The impact of cooperation on new high performance computing platforms

Daniel Cordeiro

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques-informatique**

Arrêté ministérial : 7 août 2006

Présentée par

Daniel de Angelis Cordeiro

Thèse dirigée par **Denis Trystram**

préparée au sein du **LIG, Laboratoire d'Informatique de Grenoble**
et de l'**École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

The impact of cooperation on new high performance computing platforms

Thèse soutenue publiquement le **9 février 2012**,
devant le jury composé de :

M. Bruno Gaujal
Directeur de recherche, INRIA, Président
M. Jean-Charles Billaut
Professeur des universités, Université François-Rabelais de Tours, Rapporteur
M^{me} Laetitia Vermeulen-Jourdan
Professeur des universités, Université Lille 1, Rapporteur
M. Wiesław Kubiak
Professeur des universités, Memorial University of Newfoundland, Examinateur
M. Philippe Olivier Alexandre Navaux
Professeur des universités, Universidade Federal do Rio Grande do Sul, Examinateur
M^{me} Fanny Pascual
Maître de conférences, Université Pierre et Marie Curie, Examinatrice
M. Denis Trystram
Professeur des universités, Grenoble INP, Directeur de thèse

Contents

Chapter 1

Introduction

Labs of the world, unite!!! This provocative title from the article written by Cirne et al. [Cir+06] allegorizes how the changes in computing and network in the last years transformed the quest for more computational power at low costs into a more collaborative endeavor.

The scientific community has today the unprecedented ability to combine different computational resources (possibly geographically spread around the globe) into a powerful distributed system capable of analysing massive data sets. The opportunities (and challenges) created by the use of these new technologies on large-scale scientific problems — usually referred as e-Science[1] — is changing how researchers engage in discovery [Fos11].

Computer science concepts, tools and theorems are now integrated on several aspects of the research process in several areas of knowledge. Climate and earth system research, fluid dynamics, genomics, proteomics, theoretical chemistry, astrophysics, nanostructure physics and high-energy physics are some examples [Emm+06] of areas that are continuously pushing the limits of what is possible in computing.

Computer scientists have long before recognized the increasing need for higher amounts of computational power demanded by research labs. The Condor project

[1] The UK National e-Science Centre defines [Tay06] e-Science as "the large scale science that will increasingly be carried out through distributed global collaborations enabled by the Internet. Typically, a feature of such collaborative scientific enterprises is that they will require access to very large data collections, very large-scale computing resources and high-performance visualization back to the individual user scientists."

[LLM88] was a pioneer in this trend. It was one of the first initiatives that focused on harvesting idle computer power on networks of workstations, aiming to deliver large amounts of processing capacity over long periods of time (*High Throughput Computing*).

With the development of networking technologies, the techniques used in Condor were soon enhanced to work with all machines connected to the Internet. In what was called *volunteer computing* [AF06] (also known as "peer-to-peer computing" or "global computing"), researchers started to work on how to harvest the idle computer power of these machines to solve specific problems.

Several academic and non-academic efforts explored the idea of volunteer computing in its earlier stages. Projects like "The Great Internet Mersenne Prime Search" (1996) [WK11] (that searches for Mersenne prime numbers), distributed.net (1997) [Inc10] (which demonstrated brute-force decryption), and the SETI@home project (1999) [And+02] (that analyzes large data sets of radio signals, searching for signs of extra terrestrial intelligence) showed how powerful such setups could be.

SETI@home is perhaps the most emblematic of all volunteer computing projects. Launched on May 17, 1999, SETI@home drew the attention not only from the scientific community, but also from the public in general. The idea that anyone in the world could collaborate to help in a scientific project has attracted over 6 million participants, located in 226 countries [Kor+11]. The project now uses the Berkeley Open Infrastructure for Network Computing (BOINC [And04]) Framework, a middleware for volunteer distributed computing, and is still one of the largest supercomputers in operation, currently averaging 3.5 PFLOP of actual performance.

At about the same time, Ian Foster and Carl Kesselman envisioned the concept of *grid computing* [FK04], in which computing is delivered on demand as a service. Like in the previous projects, different organizations team up to share their computational resources. But, in grid computing the entire platform is available as a service for all the participants. Each participating organization can act as either producer or consumer of resources.

As in the traditional "power grids", where different power plants are linked together and users get access to electricity without caring about where or how the electricity is actually generated, grid computing platforms allow users to use different computational resources, within and between organizations, with little or

no knowledge where those resources are physically located.

The grid computing vision is now a reality and is being used in both academic and commercial projects. Foster [Fos11] presented some contemporary examples that show the magnitude of the projects using grid computing technology today. Among the academic initiatives, we can cite the Large Hadron Collider (LHC) Computer Grid (that regularly distributes tens of terabytes to research teams worldwide), the Earth System Grid (that gives access to climate simulation data for more that 25,000 users), the US InCommon trust federation (that allows more than 5 million people to access remote resources using local credentials), among others. Among the commercial systems, we have today commercial cloud providers delivering on-demand cycles and storage at scales unachievable in academic settings.

Although volunteer computing and grid computing systems are very similar in the sense that both allow the creation of a platform composed of a huge number of computational resources, they completely differ on how the platform can be used to execute a project.

In volunteer computing, each volunteer actually chooses the project that s/he want to help. The execution of the project relies on the goodwill of the volunteers, that usually do not earn anything for their help. Volunteers are usually anonymous and cannot be made accountable for the quality of the resources or the results they provide.

In grid computing, however, the interactions between the participants are more complex. An organization chooses to integrate a grid computing platform not only to contribute with its computational resources to other projects. Their users also expect to be able to execute their own jobs more efficiently with the help of the others.

A conflict of interests may arise if the resources of the grid platform are not shared in a fair manner. For example, if an organization presumes that less resources are allocated to its own projects if compared to the resources allocated to the others, then the organization may be compelled to leave the grid and to stop sharing its resources.

Another example would be a case where one of the organizations misbehaves somehow in order to improve the execution of its own projects. For instance, a malicious organization could deny the execution of any project other than its own. Such organizations must be held accountable for their misconduct. Depending on

the circumstances, the organization could have the access to others' shared resources refused or even be legally charged for its misbehavior.

Large research projects like the ones cited by Foster [Fos11] can avoid these problems by creating and operating dedicated grid infrastructures, where every detail of the execution can be controlled. But in order to deliver a fully collaborative grid computing platform — a platform where research teams with different budgets, sizes and objectives can have the same advantages and competitiveness of the large research projects — each user's own interest must be respected.

In this thesis, we study how to encourage collaboration between such set of heterogeneous users and organizations. We model variants of this problem as multi-objective scheduling problems and, using different theoretical tools, we study how the individual behaviour of these users impacts both the performance perceived by them and the global efficiency of the platform.

1.1 Outline of the thesis and contributions

This thesis is divided in seven chapters, including this introductory chapter and a final chapter presenting our concluding remarks. The contributions described in each chapter can be summarized as follows.

Chapter 2 presents the formal framework used in this thesis. We define the problem of scheduling in modern platforms with computational resources shared by different users and organizations. We present the related works, a brief introduction to Game Theory and to concepts of multi-objective optimization.

In Chapter 3 we consider the problem of scheduling on computing platforms composed of several independent organizations, known in the literature as the Multi-Organization Scheduling Problem (MOSP). We assume that each organization provides both resources and jobs to be executed and has its own performance objectives. We are interested in the best way to minimize the makespan on the entire platform, while ensuring that every organization will always have incentive to collaborate with others. We study the complexity of the MOSP problem with two different local objectives — makespan and average completion time — and show that, as expected, MOSP is strongly NP-Hard in both cases.

We formally define a selfishness notion, by means of restrictions on the schedules. We prove that selfish behavior of the organizations imposes a lower bound of 2 on the approximation ratio for the global makespan. We present various approximation algorithms of ratio 2 which validate selfishness restrictions. These algorithms are experimentally evaluated through simulations, exhibiting good average performances and presenting good fairness to organizations' local objectives.

This joint work with Johanne Cohen and Frédéric Wagner resulted in two publications, one in the proceedings of the Euro-Par conference in 2010 and another one as an extended journal version in 2011:

- Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. "Analysis of Multi-Organization Scheduling Algorithms". In: *The 16th International Conference on Parallel Computing (Euro-Par)*. Ed. by Pasqua D'Ambra, Mario Guarracino, and Domenico Talia. Vol. 6272. Lecture Notes in Computer Science. Heidelberg: Springer, 2010, pp. 367–379. DOI: 10.1007/978-3-642-15291-7_34

- Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. "Multi-organization scheduling approximation algorithms". In: *Concurrency and Computation: Practice and Experience* 23 (17 2011), pp. 2220–2234. DOI: 10.1002/cpe.1752

In Chapter 4 we study how limited cooperation can impact the quality of the schedule obtained by multiple independent organizations in a typical grid computing platform. In this slightly relaxed version of MOSP, we study how much the collectivity can improve the makespan over the entire platform when each organization tolerates a bounded degradation on the makespan of its own jobs.

More precisely, the technical contributions are the following. First, we improve the existing inapproximation bounds for this problem proving that what was previously thought as not polynomially approximable (*unless* $P = NP$) is actually not approximable at all. Then, we present two algorithms that solve the problem with approximation ratios of (2; 3/2) and (3; 4/3) respectively. This means that when using the first (second) algorithm, if an organization tolerates that the completion time of its last job cannot exceed twice (three times) the time it would have obtained

by itself, then the algorithm provides a solution that is a 3/2-approximation (4/3-approximation) for the optimal global makespan. Both algorithms are efficient since their performance ratios correspond to the Pareto optimal solutions of the previously defined instances.

This joint work with Pierre-François Dutot and Grégory Mounié resulted in the following publication:

- Daniel Cordeiro, Pierre-François Dutot, Grégory Mounié, and Denis Trystram. "Tight Analysis of Relaxed Multi-Organization Scheduling Algorithms". In: *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. (Anchorage, AL, USA). Los Alamitos, CA, USA: IEEE Computer Society, May 2011, pp. 1177–1186. DOI: 10.1109/IPDPS.2011.112

Chapter 5 describes our efforts to study the members of a grid computing system as truly independent and rational entities, capable of taking decisions that can help the scheduler to deliver the best performance both from the local and global point of views.

We modeled MOSP as a non-cooperative game where each agent is responsible for assigning all jobs belonging to a particular organization to the available processors. The local scheduling of these jobs is defined by coordination mechanisms that first prioritize local jobs and then schedule the jobs from others according to some given priority. When different priorities are given individually to the jobs — like in classical scheduling algorithms such as LPT or SPT — then no pure ϵ-approximate equilibrium is possible for values of ϵ less than 2. We also prove that even deciding whether a given instance admits or not a pure Nash equilibrium is co-NP hard. When these priorities are given to entire organizations, we show the existence of an algorithm that always computes a pure ρ-approximate equilibrium using any ρ-approximation list scheduling algorithm. Finally, we prove that the price of anarchy of the MOSP game using this mechanism is asymptotically bounded by 2. This work resulted in the following publication:

- Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. "Coordination Mechanisms for Selfish Multi-Organization Scheduling". In: *Proceedings*

of the 18th annual IEEE International Conference on High Performance Computing (HiPC). (Bangalore, India). To appear. Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2011

A slightly change in the perspective is presented on Chapter 6. We explore a different form of collaboration between users sharing a set of common resources. We study the problem known as Multi-Users Scheduling Problem (MUSP). In this chapter, we present a generic method to address problems where the optimization function is monotonic with arguments that can be optimized. Essentially, the method enumerates the frontier of best compromise solutions between these arguments and selects the solution that brings the best value for the function to optimize. Such an approach would be exponential in the general case. However, we show how classical tools from the approximation theory help to address the problem in both theoretical and practical perspective.

During this thesis, other scientific works were developed on collaboration with members from other research teams. These works are also related with scheduling or high performance computing on parallel platforms, but they were not included in this text because their thematic greatly differs from the main subject of this thesis. The first work [Kol+08] was a German-Brazilian collaboration with Mariana Kolberg and Gerd Bohlender (Karlsruher Institut für Technologie), Luiz Gustavo Fernandes (Pontifícia Universidade Católica do Rio Grande do Sul) and Alfredo Goldman (Universidade de São Paulo) about the efficient execution of a verified method for solving linear systems. The second work [Cor+09; Cor+10] was an effort of researchers from the Laboratoire d'Informatique de Grenoble — a collaboration with Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frédéric Wagner — to raise awareness among researchers working with scheduling about the importance of the choice of the right synthetic workload for their simulations. The third work [Pil+10; Pil+11] explores the problem of load balancing on machines NUMA (with Non-Uniform Memory Access), a collaboration with Laércio L. Pilla and Philippe O. A. Navaux (Universidade Federal do Rio Grande do Sul), Christiane Pousa Ribeiro and Jean-François Méhaut (Laboratoire d'Informatique de Grenoble), Abhinav Bhatele and Laxmikant V. Kalé (University of Illinois at Urbana-Champaign).

