequeRTRM*. We studied the cases in which we have, as tasks, four different couples: one `blackscholes` task with zero `facesim` task, one `facesim` task with zero `blackscholes` task, two `blackscholes` tasks with zero `facesim` task, and two `facesim` tasks with zero `blackscholes` task. It is obvious that in general, as compared to the situation where the resource manager is involved, both the cases without the *BarbequeRTRM*, i.e. the ones that in the graphs are represented by the light-green and light-red bars, expect a worst behaviour in terms of total energy, dynamic energy, time, average power and maximum power.

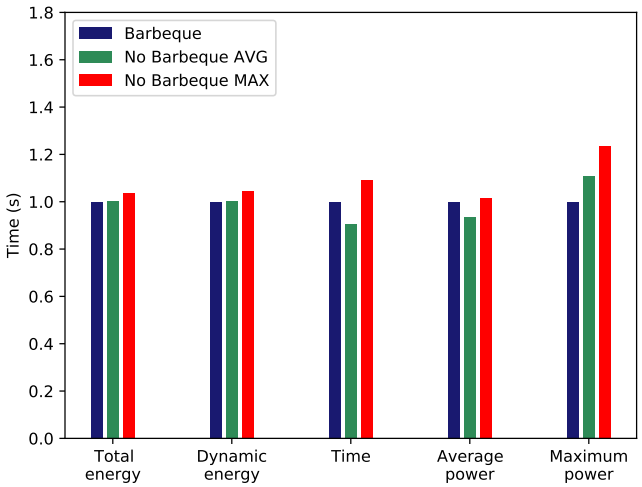

Figure 6.24: *The comparison of total energy, dynamic energy, time, average power and maximum power with and without BarbequeRTRM, in both the AVG and MAX situations, in the case of the McAfee algorithm and two* `blackscholes` *tasks.*

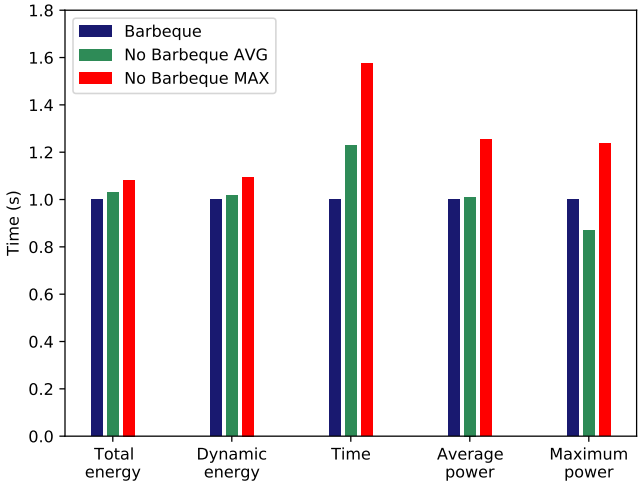

Figure 6.25: *The comparison of total energy, dynamic energy, time, average power and maximum power with and without BarbequeRTRM, in both the AVG and MAX situations, in the case of the McAfee algorithm and one* `facesim` *task.*

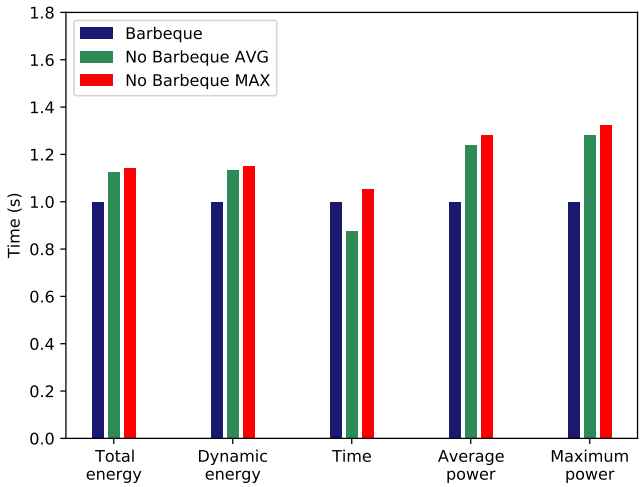

Figure 6.26: *The comparison of total energy, dynamic energy, time, average power and maximum power with and without BarbequeRTRM, in both the AVG and MAX situations, in the case of the McAfee algorithm and two* `facesim` *tasks.*