Chapter 2

Background

In this work, we study how to encourage collaboration between organizations and users in parallel and distributed platforms through the use of multi-objective scheduling. This chapter introduces the theoretical framework that serves as basis for the results presented in the next chapters.

Essentially, Section 2.1 presents a brief introduction to classical scheduling theory and some of the well-known results in single-objective scheduling used in this work. The definitions and results of this section are based mainly on the books from Leung [Leu04] or Brucker [Bru07].

Section 2.2 presents basic concepts of multi-objective optimization. Based on the surveys from [Bra+08] and [Dut+09], we briefly present the concepts that allow us to study the trade-offs related to choosing a schedule that intents to satisfy the different needs of each individual.

Finally, Section 2.3 present basic concepts of pure-strategy games and how the interactions between individuals are analyzed assuming that they reason strategically. These concepts were extracted mainly from the books by Osborne and Rubinstein [OR94] and Nisam, Roughgarden, Tardos, and Vazirani [Nis+07].

2.1 Classical scheduling theory

Generally speaking, the problem of scheduling corresponds to the allocation of a set of scarce resources to activities with the objective of optimizing one (or more)

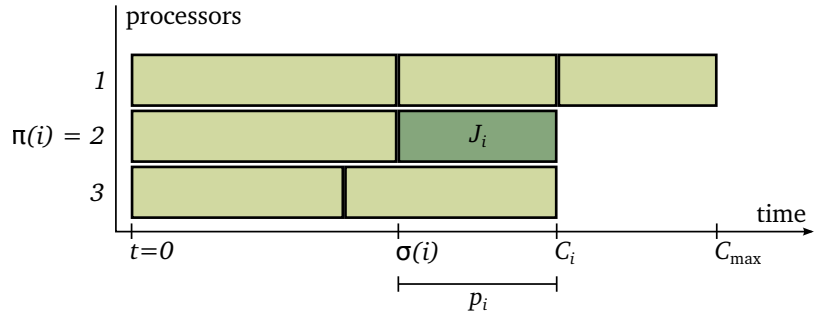


Figure 2.1: Example of a Gantt chart representing a scheduling following the LPT rule.

performance measures.

Scheduling theory has been used in different contexts. Resources and activities can be different according to the context. For instance, in an assembly plant the resources can be the machines and activities one of the operations in the manufacturing process. On an airport, the resources can be the runways and the activities the landings and take-offs, and so on.

In the context of parallel and distributed computing, resources are the processors available on the platform and activities are the jobs (applications or sub-tasks) that must be executed on these processors. We denote by m the number of processors available at the platform and by n the number of jobs to be executed. Thus, in this context, we say that a scheduler must allocate the set $\mathcal{M} = \{1, 2, \ldots, m\}$ of available processors to the set $\mathcal{J} = \{J_i, J_2, \ldots, J_n\}$ of jobs that will be executed on the platform.

A schedule is defined by a pair of functions $\pi : \mathcal{J} \mapsto \mathcal{M}$ and $\sigma : \mathcal{J} \mapsto \mathbb{N}$, that defines that job J_i will be executed on processor $\pi(i)$, starting at time $\sigma(i)$. Any valid schedule must execute each job uninterruptedly at least one time and a given processor cannot execute more than one job at the same time. Figure 2.1 illustrates an example of a simple schedule represented graphically with a Gantt chart.

The different variations that a schedule problem may present can be summarized by the three-field notation $\alpha \mid \beta \mid \gamma$ introduced by Graham, Lawler, Lenstra, and Rinnooy Kan [Gra+79]. The α field characterizes the type of the resource that will execute the activity, the β field describes details of the jobs and which scheduling constraints characterize a valid solution, and the γ field describes the objective(s) function(s) to be optimized.

In our work, we focus the study on scheduling in parallel and identical machines. This setup is represented as Pm in the α field of Graham's notation and it means that any job will execute exactly on the same way in any of the m available machines, i.e, each job J_i will be executed by $p_i \in \mathbb{N}$ units of time (its *processing time*), independently of the processor allocated to it. The notation admits the omission of the m (notating it only by P instead of Pm, meaning that the number of processors is arbitrary and will be given as part of the instance description). Other common setups for parallel and distributed systems include uniform machines (Qm), where the machines may have different speeds, and unrelated machines (Rm), where a same job can take a different time to execute depending on the machine that was allocated to it.

Usually we consider that all jobs are ready to run at the beginning of the execution. If it is not the case, the presence of the notation r_j (*release date*) at the β field indicates that job J_j cannot start its execution before time r_j. Similarly, the notation d_j (*due date*) indicates that a job J_j should not finish its execution after time d_j.

Unless stated otherwise, in this work we study *sequential* jobs, i.e., jobs that use only one processor at a given time. There are models in the literature for multiprocessor jobs. These models usually distinguish the jobs between three different kinds: (i) *rigid tasks*, where the job must always use a fixed number of processors; (ii) *moldable tasks*, where the number of processors that will execute the job is constant over time and chosen at scheduling time; and (iii) *malleable tasks*, where the scheduler is free to change the number of processors allocated to the job during the execution.

If all information about the jobs are known by the scheduler before a schedule is constructed (which is the case in all problems studied in this work), then we say that we have an *offline* scheduling problem. In contrast, *online* scheduling restricts the problem to the case where the schedule is constructed based on the currently available information. In particular, the jobs' characteristics are not known until they arrive.

Scheduling problems are studied in order to optimize some performance metric on the obtained schedule. In parallel and distributed systems, this metric usually is a function in terms of the completion times of the jobs that must be minimized.

The main objective functions of interest in this work are the *makespan* and the *average completion time*. They are indicated on the γ field as $C_{\max}$ and $\sum C_i$, respectively. Notating the completion time of a job J_i as $C_i = \sigma(i) + p_i$, we define the

makespan ($C_{\max}$) as being the $\max_i\{C_1,\ldots,C_n\}$, i.e., the completion time of the last job to finish in the schedule. The average completion time $\sum C_i$ is defined as $\frac{1}{n}\sum_i C_i$.

If f is the objective function to be minimized, we usually denote as $f^* = \min_{\pi,\sigma} f(\pi,\sigma)$ the minimum value of the function.

Figure 2.1 illustrates the basic notation of scheduling used in this work.

Classical results in single-objective scheduling

Researchers have already studied the computational complexity and efficient algorithms for many different combinations of the possible $\alpha \mid \beta \mid \gamma$ configurations.

Scheduling is usually a time-expensive operation. However, there are efficient algorithms that can be used to compute a schedule with good performance in a reasonable amount of time. The most prominent examples of such algorithms are *List Scheduling* algorithms.

Introduced by Graham [Gra66], List Scheduling algorithms comprises a wide class of algorithms used to calculate schedules aiming different performance objectives. The basic working principle of List Scheduling algorithms is to greedily assign the next job from an ordered list to the least loaded processor, scheduling it as earlier as possible.

The order of the list of jobs is defined by some priority rule chosen according to the performance objective to be optimized. Two classical algorithms that are to be noted are the *Longest Processing Time first* (LPT) algorithm, where the list is created by sorting the jobs by non-increasing processing times, and the *Shortest Processing Time first* (SPT) algorithm, where jobs are sorted in non-decreasing processing times (in both cases, ties can be broken arbitrarily).

Finding an optimal assignment that optimizes the performance objective are computationally easier in some problems than others. For instance, the problem $P \mid\mid \sum C_i$ can be solved optimally in polynomial time by the Shortest Processing Time first algorithm.

Theorem 2.1. *The SPT rule is optimal for* $P \mid\mid \sum C_i$.

Proof. The proof (adapted from [Pin08]) follows from the analysis of the cost function. If we schedule all jobs $J_1, J_2, \ldots, J_n$ in this order on the same processor, the

sum of completion times can be expressed as:

$$\sum C_i = np_1 + (n-1)p_2 + (n-2)p_3 + \ldots + 2p_{n-1} + p_n$$

This shows that there are n integer coefficients $(n, n-1, n-2\ldots, 2, 1)$ that must be assigned to n different processing times. In order to minimize this expression, we must assign the highest coefficient (n) to the smallest processing time $(\min_i p_i)$, the second highest coefficient $(n-1)$ to the second smallest processing time, and so on. In other words, it means that if $p_i \leq p_2 \leq \ldots \leq p_n$ then the $\sum C_i$ is minimum. This implies that SPT is optimal for $1 \,||\, \sum C_i$.

In the case of m parallel machines there are $m \times n$ integer coefficients (m coefficients equal to n, m coefficients equal to $n-1, \ldots, m$ coefficients equal to 1), and each job must be assigned to one of these coefficients in such way that it minimizes the sum of the products. Assume that n/m is an integer (if not, just add a number of dummy jobs with processing time equal to zero — which does not change the value of the $\sum C_i$ — that turns n/m an integer.)

It is easy to see that in order to minimize the sum of the products, we must assign the set of m longest processing times to the m smallest coefficients (there are m coefficients equal to one), the set of second m longest processing times must be assigned to the second smallest coefficients (the m coefficients equal to two), and so on. This results in m longest jobs each being processed in different machines and so on.

This does not describes only SPT, but a class of scheduling algorithms that includes SPT. To show that this class includes SPT, remember that the smallest job must go to processor 1 at time zero, the second smallest one on machine 2, and so on; the $(m+1)^{\text{th}}$ follows the smallest job on machine 1, $(m+2)^{\text{th}}$ follows the second smallest job on machine 2 and so on. In other words, SPT maintains this "layered" structure of the schedule and it is in the class above. It is easy to verify that SPT corresponds to an optimal assignment of jobs to coefficients. □

The scheduling problem with *makespan* as objective in its general form is harder. With two identical processors, the scheduling problem $P2 \,||\, C_{\max}$ is strictly equivalent to the PARTITION problem [GJ79], which is weakly NP-hard. If the number of processors is fixed and given as part of the instance, then we can show that there

exists a reduction for $P \mid\mid C_{\max}$ from the 3-PARTITION problem, showing that this scheduling problem is strongly NP-hard. Thus, unless P = NP, there is no polynomial-time algorithm to solve this problem.

The lack of polynomial-time algorithms to these problems leads researchers to study approximation algorithms: fast (polynomial) algorithms that produce solutions that are provably close to optimal. An algorithm is a ρ-approximation [Hoc97] of an objective f if the solution S returned by the algorithm respects the following inequality: $f(S) \leq \rho f^*$.

List Scheduling algorithms are an interesting class of algorithms not only because it is usually fast to compute a solution. They are also interesting because they provide a scheduling with a guaranteed approximation ratio for the makespan of its solutions. Graham [Gra66] showed that:

Theorem 2.2. *Any List Scheduling algorithm is a* $\left(2 - \frac{1}{m}\right)$*-approximation algorithm for the* $P \mid\mid C_{\max}$ *scheduling problem.*

Proof. We want to show that the $C_{\max}$ obtained by any List Scheduling algorithm, will be at least $\left(2 - \frac{1}{m}\right)$ times the value of the optimal makespan (denoted by $C^*_{\max}$).

First, note that if we want to minimize the makespan of a schedule, the best case possible would to equally divide the total work to be done among the available processors. So we have that $C^*_{\max} \geq \frac{\sum p_j}{m}$.

Let σ be the scheduling returned by the List Scheduling algorithm and J_ℓ the last job to finish its execution in this schedule. Then we have that the $C_{\max}$ of this schedule is given by the completion time of this job, i.e., $C^\sigma_{\max} = \sigma(\ell) + p_\ell$. We know that at the time that the algorithm chose to start the last job, all processors are busy (otherwise the List Scheduling algorithm would have scheduled the last job earlier). Since no machine is idle, we have that the total processing time of all jobs apart J_ℓ must be at least equal to $m\sigma(\ell)$. So we have that:

$$\sigma(\ell) \leq \frac{(\sum p_j - p_\ell)}{m} \leq C^*_{\max} - \frac{p_\ell}{m}$$

Which implies that:

$$C^\sigma_{\max} = \sigma(\ell) + p_\ell \leq C^*_{\max} + p_\ell \left(1 - \frac{1}{m}\right) \leq \left(2 - \frac{1}{m}\right) C^*_{\max}$$

□

We can show with a simple example that this bound for the approximation ratio is tight. Take the case where we have $m = 2$ processors and three jobs with processing times equal to 1, 1, and 2. The List Scheduling algorithm will schedule job 1 on processor 1, job 2 on processor 2, and job 3 on processor 1 or 2. The obtained $C_{\max}$ is $1 + 2 = 3$, while a $C^*_{\max} = 2$ would have been obtained if we schedule jobs 1 and 2 on the same processor and job 3 in the other processor. A more complex example can show that it is also true for any value of m.