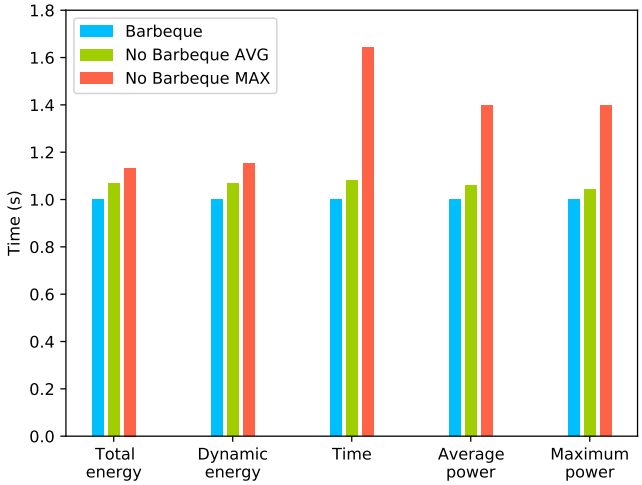

Figure 6.27: *The comparison of total energy, dynamic energy, time, average power and maximum power with and without BarbequeRTRM, in both the AVG and MAX situations, in the case of the VCG mechanism and one* `blackscholes` *task.*

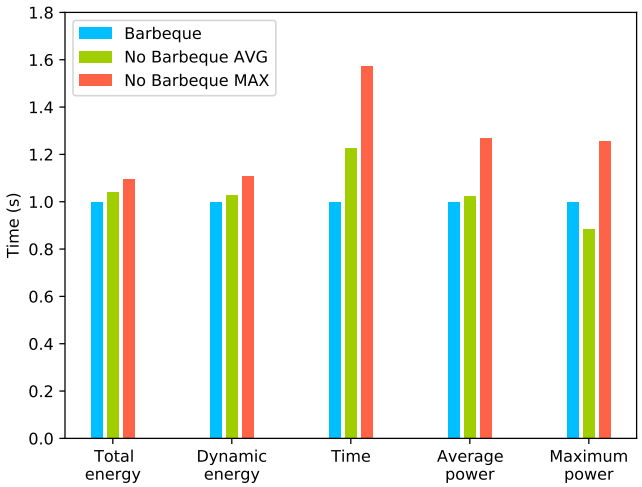

Figure 6.28: *The comparison of total energy, dynamic energy, time, average power and maximum power with and without BarbequeRTRM, in both the AVG and MAX situations, in the case of the VCG mechanism and one* `facesim` *task.*

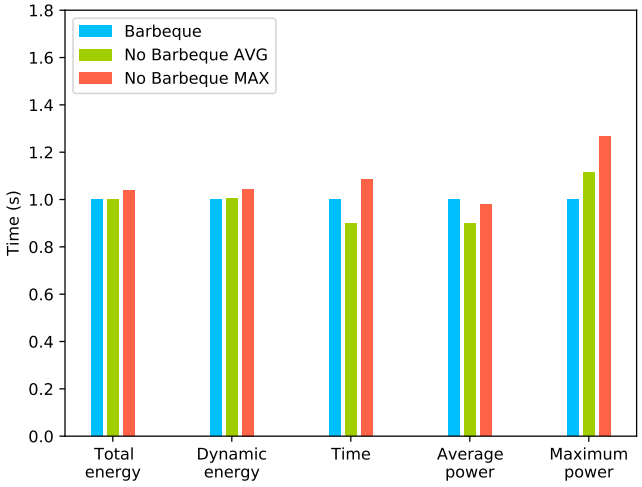

Figure 6.29: *The comparison of total energy, dynamic energy, time, average power and maximum power with and without BarbequeRTRM, in both the AVG and MAX situations, in the case of the VCG mechanism and two* `blackscholes` *tasks.*

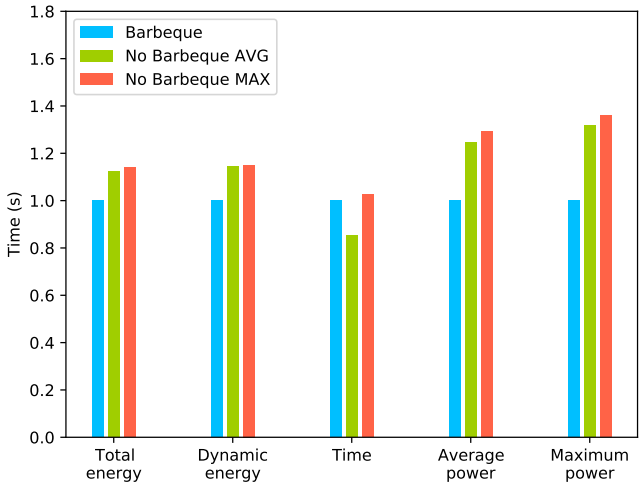

Figure 6.30: *The comparison of total energy, dynamic energy, time, average power and maximum power with and without BarbequeRTRM, in both the AVG and MAX situations, in the case of the VCG mechanism and two* `facesim` *tasks.*

6.5 Discussion

We evaluated two policies that implement, respectively, the mechanism behind a congestion game and the double auctions scenario, on a platform which simulates a heterogeneous system with four different `cpus` by running them at four different frequencies. After the calculation of the parameters $\hat{\alpha}$, $\hat{\beta}$, and $\hat{\gamma}$, we studied their impact on time, energy, and power, by considering several cases with a different workload, i.e. the number of `facesim` tasks and `blackscholes` tasks: in this way, we observed how the energy, the power, and the execution time react to those different situations.

In the case of the congestion games, even though we put ε[6] only equal to 0, by introducing in this way a substantial overhead caused by the policy to be executed, it is clear that the *BarbequeRTRM* allocations are generally better than a random policy. After the analysis of the variations regarding $\hat{\alpha}$, $\hat{\gamma}$, and $\hat{\beta}$, the first two parameters have an evident impact on, respectively, execution time and power, while the third cost function coefficient, instead, is not able to be effective on the energy, probably because the policy does not have a per-task characterization.

On the other hand, according to the double auction scenario, $\hat{\alpha}$, $\hat{\beta}$, and $\hat{\gamma}$ are more effective when the number of tasks is limited. Also in this case, we studied how the variation of the three parameters influence the result, by noticing that here $\hat{\beta}$ has a better effect compared to the congestion game case. By comparing this solution with the one of the default scheduler *Linux*, we could analyse how the policy improves the data, i.e. energy, execution time, and power, also simultaneously, once the tasks number is less than the resources amount, that is the system is not oversubscribed. Nevertheless, this limitation is not too much concerning because, in the case of *HPC* systems, the allocation of a tasks number equal to the amount of available processors on every single server is frequent and the nodes are never oversubscribed.

To summarize, the experimental facts show how the double auction policy works better when the node is not oversubscribed, while the congestion game policy, like expected from the theory, operates better when the nodes are congested, i.e. oversubscribed. Both the solutions have

[6]This variable represents how much we move away from the exact solution.

good or optimal results compared to the use of the default *Linux* scheduler allocation, in the execution time, the maximum power, the average power, and, with regard to double auctions case, also on the energy, both the dynamic and the total.

CHAPTER 7

Conclusions

In this final chapter, we present the possible future works in the field of the Game Theory applied to resource management and we take stock of our work.