Note that the SPT algorithm mentioned earlier is also a List Scheduling algorithm, where the list is computed by sorting the jobs in non-decreasing processing times. This means that the algorithm produces an optimal scheduling for the users interested in the $\sum C_i$, while, at the same, producing a $\left(2 - \frac{1}{m}\right)$-approximation for the makespan.

Graham [Gra69] also studied the approximation bounds for the Longest Processing Time first (LPT) algorithm. He showed that the approximation ratio of LPT is equal to $\frac{4}{3} - \frac{1}{3m}$, which is actually better than the general case for List Scheduling algorithms. He showed that the bound for LPT is also tight.

Hochbaum and Shmoys [HS88] presented a family of polynomial time approximation algorithms $\{A_\epsilon \mid \epsilon > 0\}$. Each algorithm A_ϵ generates a schedule with a approximation ratio for the makespan of $(1 + \epsilon)$ and runs in time $O\left((n/\epsilon)^{1/\epsilon^2}\right)$. This family of algorithms is a *polynomial time approximation scheme* (PTAS) for the $P \mid\mid C_{\max}$ problem.

More recently, an interesting variant of LPT was studied for the case where all jobs are available at time zero, but some machines are not. This generalization of the problem of scheduling on multiple processors was first studied by Lee [Lee91], when studying setups where jobs arrive in batches on a periodic basis and it is interesting to begin scheduling each batch before the completion of the previous batch. Lee proposed an algorithm called Modified LPT (MLPT), with a guaranteed approximation ratio of $\frac{4}{3}$ that was later shown [Guo98] to be tight. Kellerer [Kel98] extended this result presenting more a sophisticated approximation algorithm with a worst case bound of $\frac{5}{4}$ for the problem.

2.2 Multi-objective optimization

The problem of scheduling in collaborative platforms is naturally a multi-objective problem. The quality of any proposed schedule must be evaluated according to the expectations of each participant. These expectations — usually modeled as objective functions — may be very different from participant to participant and even conflicting with each other.

Definition 2.3 (Multi-objective optimization scheduling problem)**.** Given a set of jobs $\mathcal{J}$ to be scheduled on the available processors of the set $\mathcal{M}$, the multi-objective scheduling problem consists in finding a solution $S = (\pi, \sigma)$ that respects some constraints and that minimizes the functions $f_1(\pi, \sigma)$, $f_2(\pi, \sigma)$, ..., $f_k(\pi, \sigma)$.

Multi-objective problems have a different notion of what means the optimal solution. Typically a multi-objective problem does not have a single solution that optimizes all objectives at once, but a multitude of solutions with different compromises. In the single-objective case it is easier to compare the different solutions. The total order induced by the single objective function is sufficient to define which solution is better. In multi-objective problems this is not possible.

In order to capture this notion of trade-off between different objectives, we use the concept of *Pareto dominance* [Voo03].

Definition 2.4 (Pareto dominance)**.** Given S and S' two solutions for a multi-objective optimization problem:

$$\begin{aligned} S \text{ Pareto dominates } S' \iff & \forall l \in \{1, \ldots, k\},\ f_l(S) \leq f_l(S') \\ & \text{and } \exists l \in \{1, \ldots, k\} \mid f_l(S) < f_l(S') \end{aligned}$$

The intuition is that a solution S Pareto dominates a solution S' if S is at least as good as S' in all objectives and strictly better than S' in at least one objective. Thus, instead of looking for a single optimal solution, we are interested in finding solutions that are not Pareto dominated by any other solution. Such solutions are known in the literature as *Pareto optimal* (or *Pareto efficient*) solutions:

Definition 2.5 (Pareto optimality)**.** A solution S is said to be *Pareto optimal* if and only if there is no other solution $S' \neq S$ such that S' Pareto dominates S.

The set of Pareto optimal solutions represents the range of reasonable "optimal" solutions in the sense that there is no better solution that improves one objective without degrading another objective. They are precisely the optimal solutions for all possible global "utility" functions that depend monotonically on the different objectives [DY07]. This set of solutions is called the *Pareto set* (also known in the literature as *Pareto curve*, *Pareto front* or *Pareto frontier*).

Definition 2.6 (Pareto set)**.** The Pareto set P is the set of all Pareto optimal solutions for a multi-objective problem.

Multi-objective optimization problems are usually NP-hard due to the fact that the Pareto set is typically exponential in size even in the case of two objectives [Zar05]. Thus, in practice, multi-objective problems are treated using different approaches, such as:

- studying approximate (guaranteed) versions of the Pareto set;
- optimizing one objective while bounding the others (constrained approach);
- normalizing the objectives, by introducing a utility (often non-linear) function on the objectives (normalization approach);
- approximating all the objectives at once from a known (but not always feasible) lower bound (zenith approximation) or upper bound (nadir);

The concept of approximation algorithms for multi-objective problems can be extended from the single-objective case in a straightforward way. Instead of having an optimal point of reference, we extend the notion by saying that an algorithm is a ρ-approximation of a solution S if the algorithm always give solutions that are a ρ-approximation of S.

Definition 2.7. Given S and S' two solutions of a multi-objective problem. S is a $\rho = (\rho_1, \rho_2, \ldots, \rho_k)$-approximation $(\forall l, \rho_l \geq 1)$ of S' if and only if $\forall l \in \{1, \ldots, k\}$, $f_l(S) \leq \rho_l f_l(S')$.

And since it is usually not possible to enumerate all solutions on the Pareto set because of its exponential size, we are also interested in approximations from sets of solutions.

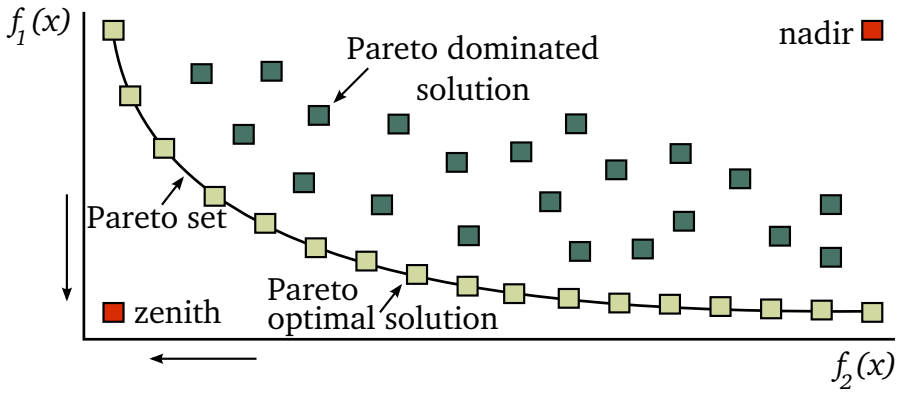


Figure 2.2: Example of a Pareto set for a two-objective problem.

Definition 2.8. A set of solutions P is a $\rho = (\rho_1, \rho_2, \ldots, \rho_k)$-approximation ($\forall l, \rho_l \geq 1$) of a set of solutions P' if and only if $\forall S' \in P'$, $\exists S \in P \mid S$ is a ρ-approximation of S'.

Figure 2.2 illustrates the concepts presented in this section.

2.3 Game theory

Game theory [Nis+07; OR94] comprises a large set of analytical tools designed to study interactions between individuals free to make a set of choices in order to pursue some specific objective. Game theory have been successfully applied in several areas of knowledge, such as biology, computer science, economics, political sciences, psychology, etc.

In game theory, we study problems in which these decision-makers are assumed to be rational individuals, interested in some well-defined objective. These individuals reason strategically, i.e., they are expected to take decisions towards their goals based on their knowledge or expectations about the decisions taken by other decision-makers.

We call *game* the set of rules that define the strategic interactions between the *players* (the decision-makers). The game describes constraints on the actions that the players can take and the players' objectives, but it does not specify which actions the players do take to achieve their objectives. Game theory studies the outcomes that may result from the interactions that follow the rules defined in the game.

More formally, a game consists of a set of n players $\{1, 2, \ldots, n\}$. Each player i has its own set S_i of possible strategies. To play the game, each player i selects a strategy $s_i \in S_i$. We will use $s = (s_1, \ldots, s_n)$ to denote the vector of strategies, with the strategies selected by each player and $S = \times_i S_i$ to denote all possible ways in which players can choose strategies. We also denote by s_{-i} the $(n-1)$ dimensional vector of strategies played by all players other than i.

The vector of strategies $s \in S$ selected by the players determines the outcome of each player. Each player chooses its strategy based on a preference ordering on the outcomes. The simplest way to specify these preferences is by assigning a value to each outcome. It is usually done with the use a payoff function, denoted by an utility function $u_i : S \mapsto \mathbb{R}$ or a cost (or disutility) function $c_i : S \mapsto \mathbb{R}$, with $u_i(s) = -c_i(s)$. When convenient, we will also use the notation $u_i(s_i, s_{-i})$ (or $c_i(s_i, s_{-i})$) to express the payoff of player i.

It is important to note that the payoff function of player i depends not only on its own strategy (s_i), but also on the strategies chosen by all players (s).

In this work, we focus on the case where strategies are selected in a deterministic fashion by the players. A deterministic decision made by the player is called *pure strategy*. We can also define randomized strategies, where each player can pick a probability distribution over its set of possible strategies; such choice is called a *mixed strategy*. From now on, unless stated otherwise, whenever we refer to "strategy" we are referring to a "pure strategy".

Definition 2.9 (Dominant strategy solution)**.** We say that a game has a *dominant strategy solution* if each player has a unique best strategy, independent of the strategies played by the other players. More formally, a strategy vector $s \in S$ is a dominant strategy solution, if for each player i, and each alternate pure-strategy vector $s' \in S$, we have:

$$u_i(s_i, s'_{-i}) \geq u_i(s'_i, s'_{-i})$$

Games with dominant strategy solutions are rare. In most of the cases, we seek for game-theoretic solutions in which individual players reach a consensus, while optimizing their own payoff. The idea is to find solutions from which no single player

can deviate to individually improve its welfare. This central concept in game theory is called *Nash equilibrium*.

Definition 2.10 (Nash equilibrium)**.** A strategy vector $s \in S$ is said to be a *Nash equilibrium* if for all players i and each alternate strategy $s'_i \in S_i$, we have that:

$$u_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i})$$

In other words, no player i has any incentive to change its strategy from s_i to s'_i in order to unilaterally improve its payoff.

It is easy to see that a dominant strategy solution is a Nash equilibrium. Moreover, if there is a solution that switching to it always strictly improves the outcome, then this solution is the unique Nash equilibrium of the instance. Note, however, that some pure-strategy games may have more than one Nash equilibrium strategy or may not have a Nash equilibrium at all[1].

Finding such equilibria, even if their existence can be proved like in the case of mixed strategies, is hard. The evidence of the intractability of this problem was first shown by Chen and Deng [CD06]:

Theorem 2.11 ([CD06])**.** *The problem of finding a Nash equilibrium is PPAD-complete even for two-player games.*

Being in the PPAD class (*Polynomial Parity Arguments on Directed graphs*)[2] is a strong evidence that no general efficient solution exists for the problem of finding Nash equilibria in arbitrary games.

Another interesting problem studied in game theory is related to the social impact of the selfish choices. Exactly how inefficient is a system where each selfish player makes its own decisions if compared to an idealized situation, where all players would collaborate selflessly, pursuing a common goal of minimizing some cost objective? In order to measure this inefficiency, Koutsoupias and Papadimitriou [KP99] introduced the concept of *price of anarchy*, which is defined as the ratio between the worst objective function value of an equilibrium and the one of an optimal outcome.

[1] Nash [Nas51] proved that every mixed-strategy game with a finite set of players and strategies has a Nash equilibrium.

[2] Please refer to Papadimitriou [Pap94] for a formal definition of the PPAD class.

Definition 2.12 (Price of anarchy)**.** Given S the set of possible strategies, $E \subseteq S$ the set of all Nash equilibria and $C : S \mapsto \mathbb{R}$ a cost function that measures the inefficiency of the system, the *price of anarchy* (PoA) is defined by:

$$\text{PoA} = \frac{\max_{s \in E} C(s)}{\min_{s \in S} C(s)}$$

Similarly, game theorists also study the *price of stability* (PoS) of games, that measures the ratio between the best objective function value of one of its equilibrium and that of an optimal outcome.