7.1 Future works

First of all, further experiments can be performed, not only to observe how the congestion game's approximated algorithm (i.e. the case $\varepsilon > 0$) behaves on a real platform but also to measure the overhead introduced by the new policy. The overhead aspect was not considered in this thesis because in the used architecture it does not have a significant effect. In other systems, e.g. little embedded systems, the overhead of the policy could potentially represent a problem and, consequently, it may be worth investigating it.

With a focus on the double auctions games, it would be an interesting idea to use other state-of-the-art algorithms and comparing them to the two methods proposed in this thesis: the *McAfee* procedure and the *VCG*

mechanism.

Since we used a simulated heterogeneous platform, future experiments could run on a real heterogeneous system, e.g. the *MANGO* platform that is been described in Section 1.1.2, to increase the confidence on the results. Moreover, the policies could also be tested on architectures that are not *HPC*, e.g. mobile systems. Another aspect of this work that could be improved is to test them with a larger number of benchmarks and applications.

Other scenarios proposed by the Game Theory can be inspected to verify if they adapt themselves to the resource management problem: for example, if we will deal with a distributed version of *BarbequeRTRM*, it may be useful to study the so-called *Stackelberg Competition*, or, by referring to situations in which the tasks can decide autonomously the actions, i.e. there is no a centralized actor, we can study the model of *simultaneous action games*.

7.2 Conclusions

In this thesis, we presented the resource management problem in heterogeneous platforms and how it could be solved by using Game Theory algorithms. In particular, we chose two classes of games, the congestion games, and the double auctions games, and we applied them to the resource allocation problem on heterogeneous platforms. The policies designed in this work allocate the resources of a single node, differently from previous works dealing with Game Theory. In fact, state-of-the-art solution applied the resource management problem in a distributed computing context, i.e. how to allocate distributed resources in large environments (e.g. cloud computing).

We described both these algorithms and the modifications we introduced to use them as *BarbequeRTRM* policies. Then, these policies have been implemented, integrated into the *BarbequeRTRM* framework and tested through the state-of-the-art benchmarks. For each policy, we performed two classes of experiments: one in simulation and one on a real platform, which simulated a heterogeneous system. The simulation part verified the execution time and the solution optimality, from a theoretical standpoint. Instead, in the section devoted to the experiments on the real system, we checked the performance parameters, i.e. execution

time, power, and energy, by measuring them directly on the real system. The results showed that the double auctions games obtain better performance than the normal allocation of *Linux*, when the system is not oversubscribed, i.e. when the number of tasks does not exceed the number of computational units. On the other hand, the experimental results exhibited how the congestion games are more effective in the trade-off management of the above-mentioned metrics than the basic scheduler of *Linux* which focuses only on the execution time. The congestion games are more adequate also when the system is oversubscribed, that is when the number of tasks exceeds the number of computational units. The introduction of these policies opens new paths in *HPC*, which does not, usually, manage the situations with high congestion and tends to do not oversubscribe nodes.

Bibliography

[1] Giovanni Agosta, William Fornaciari, David Atienza, Ramon Canal, Alessandro Cilardo, José Flich Cardo, Carles Hernandez Luz, Michal Kulczewski, Giuseppe Massari, Rafael Tornero Gavilá, and Marina Zapater. The recipe approach to challenges in deeply heterogeneous high performance systems. *Microprocessors and Microsystems*, page 103185, 2020. doi:https://doi.org/10.1016/j.micpro.2020.103185.

[2] Giovanni Agosta, William Fornaciari, David Atienza, Ramon Canal, Alessandro Cilardo, Carles Flich, José Hernandez, Michal Kulchewski, Giuseppe Massari, Rafael Tornero, and Marina Zapater. Challenges in deeply heterogeneous high performance systems. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pages 428–435, 2019.

[3] Giovanni Agosta, William Fornaciari, Giuseppe Massari, Anna Pupykina, Federico Reghenzani, and Michele Zanella. Managing heterogeneous resources in hpc systems. In *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, pages 7–12, 2018.