Definition 2.13 (Price of stability)**.** Given S the set of possible strategies, $E \subseteq S$ the set of all Nash equilibria and $C : S \mapsto \mathbb{R}$ a cost function that measures the inefficiency of the system, the *price of stability* (PoS) is defined by:

$$\text{PoS} = \frac{\min_{s \in E} C(s)}{\min_{s \in S} C(s)}$$

2.4 Concluding remarks

In this chapter, we presented a brief introduction to classical concepts of scheduling theory. These concepts will be used and extended in the remaining of the text to model the problem of collaboration on modern platforms. Some of the most important theoretical results on single-objective scheduling were presented and detailed.

We also presented concepts of multi-objective optimization and game theory. These concepts will be useful to our study about relations and trade-offs among participants sharing resources on parallel and distributed platforms.

CHAPTER 3

Multi-organization scheduling approximation algorithms

GRID computing systems allow unprecedented computational power by combining geographically dispersed computers into a single massively parallel distributed system. Users of such systems contribute with computational power (multi-core machines, clusters, etc.) and expect to be able to execute their own jobs more efficiently by sharing their own resources with others.

The heterogeneity of the available resources, the large number of available processors and cores, and different demands from users make the problem of scheduling in such parallel platforms really hard in practice. In order to fully exploit such systems, we need sophisticated scheduling algorithms that encourage the users to share their resources and, at the same time, that respect each user's own interests.

In this chapter we present our efforts on how to deal with such issues. Our main contribution is the extension and analysis of the problem for the case in which sequential jobs are submitted by *selfish organizations* that can handle different local objectives (namely, makespan and average completion times).

We introduce new restrictions to the schedule that take into account this concept of *selfish organizations*, i.e., organizations that refuse to cooperate if their objectives could be improved just by executing earlier one of their jobs in one of their own machines.

A fairness study is conducted using two different fairness metrics. The objective

of this study is to evaluate if the results obtained by all organizations equally improve every organization's local objective by constructing schedules that are considered fair according to these metrics. Unfairness in the results obtained by each organization could be considered a reason to make an organization stop the cooperation with the others.

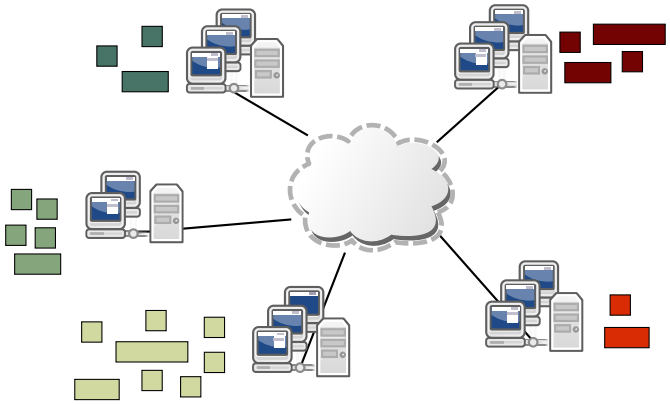
The remaining of this chapter is organized as follows. The formal description of the problem and the notations used in this paper are described in Section 3.1. Section 3.2 shows that any algorithm respecting our new selfishness restrictions cannot achieve approximation ratios better than 2 and that the problem is computationally hard wherever the users are interested in minimizing their makespan or their average completion time. 2-approximation algorithms for solving the problem are presented in Section 3.3. The heuristics are analyzed using fairness metrics presented in Section 3.4. Simulation experiments, discussed in Section 3.5, show the good results obtained by our algorithms in average and how each organization perceives the fairness of the results obtained. Some concluding remarks are presented in Section 3.6.

3.1 The Multi-Organization Scheduling Problem

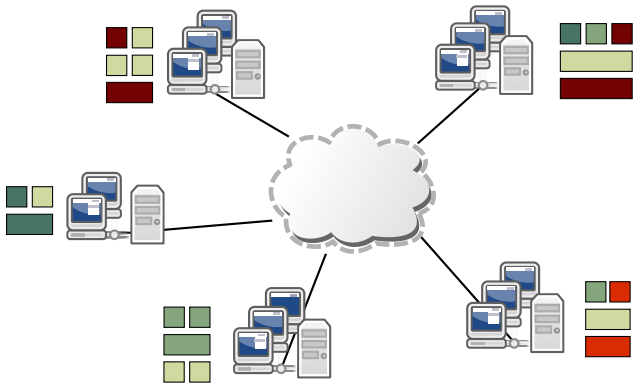
Grid computing platforms are typically organized as a federated system where users and computational resources, belonging to different administrative domains, share resources and exchange jobs with each other in order to simultaneously maximize the profits of the collectivity and their own interests.

We call each participant of this federation an *organization*. In terms of the governance of a grid computing platform, an organization has full control over the management of its own resources. The organization chooses how many (and which) resources are shared and used. Examples of such organizations are research laboratories, universities or company departments. Figure 3.1 depicts the described platform.

Organizations are free to join or leave the platform at any time if they feel that their expectations are not being fulfilled. This could happen, for instance, if the organization presumes that their jobs are not getting a fair share of the available



(a) Organizations and their jobs



(b) Scheduling made possible by cooperation

Figure 3.1: Illustration of the target platform, showing how the scheduling of the jobs belonging to these organizations can be improved if the computational resources are shared with each other.

resources or if the execution of its own jobs is being retarded because of the execution of jobs from other organizations.

On the one hand, it is crucial to determine schedules that optimize the allocation of the jobs for the whole platform in order to achieve good system performances. On the other hand, it is important to guarantee the performance perceived by each organization in order to provide incentive for all organizations to collaborate.

In order to achieve this, we study a scheduling model for the problem in which different organizations own a physical cluster of identical machines that are interconnected. All organizations intent to minimize the global makespan (i.e., the maximum completion time in any local schedule) while they individually try to minimize their own objectives, either the makespan or the average completion time of their own jobs.

Although each organization accepts to cooperate with others in order to minimize the global makespan, we assume that individually it behaves in a selfish way. An organization can refuse to cooperate if in the final schedule one of its migrated jobs could be executed earlier in one of the machines owned by the organization.

We start by first extending the notation for classical scheduling problems that were described in Section 2.1, in order to consider the presence of multiple organizations in the model. Formally, we define our target platform as a grid computing system with N different organizations interconnected by a middleware. Each organization $O^{(k)}$ $(1 \leq k \leq N)$ has $m^{(k)}$ identical machines available that can be used to run jobs submitted by users from any organization.

Each organization $O^{(k)}$ has $n^{(k)}$ jobs to execute, and the total number of jobs is denoted by $n = \sum_k n^{(k)}$. Each job $J_i^{(k)}$ $(1 \leq i \leq n^{(k)})$ will use one processor for exactly $p_i^{(k)}$ units of time[1]. The size of the largest job from organization $O^{(k)}$ is denoted as $p_{\max}^{(k)}$. No preemption is allowed, i.e., after its activation, a job runs until its completion at time $C_i^{(k)}$.

We denote the makespan of a particular organization $O^{(k)}$ by $C_{\max}^{(k)} = \max_{1 \leq i \leq n^{(k)}} (C_i^{(k)})$ and its sum of completion times as $\sum C_i^{(k)}$. The global makespan for the entire grid computing system is defined as $C_{\max} = \max_{1 \leq k \leq N} (C_{\max}^{(k)})$.

[1] All machines are identical, i.e., every job will be executed at the same speed independently of the chosen machine.

Local constraint

The *Multi-Organization Scheduling Problem* (MOSP), as first described in [PRT07], consists in minimizing the global makespan ($C_{\max}$) with an additional *local constraint*: at the end, no organization can have its makespan increased if compared with the makespan that the organization could have by scheduling the jobs in its own machines ($C_{\max}^{(k)\ local}$). More formally, we call **MOSP($C_{\max}$)** the following optimization problem:

$$\text{minimize } C_{\max} \text{ such that, for all } k\ (1 \leq k \leq N),\ C_{\max}^{(k)} \leq C_{\max}^{(k)\ local}$$

We also are interested in the study of the case where all organizations are locally interested in minimizing their average completion time while minimizing the global makespan. As in MOSP($C_{\max}$), each organization imposes that the sum of completion times of its jobs cannot be increased if compared with what the organization could have obtained using only its own machines ($\sum C_i^{(k)\ local}$). We denote this problem **MOSP($\sum C_i$)** and the goal of this optimization problem is to:

$$\text{minimize } C_{\max} \text{ such that, for all } k\ (1 \leq k \leq N),\ \sum C_i^{(k)} \leq \sum C_i^{(k)\ local}$$

Selfishness

In both MOSP($C_{\max}$) and MOSP($\sum C_i$), while the global schedule might be computed by a central entity, the organizations keep control on the way they execute the jobs in the end. This property means that, in theory, it is possible for organizations to cheat the devised global schedule by re-inserting their jobs earlier in the local schedules.

In order to prevent such behavior, we define a new restriction on the schedule, called *selfishness restriction*. The idea is that, in any schedule respecting this restriction, no single organization can improve its local schedule by cheating.

Given a fixed schedule, let $J_f^{(l)}$ be the first foreign job scheduled to be executed in $O^{(k)}$ (or the first idle time if $O^{(k)}$ has no foreign job) and $J_i^{(k)}$ any job belonging to $O^{(k)}$. Then, the *selfishness restriction* forbids any schedule where $C_f^{(l)} - p_f^{(l)} < C_i^{(k)} - p_i^{(k)}$. In other words, $O^{(k)}$ refuses to cooperate if one of its jobs could be executed earlier in one of $O^{(k)}$ machines even if this leads to a larger global makespan.

Related work

From the classical scheduling theory, the problem of scheduling parallel jobs is related to the Strip packing [BCR80]. It corresponds to pack a set of rectangles (without rotations and overlaps) into a strip of machines in order to minimize the height used. Then, this problem was extended to the case where the rectangles were packed into a finite number of strips [YHZ09; Zhu06]. More recently, an asymptotic $(1+\epsilon)$-approximation AFPTAS with additive constant $O(1)$ and with running-time polynomial in n and in $1/\epsilon$ was presented in [Bou+10].

Schwiegelshohn, Tchernykh, and Yahyapour [STY08] studied a very similar problem, where the jobs can be scheduled in non-contiguous processors. Their algorithm is a 3-approximation for the maximum completion time (makespan) if all jobs are known in advance, and a 5-approximation for the makespan on the on-line, non-clairvoyant case. A comprehensive study concerning the applicability of their techniques were presented by Ramírez-Alcaraz et al. [Ram+11].

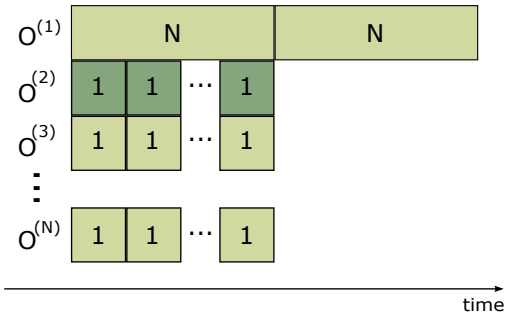
The Multi-Organization Scheduling Problem (MOSP) was introduced by Pascual et al. [PRT07; PRT09] and studies how to efficiently schedule parallel jobs in new computing platforms, while respecting users' own selfish objectives. A preliminary analysis of the scheduling problem on homogeneous clusters was presented with the target of minimizing the makespan, resulting in a centralized 3-approximation algorithm that is proven to be tight [Dut+11].

The notion of cooperation between different organizations and the study of the impact of users' selfish objectives are directly related to Game Theory. The relations between MOSP and game theory will be studied in Chapter 5.