[4] Patrick Bellasi, Giuseppe Massari, and William Fornaciari. A rtrm proposal for multi/many-core platforms and reconfigurable applications. In *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 2012.

[5] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81, 2008.

[6] H-L Chen, Jason R Marden, and Adam Wierman. On the impact of heterogeneity and back-end scheduling in load balancing designs. In *IEEE INFOCOM 2009*, pages 2267–2275. IEEE, 2009.

[7] Francis M. David, Jeffrey C. Carlyle, and Roy H. Campbell. Context switch overheads for linux on arm platforms. In *Proceedings of the 2007 Workshop on Experimental Computer Science*, ExpCS '07, pages 3–es, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1281700.1281703.

[8] Mian Dong, Tian Lan, and Lin Zhong. Rethink energy accounting with cooperative game theory. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 531–542. ACM, 2014.

[9] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. *IEEE Micro*, 32(3):122–134, 2012.

[10] J. Flich, G. Agosta, P. Ampletzer, D. A. Alonso, C. Brandolese, A. Cilardo, W. Fornaciari, Y. Hoornenborg, M. Kovač, B. Maitre, G. Massari, H. Mlinarič, E. Papastefanakis, F. Roudet, R. Tornero, and D. Zoni. Enabling hpc for qos-sensitive applications: The mango approach. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 702–707, 2016.

[11] José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Etienne Cappe, Alessandro Cilardo, Leon Dragić, Alexandre Dray, Alen Duspara, et al. Mango: Exploring manycore architectures for next-generation hpc systems. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 478–485. IEEE, 2017.

[12] José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Etienne Cappe, Alessandro Cilardo, Leon Dragić, Alexandre Dray, Alen Duspara, et al. Exploring manycore architectures for next-generation hpc systems through the mango approach. *Microprocessors and Microsystems*, 61:154–170, 2018.

[13] William Fornaciari, Giovanni Agosta, David Atienza, Carlo Brandolese, Leila Cammoun, Luca Cremona, Alessandro Cilardo, Albert Farres, José Flich, Carles Hernandez, Michal Kulchewski, Simone Libutti, José Maria Martínez, Giuseppe Massari, Ariel Oleksiak, Anna Pupykina, Federico Reghenzani, Rafael Tornero, Michele Zanella, Marina Zapater, and Davide Zoni. Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems. In *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, SAMOS '18, pages 187–194, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3229631.3239368.

[14] Kim Grüttner, Ralph Gürgen, Süren Schreiner, Fernando Herrera, Pablo Pe nil, Julio Medina, Eugenio Villar, Gianluca Palermo, William Fornaciari, Carlo Brandolese, Davide Gadioli, Emanuele Vitali, Davide Zoni, Sara Bocchio, Luca Ceva, Paolo Azzoni, Massimo Poncino, Sara Vinco, Enrico Macii, Salvatore Cusenza, John Favaro, Raúl Valencia, Ingo Sander, Kathrin Rosvall, Nima Khalilzad, and Davide Quaglia. Contrex: Design of embedded mixed-criticality control systems under consideration of extra-functional properties. In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 286–293, 2016.

[15] A. Guyot and S. Abou-Samra. Low power cmos digital design. In *Proceedings of the Tenth International Conference on Microelectronics (Cat. No.98EX186)*, pages IP6–I13, 1998.

[16] Mohammad Mehedi Hassan, Biao Song, and Eui-Nam Huh. Game-based distributed resource allocation in horizontal dynamic cloud federation platform. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 194–205. Springer, 2011.

[17] Qiang He, Guangming Cui, Xuyun Zhang, Feifei Chen, Shuiguang Deng, Hai Jin, Yanhui Li, and Yun Yang. A game-theoretical approach for user allocation in edge computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[18] Maha Jebalia, Asma Ben Letaifa, Mohamed Hamdi, and Sami Tabbane. A comparative study on game theoretic approaches for resource allocation in cloud computing architectures. In *2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 336–341. IEEE, 2013.