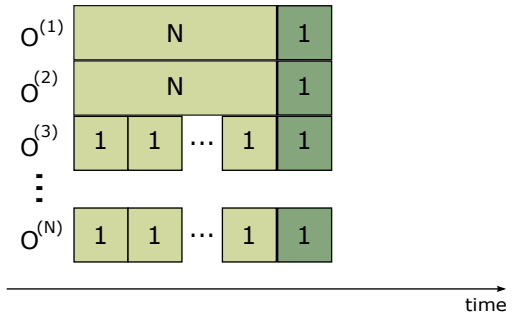
3.2 Complexity analysis

Lower bounds

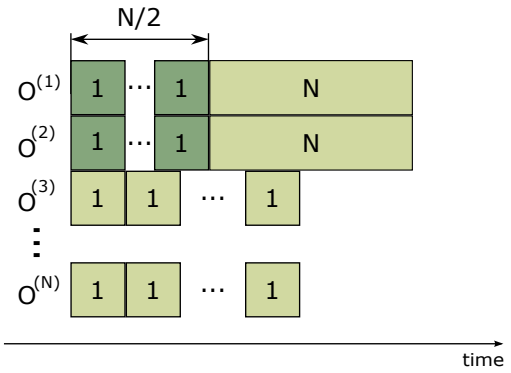
The lower bound of the problem was first shown by Pascual et al. [PRT07]. They showed with an instance composed of two organizations and two machines per organization that every algorithm that solves MOSP (for rigid, parallel jobs and $C_{\max}$ as local objectives) has at least a $\frac{3}{2}$-approximation ratio when compared to the



(a) Initial instance — $C_{\max} = 2N$



(b) Global optimum without constraints — $C_{\max} = N + 1$



(c) Optimum with MOSP constraints — $C_{\max} = \frac{3N}{2}$

Figure 3.2: Ratio between global optimum makespan and the optimum makespan that can be obtained for both MOSP($C_{\max}$) and MOSP($\sum C_i$). Jobs owned by organization $O^{(2)}$ are highlighted.

optimal makespan that could be obtained *without the local constraints*. We show that the same bound applies asymptotically even with a larger number of organizations.

Take the instance depicted in Figure 3.2a. $O^{(1)}$ initially has two jobs of size N and all the others initially have N jobs of size 1. All organizations contribute with only 1 machine each. The optimal makespan for this instance is $N + 1$ (Figure 3.2b), nevertheless it delays jobs from $O^{(2)}$ and, as consequence, does not respect MOSP's local constraints. The best possible makespan that respects the local constraints (whenever the local objective is the makespan or the average completion time) is $\frac{3N}{2}$, as shown in Figure 3.2c.

Selfishness and lower bounds

Although all organizations will likely cooperate with each other to achieve the best global makespan possible, their selfish behavior will certainly impact the quality of the best attainable global makespan.

We study here the impact of new selfishness restrictions on the quality of the achievable schedules. We show that these restrictions impact MOSP($C_{\max}$) and MOSP($\sum C_i$) as compared with unrestricted schedules and, moreover, that MOSP($C_{\max}$) with selfishness restrictions suffers from limited performances compared to MOSP($C_{\max}$) with local constraints.

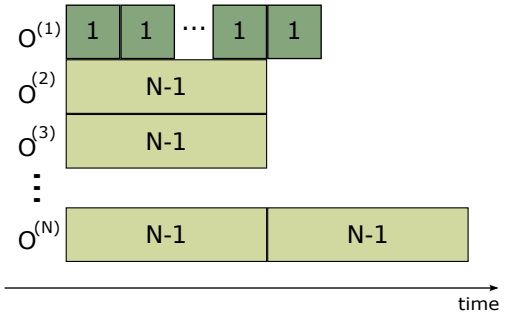
Proposition 3.1. *No approximation algorithm for both* MOSP($C_{\max}$) *and* MOSP($\sum C_i$) *has ratio asymptotically better than to* 2 *regarding the optimal makespan* without constraints *if all organizations behave selfishly.*

Proof. We prove this result by using the example described in Figure 3.2. It is clear from Figure 3.2b that an optimal solution for a schedule without local constraints can be achieved in $N+1$. However, with added selfishness restrictions, Figure 3.2a (with a makespan of $2N$) represents the only possible valid schedule. We can, therefore, conclude that local constraints combined with selfishness restrictions imply that no algorithm can provide an approximation ratio asymptotically better than 2 when compared with the problem without constraints. □

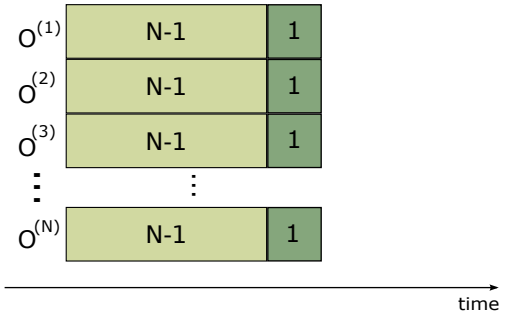
Proposition 3.1 gives a ratio regarding the optimal makespan without the local constraints imposed by MOSP. We can show that the same approximation ratio of 2 also applies for MOSP($C_{\max}$) regarding the optimal makespan even if MOSP constraints are respected.

Proposition 3.2. *Any approximation algorithm for* MOSP($C_{\max}$) *has a ratio greater than or equal to* $2-\frac{2}{N}$ *regarding the optimal makespan* with local constraints *if all organizations behave selfishly.*

Proof. Take the instance depicted in Figure 3.3a. $O^{(1)}$ initially has N jobs of size 1, $O^{(N)}$ has two jobs of size $N-1$, and the remaining organizations have each one a job of size $N-1$. The optimal solution that respects MOSP local constraints is given in



(a) Initial instance — $C_{\max} = 2N - 2$



(b) Global optimum with MOSP constraints — $C_{\max} = N$

Figure 3.3: Ratio between global optimum makespan with MOSP constraints and the makespan that can be obtained by MOSP($C_{\max}$) with selfish organizations.

Figure 3.3b and have $C_{\max}$ equal to N. Nevertheless, the best solution that respects the selfishness restrictions is the initial instance with a $C_{\max}$ equal to $2N - 2$. So, the ratio of the optimal solution with the selfishness restrictions to the optimal solution with MOSP constraints is $2 - \frac{2}{N}$. □

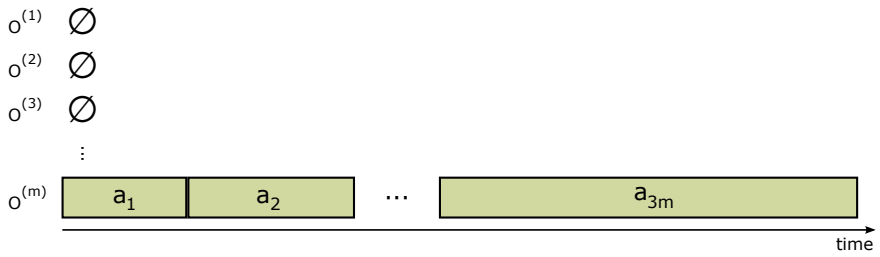
Computational complexity

This section studies how hard it is to find optimal solutions for the MOSP problem. We consider the decision version of the MOSP defined as follows:

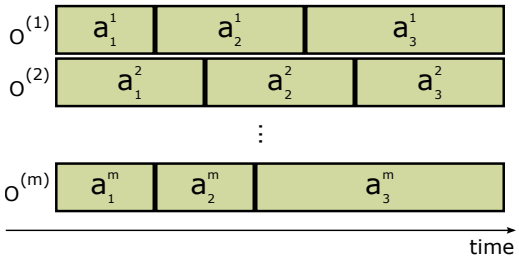
Instance: a set of N organizations (for $1 \leq k \leq N$, organization $O^{(k)}$ has $n^{(k)}$ jobs, $m^{(k)}$ identical machines, and makespan as the local objective) and an integer ℓ.

Question: Does there exist a schedule with a makespan less than ℓ that respects the local constraints?

Theorem 3.3. MOSP($C_{\max}$) *and* MOSP($\sum C_i$) *are strongly NP-complete.*



(a) Initial instance



(b) Optimum

Figure 3.4: Direct reduction of MOSP from 3-PARTITION.

A reduction from the classical 3-PARTITION problem [GJ79] (see definition below) to MOSP is straightforward and it is schematized in Figure 3.4. This reduction, however, represents a degenerate case of the MOSP problem. The instance presents a configuration where initially one organization is very loaded and some organizations do not contribute with any work. We believe that this configuration misrepresents the concept of cooperation introduced by the MOSP problem.

We present in this section a slightly more complex proof for the complexity of MOSP($C_{\max}$) and MOSP($\sum C_i$) with an instance where any organization could potentially have its makespan improved by cooperating with others. We show that the problem remains hard even for instances limited to two jobs per organization.

Theorem 3.4. MOSP($C_{\max}$) *and* MOSP($\sum C_i$) *are strongly NP-complete even if each organization has exactly two jobs per organization.*

Proof. It is straightforward to see that MOSP($C_{\max}$) $\in NP$ and MOSP($\sum C_i$) $\in NP$. Our proof is based on a reduction from the well-known 3-PARTITION problem [GJ79]:

Instance: a bound $B \in Z^+$ and a finite set A of $3m$ integers $\{a_1, \ldots, a_{3m}\}$, such that every element of A is strictly between $B/4$ and $B/2$ and such that $\sum_{i=1}^{3m} a_i = mB$.

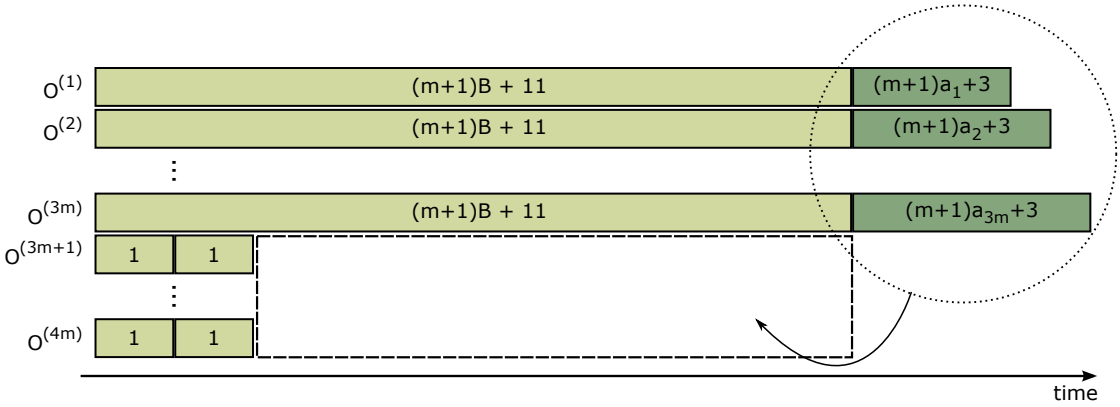


Figure 3.5: Reduction of MOSP($C_{\max}$) and MOSP($\sum C_i$) from 3-PARTITION.

Question: can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that, for all $1 \leq i \leq m$, $\sum_{a \in A_i} a = B$ and A_i is composed of exactly three elements?

Given an instance of 3-PARTITION, we construct an instance of MOSP where, for $1 \leq k \leq 3m$, organization $O^{(k)}$ initially has two jobs $J_1^{(k)}$ and $J_2^{(k)}$ with $p_1^{(k)} = (m+1)B + 11$ and $p_2^{(k)} = (m+1)a_k + 3$, and all other organizations have two jobs with processing time equal to 1. We then set ℓ to be equal to $(m+1)B + 11$. Figure 3.5 depicts the described instance. This construction is performed in polynomial time. Now, we prove that A can be split into m disjoint subsets $A_1, \ldots, A_m$, each one summing up to B, if and only if this instance of MOSP has a solution with $C_{\max} \leq (m+1)B + 11$.

Assume that $A = \{a_1, \ldots, a_{3m}\}$ can be partitioned into m disjoint subsets $A_1, \ldots, A_m$, each one summing up to B. In this case, we can build an optimal schedule for the instance as follows:

- for $1 \leq k \leq 3m$, $J_1^{(k)}$ is scheduled on machine k;
- for $3m + 1 \leq k \leq 4m$, $J_1^{(k)}$ and $J_2^{(k)}$ are scheduled on machine k;
- for $1 \leq i \leq m$, let $A_i = \{a_{i_1}, a_{i_2}, a_{i_3}\} \subseteq A$. The jobs $J_2^{(a_{i_1})}$, $J_2^{(a_{i_2})}$ and $J_2^{(a_{i_3})}$ are scheduled on machine $3m + i$.

This construction provides a schedule where the global $C_{\max}$ is the optimal one. Also, it is easy to see that the local constraints for MOSP($C_{\max}$) and MOSP($\sum C_i$) are respected: for MOSP($C_{\max}$), it is sufficient to note that no organization have its makespan increased; for MOSP($\sum C_i$), note that each job either remains in its

original organization and schedule (lighter colored jobs in Figure 3.5) or is migrated to a different organization at an earlier time (darker colored jobs in Figure 3.5), strictly decreasing the average completion time.

Conversely, assume that MOSP has a solution with $C_{\max} \leq (m+1)B+11$. The total work (W) of the jobs that must be executed is $W = 3m \cdot ((m+1)B+11) + \sum_{i=1}^{3m}((m+1)B+a_i) + m \cdot 2 = 4m \cdot ((m+1)B+11)$. Since we have exactly $4m$ organizations, the solution must be the optimal solution and there are no idle times in the scheduling. Moreover, $3m$ machines must execute only one job of size $(m+1)B+11$. W.l.o.g, we can consider that for $3m+1 \leq k \leq 4m$, machine k performs jobs of size less than $(m+1)B+11$.

To prove our proposition, we first show two lemmas that characterize the structure of the solution for the MOSP problem:

Lemma 3.5. *For all $3m+1 \leq k \leq 4m$, exactly three jobs of size not equal to 1 must be scheduled on machine k if $C_{\max}^{(k)} = (m+1)B+11$.*

Proof. We proof this lemma by contradiction. First, note that for all integers from the 3-PARTITION instance, it holds that $a_i > B/4$.

Assume that one machine has less than three jobs of size not equal to 1. $3m$ jobs of this kind must be assigned to the last m organizations (since the first $3m$ organizations execute each one a large job of size $(m+1)B+11$). If one machine has less than three jobs, then we must have at least one other machine with four or more jobs of this kind assigned to it. The makespan on this other machine must be of at least $4 \cdot ((m+1)a_i + 3) > 4 \cdot ((m+1)B/4 + 3) = (m+1)B+12$. This contradicts the fact that $C_{\max}^{(k)} = (m+1)B+11$ and shows that exactly three jobs of size not equal to 1 must be scheduled on each machine. □

Lemma 3.6. *For all $3m+1 \leq k \leq 4m$, exactly two jobs of size 1 are scheduled on each machine if $C_{\max}^{(k)} = (m+1)B+11$.*

Proof. Since $C_{\max}^{(k)} = (m+1)B+11$, each large job of size $(m+1)B+11$ must remain in its original organization. Consequently, the $2m$ jobs of size 1 must be scheduled in one of the m organizations not having a large job assigned to it. However, each of these organizations has $C_{\max}^{(k)\ local} = 2$, which means that any migration that results in a

schedule with one organization not having two jobs of size 1 will result in a schedule that does not respect both MOSP($C_{\max}$) and MOSP($\sum C_i$) local constraints. □

From the solution for the MOSP problem, we construct m disjoint subsets $A_1, A_2, \ldots, A_m$ of A as follows: for all $1 \leq i \leq m$, a_j is in A_i if the job with size $(m+1)a_j + 3$ is scheduled on machine $3m+i$. Note that all elements of A belong to one and only one set in $\{A_1, \ldots, A_m\}$. We prove that A is a partition with the desired properties.

We focus on organization $O^{(3m+i)}$, where A_i was assigned. Lemmas 3.5 and 3.6 show how the jobs will be assigned to this organization. First, Lemma 3.5 shows that A_i is composed of exactly three jobs that will be executed on $O^{(3m+i)}$. Second, Lemma 3.6 shows that exactly two jobs of size 1 will be scheduled at the beginning of the schedule.

Since all organizations must have a local $C_{\max}$ exactly equal to $(m+1)B+11$, we have that $C_{\max}^{(3m+i)} = (m+1)B + 11 = 1 + 1 + \sum_{a_j \in A_i}((m+1)a_j + 3)$, which implies that:

$$
\begin{aligned}
2 + \sum_{a_j \in A_i}((m+1)a_j + 3) &= (m+1)B + 11 \\
\Rightarrow \sum_{a_j \in A_i}(m+1)a_j &= (m+1)B \\
\Rightarrow \sum_{a_j \in A_i} a_j &= B
\end{aligned}
$$

Thus, we can deduce that A_i is composed of exactly three elements and $\sum_{a_j \in A_i} a_j = B$. In other words, $\{A_i, \ldots, A_m\}$ is a solution for the 3-PARTITION problem. □

3.3 Algorithms

In this section, we present four different heuristics to solve MOSP($C_{\max}$) and MOSP($\sum C_i$). All algorithms present the additional property of respecting selfishness restrictions.

Iterative load balancing algorithm

The Iterative Load Balancing Algorithm (ILBA) [PRT09] is a heuristic that redistributes the load from the most loaded organizations.

The idea is to incrementally rebalance the load without delaying any job. First the less loaded organizations are rebalanced. Then, one-by-one, each organization has its load rebalanced.

The heuristic works as follows. First, each organization schedules its own jobs locally and the organizations are enumerated by non-decreasing makespans, i.e. $C_{\max}^{(1)} \leq C_{\max}^{(2)} \leq \ldots \leq C_{\max}^{(N)}$. For $k = 2$ until N, jobs from $O^{(k)}$ are rescheduled sequentially, and assigned to the less loaded of organizations $O^{(1)} \ldots O^{(k)}$.

Each job is rescheduled by ILBA either earlier or at the same time that the job was scheduled before the migration. In other words, no job is delayed by ILBA, which guarantees that the local constraint is respected for MOSP($C_{\max}$) and MOSP($\sum C_i$).

The ILBA algorithm has the same time complexity as any list scheduling algorithm for the classical problem of minimizing the makespan. Its time complexity is $O(n \log n)$. We recall that n is the total number of jobs.

LPT-LPT and SPT-LPT heuristics

We developed and evaluated (see Section 3.5) two new heuristics based on the classical LPT (Longest Processing Time First [Gra69]) and SPT (Smallest Processing Time First [BCS74]) algorithms for solving MOSP($C_{\max}$) and MOSP($\sum C_i$), respectively. Both heuristics work in two phases. During the first phase, all organizations minimize their own local objectives. Each organization starts applying LPT for its own jobs if the organization is interested in minimizing its own makespan, or starts applying SPT if the organization is interested in its own average completion time.

The second phase is when all organizations cooperatively minimize the makespan of the entire grid computing system without worsening any local objective. This phase works as follows: each time an organization becomes idle, i.e., it finishes the execution of all jobs assigned to it, the longest job that does not have started yet is migrated and executed by the idle organization. This greedy algorithm works like a global LPT, always choosing the longest job yet to be executed among jobs from all organizations.

The first and second phases correspond to a list scheduling algorithm, so the total running time of both algorithms is $O(n \log n)$.

Less Helped First heuristic

Like the previous algorithms, Less Helped First works in two phases. First, each organization $O^{(k)}$ is classified according to the total work originally submitted by a user in $O^{(k)}$ that was actually executed by a different organization. Locally, the order of the jobs is arbitrary (it could be the submission order, for instance).

Second, each time a processor becomes idle, the algorithm will prioritize the organization that received *less help* from the others, i.e. that had less work executed by other organizations. The algorithm will migrate any job from the *less helped* organization that has not started yet to be executed by the idle processor.

The running time of the first phase is $O(n \log n)$. During the second phase, after each job migration, the classification of the organizations must be updated. This update can be done in $O(\log n)$ and there are at most n migrations (one per job). Since the number of organizations is negligible compared to the number of jobs, the total running time of the algorithm is $O(n \log n)$.

Analysis

ILBA, LPT-LPT, SPT-LPT, and Less Helped First do not delay any of the jobs when compared to the initial local schedule. During the rebalancing phase, all jobs either remain in their original organization or are migrated to an organization that became idle at a preceding time. The implications are:

- the *selfishness* restriction is respected — if a job is migrated, it will start before any foreign jobs that may have been assigned to its original organization;
- if organizations' local objective is to minimize the makespan, migrating a job to a previous moment in time will decrease the job's completion time and, as consequence, it will not increase the initial makespan of the organization;
- if organizations' local objective is to minimize the average completion time, migrating a job from the initial organization to another that became idle at a

previous moment in time will not increase the completion time of any job. This means that the $\sum C_i$ of the jobs from the initial organization is either decreased or remains constant;

- the rebalancing phase of all four algorithms works as the list scheduling algorithms. Graham's classical approximation ratio $2 - \frac{1}{N}$ (see Theorem 2.2) of list scheduling algorithms [Gra69] holds for all of them.

Proposition 3.1 (Section 3.2) states that no algorithm respecting selfishness restrictions can achieve an approximation ratio for MOSP($C_{\max}$) better than 2 regarding the optimal makespan without constraints. Since all our algorithms reach an approximation ratio of 2, no further enhancements are possible without removing selfishness restrictions.

3.4 Fairness

Another form of encouragement of cooperation between the organizations is providing algorithms that equally improve their local objectives. Organizations that are not invidious of other organizations are more likely to share their resources to the platform. Selfish organizations could potentially become invidious if the results obtained when solving the MOSP problem improve the local objectives of the organizations in an unfair manner.

Some fairness metrics can be found on the literature — Variance, Coefficient of Variance $\left(\frac{\text{Variance}}{\text{Mean}}\right)$, Min-Max ratio, etc. The Jain Index [JCH84] is nowadays one of the most popular metrics used to measure fairness [LT07]. We also studied the *Stretch* as an index of fairness, as we will show in the next sections.

In this section we study the fairness of the results produced by the algorithms presented in Section 3.3. Using the same workload generated for the experimental results shown in Section 3.5, we evaluate the fairness of these algorithms for two important metrics: Stretch and the Jain Index.

Stretch

Stretch is a performance metric that was extensively studied in the scheduling literature. This performance metric can also be considered as a fairness metric. We adapted the notion of Stretch to the MOSP problem, to measure the Stretch of an entire organization. We define Stretch as the ratio of the local $C_{\max}$ obtained by the organization to the $C_{\max}$ that the organization could have obtained if it had all the machines available only for its jobs (denoted by $C_{\max}^{(k)\ alone}$). Formally, the Stretch is calculated as follows:

$$\text{Stretch} = \max_k \frac{C_{\max}^{(k)}}{C_{\max}^{(k)\ alone}}$$

Since $C_{\max}^{(k)\ alone}$ is bounded by the theoretical lower bound $\frac{\sum_i p_i^{(k)}}{N} = \frac{C_{\max}^{(k)\ local}}{N}$ and since MOSP($C_{\max}$) constraint guarantees that $C_{\max}^{(k)} \leq C_{\max}^{(k)\ local}$, we have:

$$\text{Stretch} = \max_k \frac{C_{\max}^{(k)}}{C_{\max}^{(k)\ alone}} \leq \frac{C_{\max}^{(k)\ local}}{\frac{C_{\max}^{(k)\ local}}{N}} \leq N$$

Note that this upper bound for the Stretch metric is tight. Consider the instance with N organizations, each one having N jobs with processing times equal to 1. $C_{\max}^{(k)\ alone}$ is equal to one for all organization $O^{(k)}$, and $C_{\max}^{(k)\ local} = N$. There is only one scheduling that respects MOSP($C_{\max}$) local constraint, the one that schedules all jobs in their original organizations.

Jain index

Originally, the Jain Index was introduced for the study of resource sharing and allocation problems. The Jain Index of a set of n users receiving an "allocation" x_i [JCH84] is defined as:

$$f(x) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}, x_i \geq 0 \tag{3.1}$$

If the values of x_i follow a certain random distribution, the index is the ratio of the first moment (the mean) to the second moment of the distribution.

The Jain Index has the following useful properties:

- It is independent of the size of the population;
- It is independent of the scale (or metric unit) used;
- The values of the index are bounded between $1/n$ and 1;
- It is continuous.