[19] S. Kim. *Game Theory Applications in Network Design*. Advances in Wireless Technologies and Telecommunication (2327-3305). IGI Global, 2014.

[20] Giuseppe Massari, Simone Libutti, Antoni Portero, Radim Vavrik, Stepan Kuchar, Vit Vondrak, Luca Borghese, and William Fornaciari. Harnessing performance variability: A hpc-oriented application scenario. In *2015 Euromicro Conference on Digital System Design*, pages 111–116. IEEE, 2015.

[21] Giuseppe Massari, Anna Pupykina, Giovanni Agosta, and William Fornaciari. Predictive resource management for next-generation high-performance computing heterogeneous platforms. In *International Conference on Embedded Computer Systems*, pages 470–483. Springer, 2019.

[22] Vladimir V Mazalov, Natalia N Nikitina, and Evgeny E Ivashko. Hierarchical two-level game model for tasks scheduling in a desktop grid. In *2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 541–545. IEEE, 2014.

[23] Lara Premi, Federico Reghenzani, Giuseppe Massari, and William Fornaciari. A game theory approach to heterogeneous resource management. In *Proceedings of*

the International Conference on Embedded Software Companion, EMSOFT '20. Association for Computing Machinery, 2020.

[24] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. A probabilistic approach to energy-constrained mixed-criticality systems. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[25] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. The Real-Time Linux Kernel: A Survey on PREEMPT_RT. *ACM Comput. Surv.*, 52(1), February 2019. doi:10.1145/3297714.

[26] Federico Reghenzani, Giuseppe Massari, Anna Pupykina, Giovanni Agosta, and William Fornaciari. Resource and memory management in mango heterogeneous system. 2017.

[27] Dimitrios Rodopoulos, Simone Corbetta, Giuseppe Massari, Simone Libutti, Francky Catthoor, Yiannakis Sazeides, Chrysostomos Nicopoulos, Antoni Portero, Etienne Cappe, Radim Vavrík, et al. Harpa: Solutions for dependable performance under physically induced performance variability. In *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 270–277. IEEE, 2015.

[28] Bernd Steinbach. *Recent Progress in the Boolean Domain*. Cambridge Scholars Publishing, 2014.

[29] Fei Teng and Frederic Magoules. Resource pricing and equilibrium allocation policy in cloud computing. In *2010 10th IEEE International Conference on Computer and Information Technology*, pages 195–202. IEEE, 2010.

[30] Daniel C Vanderster, Nikitas J Dimopoulos, Rafael Parra-Hernandez, and Randall J Sobie. Resource allocation on computational grids using a utility model and the knapsack problem. *Future Generation computer systems*, 25(1):35–50, 2009.

[31] John Von Neumann. On the theory of games of strategy. *Contributions to the Theory of Games*, 4:13–42, 1959.

[32] John Von Neumann, Oskar Morgenstern, and Harold William Kuhn. *Theory of games and economic behavior (commemorative edition)*. Princeton university press, 2007.

[33] Xin Xu and Huiqun Yu. A game theory approach to fair and efficient resource allocation in cloud computing. *Mathematical Problems in Engineering*, 2014, 2014.

[34] Bo Yang, Zhiyong Li, Shaomiao Chen, Tao Wang, and Keqin Li. Stackelberg game approach for energy-aware resource allocation in data centers. *IEEE Transactions on Parallel and Distributed systems*, 27(12):3646–3658, 2016.

[35] Deshi Ye and Jianhai Chen. Non-cooperative games on multidimensional resource allocation. *Future Generation Computer Systems*, 29(6):1345–1352, 2013.

[36] Michele Zanella, Giuseppe Massari, and William Fornaciari. Enabling run-time managed distributed mobile computing. In *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM '18, pages 39–44, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3183767.3183778.