We evaluated the fairness of the heuristics presented in Section 3.3 regarding the improvements obtained on the local $C_{\max}$ by each organization. We define this *improvement* as the ratio of the obtained $C_{\max}$ to the theoretical lower bound of each organization over all available machines of the system. The lower bound of each organization is calculated by:

$$LB^{(k)} = \max\left(\sum_i \frac{p_i^{(k)}}{N}, p_{\max}^{(k)}\right)$$

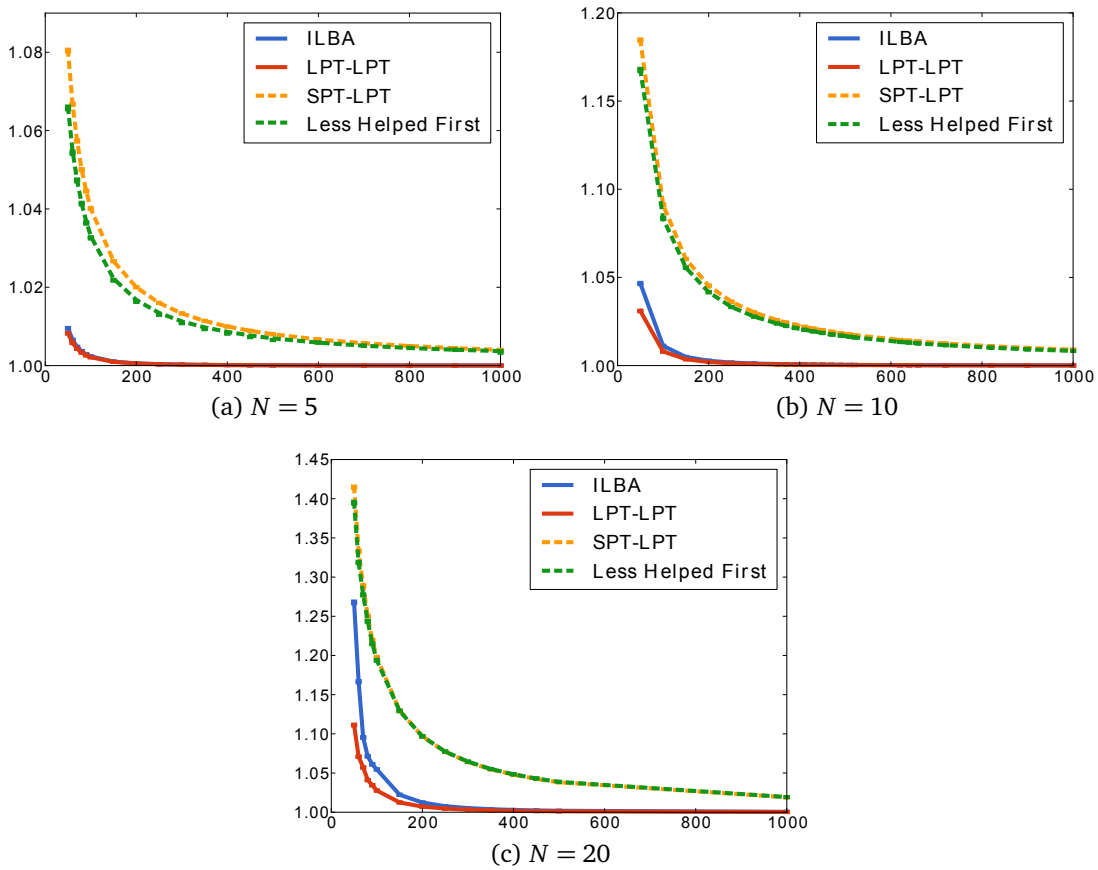
The improvement for organization $O^{(k)}$ (the x_k of Equation 3.1) is then defined as: $\frac{C_{\max}^{(k)}}{LB^{(k)}}$. So, the final Jain Index calculated to measure the fairness of the improvements can be stated as follows:

$$\text{Jain Index} = \frac{\left(\sum_{k=1}^{N} \frac{C_{\max}^{(k)}}{LB^{(k)}}\right)^2}{N\sum_{k=1}^{N}\left(\frac{C_{\max}^{(k)}}{LB^{(k)}}\right)^2}$$

3.5 Experiments

We conducted an extensive series of simulations comparing ILBA, LPT-LPT, SPT-LPT, and Less Helped First under various experimental settings. The workload was randomly generated with parameters matching the typical environment found in academic grid computing systems [PRT09].

We generated random workloads with $N = 5, 10,$ and 20 organizations. For each number of organizations, we have generated instances with different number of total jobs (from 100 to 1,000), with sizes chosen uniformly at random from 1 to 1,000. For each combination of these parameters, we generated 20,000 random instances. In our tests, the number of initial jobs in each organization follows a Zipf distribution



(a) $N = 5$ (b) $N = 10$

(c) $N = 20$

Figure 3.6: Mean and confidence interval of the measured $C_{\max}^{(k)}$ obtained by ILBA, LPT-LPT, SPT-LPT, and Less Helped First.

with exponent equal to 1.4267, which best models virtual organizations in real-world grid computing systems [Ios+06]. All results are presented with confidence level of 95%.

Global $C_{\max}$ analysis

We are interested in the improvement of the global $C_{\max}$ provided by the different algorithms. The results are evaluated with comparison to the $C_{\max}$ obtained by the algorithms with the well-known theoretical lower bound for the scheduling problem without constraints $LB = \max(\sum_{i,k} \frac{p_i^{(k)}}{m^{(k)}}, p_{\max})$.

Our main conclusion is that, despite the fact that the selfishness restrictions are

respected by all heuristics, ILBA and LPT-LPT obtained near optimal results for most cases. This is not unusual, since it follows the patterns of experimental behavior of standard list scheduling algorithms, in which it is easy to obtain a near optimal schedule when the number of jobs grows large. SPT-LPT and Less Helped First produce worse results due to the effect of the local order of the jobs. Figure 3.6 shows the average global $C_{\max}$ obtained by the heuristics for instances with different number of organizations and total number of jobs.

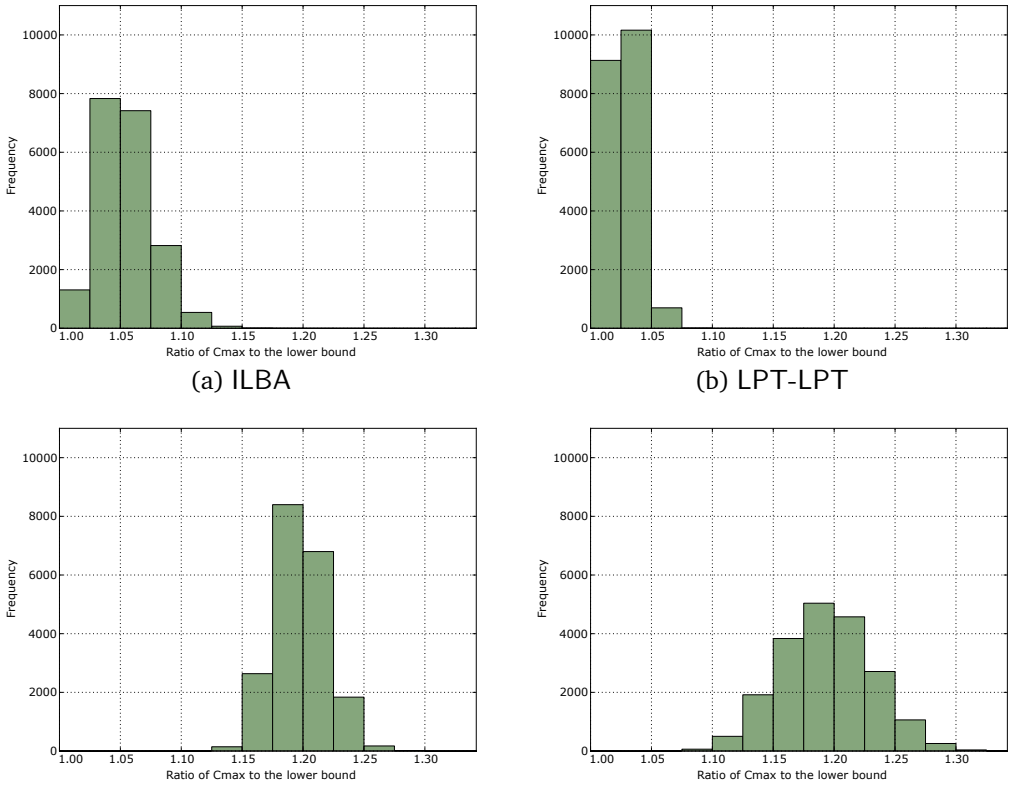
However, in some particular cases, in which the number of jobs is not much larger than the number of machines available, the experiments yield more interesting results. Figure 3.7 shows the histogram of a representative instance of such a particular case. The histograms show the frequency of the ratio $C_{\max}$ obtained to the lower bound over 20,000 different instances with 20 organizations and 100 jobs for ILBA, LPT-LPT, SPT-LPT, and Less Helped First. Similar results have been obtained for many different sets of parameters. LPT-LPT outperforms ILBA (and SPT-LPT) for most instances and its average ratio to the lower bound is less than 1.3. Less Helped First is the heuristic that produces the worst results.

Local $C_{\max}$ analysis

We are also interested in the improvements obtained by each organization that our four algorithms provide. In order to have a global view of the total improvement obtained, we have evaluated the sum of the local makespans obtained by all organizations: $\sum_k C_{\max}^{(k)}$. The sum of the local makespans is important because it shows the average results that each selfish organization obtains.

The experiments show that ILBA produces results with a lower makespan in average. LPT-LPT, SPT-LPT, and Less Helped First produce similar results in average. Recall from Section 3.3 that ILBA rebalances the load of all organizations, from the less loaded to the more loaded organization. This mechanism minimizes the makespan of the organization individually in a more aggressive way and, therefore, produces lower average local makespans. Figure 3.8 shows the average local $C_{\max}$ obtained by all heuristics.

It is interesting to note that the gap between the average result obtained by ILBA and the other heuristics increases as the number of organizations and the total



(a) ILBA (b) LPT-LPT

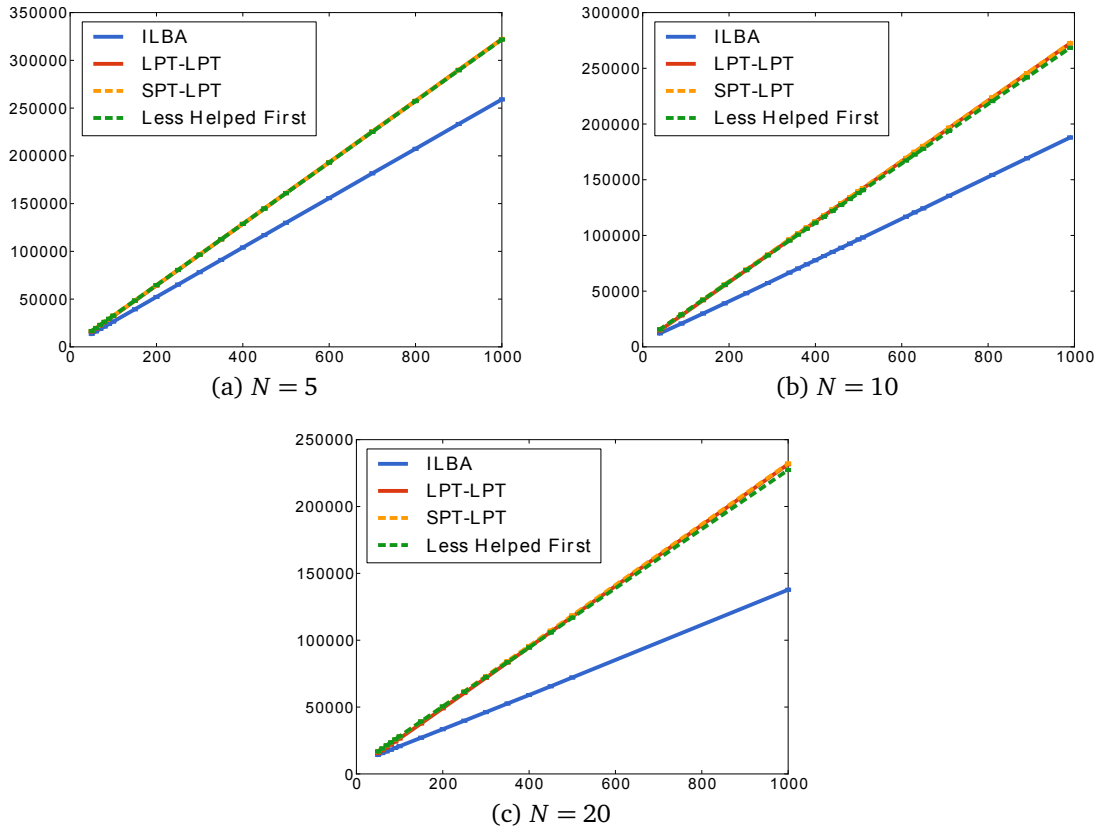
(c) SPT-LPT (d) Less Helped First

Figure 3.7: Frequency of results obtained by ILBA, LPT-LPT, SPT-LPT, and Less Helped First for $N = 20$ and $n = 100$.

number of jobs increase.

Fairness analysis

In the two previous sections we have analyzed the performance obtained by the four algorithms presented in Section 3.3 for the global makespan obtained and for the local makespan obtained by each organization. In this section we use the same testbed to study how each organization perceives the fairness of these results. To study the fairness, we use the Stretch and Jain Index metrics presented in Section 3.4.



(a) $N = 5$ (b) $N = 10$

(c) $N = 20$

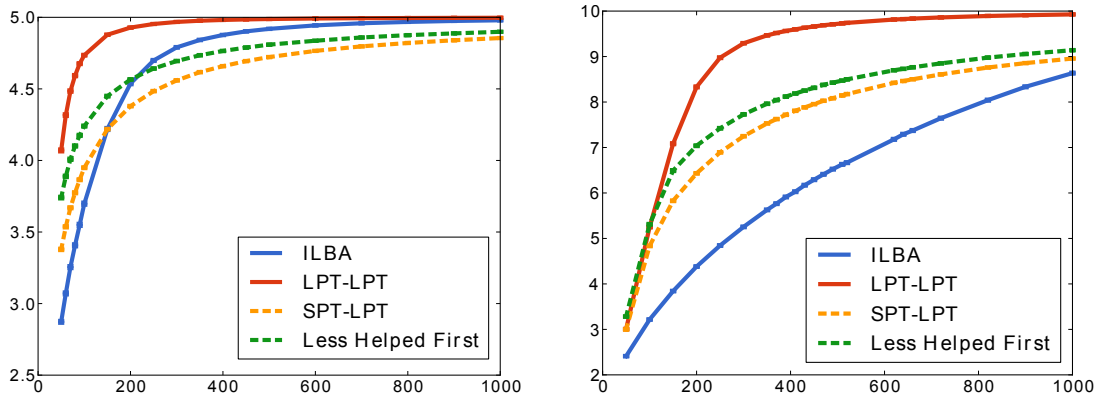
Figure 3.8: Mean and confidence interval of the measured $\sum_k C_{\max}^{(k)}$ obtained by ILBA, LPT-LPT, SPT-LPT, and Less Helped First.

Stretch

We have measured the worst — i.e. maximum — Stretch (as defined in Section 3.4) obtained during the simulations. Figure 3.9 shows how the Stretch varies according to the total number of jobs of the system.

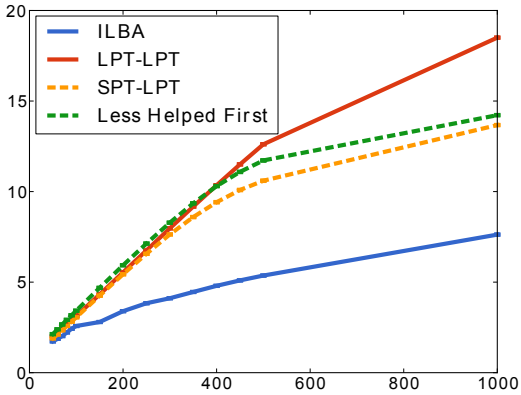
The results show that the Stretch increases asymptotically to the upper-bound of the metric (i.e. the number of organizations) as the total number of jobs on the system increases.

The Stretch obtained with our algorithms is given by the less loaded organization. Our heuristics never migrate jobs from the organization that has the lower local $C_{\max}$. For this organization, the $C_{\max}^{(k)}$ will always be equal to the $C_{\max}^{(k)\,local}$ and the Stretch will be higher. The Zipf distribution used to randomly assign the jobs to the



(a) $N = 5$

(b) $N = 10$



(c) $N = 20$

Figure 3.9: Mean and confidence interval of the measured *Stretch* obtained by ILBA, LPT-LPT, SPT-LPT, and Less Helped First.

organizations distributes more jobs to the less loaded organization when the total number of organizations is smaller. For this reason, the average Stretch grows faster for $N = 5$.

When the number of jobs is small, the performance of ILBA degrades more slowly because ILBA rebalances first the less loaded organizations, while the others rebalance the organizations in a dynamic order.

LPT-LPT presented the worst values for Stretch. Bigger Stretch values indicate that organizations are more penalized by the jobs from other organizations. Recall from Section 3.5 that LPT-LPT produces better results for the global $C_{\max}$. To achieve these results, LPT-LPT must penalize some organizations, and Stretch measures

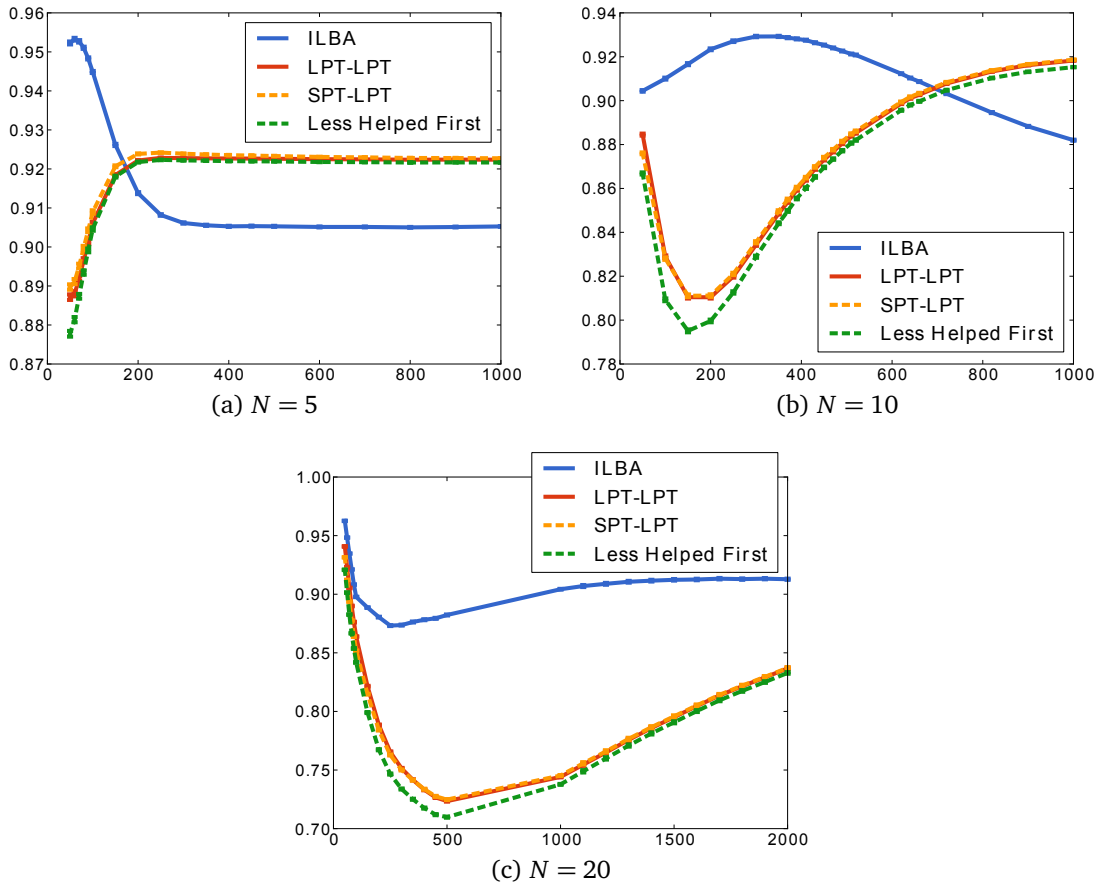


Figure 3.10: Mean and confidence interval of the measured *Jain Index* obtained by ILBA, LPT-LPT, SPT-LPT, and Less Helped First.

exactly this behavior.

Jain index

Jain Index and Stretch metrics measure two different forms of fairness. The former measures how each organization perceives the improvements on its own $C_{\max}$, while the latter shows how the $C_{\max}$ of each individual organization is affected by the jobs from other organizations.

When the number of organizations N is 5, our experiments showed that the four heuristics provide fair results, with Jain Index higher than 0.88. Recall that the Jain Index is bounded from $1/N$ (unfair) to 1 (perfect fairness).

Figure 3.10a indicates that the ILBA heuristics have two moments. In the first, when the number of jobs are smaller, ILBA heuristic produces more fair results than the other heuristics. Its Jain Index varies from 0.92 up to 0.95.

During the second moment, when the number of jobs is higher, ILBA produces the less fair results between all the four heuristics. LPT-LPT, SPT-LPT, and Less Helped First produce results with similar fairness indices, while ILBA fairness is clearly worse when compared to the others. Remark that the index is almost constant when $N = 5$ and there are more than 300 jobs on the platform.

The experiments with $N = 10$ show that ILBA produces results with fairness similar to the results with $N = 5$. Their Jain Index is of about 0.90. When the number of jobs is over 700, LPT-LPT, SPT-LPT, and Less Helped First are fairer. Their Jain Index varies between 0.79 and 0.91.

For $N = 20$, the difference between the performance of ILBA and the others is higher. ILBA has Jain Index of about 0.90 while the fairness of the other three heuristics decreases from 0.94 to 0.70. The number of jobs of the simulations shown in Figure 3.10c was increased up to 2,000 to show that for $N = 20$ the results indicate that ILBA has also two moments, like for the cases where $N = 5$ and $N = 10$.

Even if ILBA provided fairer results than the others with less jobs on the system, all four algorithms improve the local $C_{\max}$ obtained by each organization in a fair way.

3.6 Concluding remarks

In this chapter, we have investigated the scheduling on multi-organization platforms. We presented the MOSP($C_{\max}$) problem from the literature and extended it to a new related problem MOSP($\sum C_i$) with another local objective. In each case we studied how to improve the global makespan while guaranteeing that no organization will worsen its own results.

We showed first that both versions MOSP($C_{\max}$) and MOSP($\sum C_i$) of the problem are strongly NP-hard. Furthermore, we introduced the concept of *selfishness* in these problems which corresponds to additional scheduling restrictions designed to reduce the incentive for the organizations to cheat locally and disrupt the global schedule. We proved that any algorithm respecting selfishness restrictions cannot achieve a

better approximation ratio than 2 for MOSP($C_{\max}$) regarding the optimal makespan without MOSP local constraints.

Three new scheduling algorithms were proposed, namely LPT-LPT, SPT-LPT, and Less Helped First in addition to ILBA from the literature. All these algorithms are list scheduling, and thus achieve a 2-approximation. We provided an in-depth analysis of these algorithms, showing that all of them respect the selfishness restrictions.

Finally, all these algorithms were implemented and analyzed through experimental simulations. The results show that when considering the global makespan, our new LPT-LPT outperforms ILBA, and that all algorithms exhibit near optimal global performances when the number of jobs becomes large.

Considering the local objectives, ILBA achieves better average local makespans. ILBA also proved to present more fair results according to the Stretch metric. All the four algorithms presented good fairness using the Jain Index, showing that the local makespan of all organizations are equally improved.

Chapter 4

Relaxed multi-organization scheduling algorithms

The results presented in the previous chapter show that if in the one hand it is possible for the scheduler to encourage collaboration by providing guarantees on the performance perceived by each organization, on the other hand it also shows that these encouragements have a cost on the global performance.

First, there is the impact of MOSP local constraints, that inflict a degradation on the best possible global makespan. The instance depicted on Figure 3.2 showed that any algorithm respecting MOSP local constrains has at least a $\frac{3}{2}$-approximation ratio when compared to the optimal makespan that could have been obtained by altruistic organizations (i.e., if all organizations are willing to help even if this results in disadvantages for their own jobs).

Then, there is the influence of the selfishness constraints on the global performance. Without MOSP local constraints, Proposition 3.1 showed that there is no approximation algorithm with a ratio asymptotically better than 2. Mixing both MOSP and selfishness constraints, we cannot have a better ratio than $2-\frac{2}{N}$, according to Proposition 3.2.

Therefore, there is a clear correlation between the guarantees that we can provide for each organization, in order to incentive collaboration, and the guarantee we can provide for the collectivity, in order to improve the global performance. In this chapter we study such correlations and investigate how much the collectivity

can improve the makespan over the entire platform when each organization is neither completely selfish nor completely altruistic, but actually tolerates a bounded degradation on the makespan of its own jobs.

The remaining of this chapter is organized as follows. Section 4.1 presents how we model this correlation between local and global performance by presenting the α-MOSP problem and its previously known results. In Section 4.2 we show how we improved the existing inapproximation bounds for α-MOSP by showing that, unlike previously thought, there is no polynomial time approximation algorithm for these bounds *even if* $P = NP$. We also present two families of instances whose Pareto optimal points corroborate the presented inapproximation bounds. Two new algorithms with guaranteed performance to solve the α-MOSP problem are presented in Section 4.3. The analysis shows that the first one achieves a $\frac{3}{2}$-approximation for the obtained global makespan, while it guarantees that no organization will have its makespan more than doubled. This solution is Pareto optimal according to [OII09]. The second one guarantees a $\frac{4}{3}$-approximation for the global makespan, while no organization has its makespan more than tripled. This solution belongs to the border of the inapproximability of the second family, and, thus, it is also Pareto optimal.

4.1 The α-Cooperative Multi-Organization Scheduling Problem

Recall from Section 3.1 that the Multi-Organization Scheduling Problem introduces *local constraints* to guarantee that all organizations will always have incentive to cooperate. In the final schedule, no organization will have its makespan increased when compared to the makespan that the organization could have by scheduling its jobs alone in its own set of processors ($C_{\max}^{(k)\ local}$).

By restraining the feasible schedules to the ones that respect the *local constraints*, the minimum attainable global $C_{\max}$ is restricted. There is a clear trade-off between how much each organization can improve its own local makespan and how much the global makespan can be improved.

This motivated Ooshita et al. [OII09] to study a relaxed version of the MOSP problem called the *α-Cooperative Multi-Organization Scheduling Problem* (abbreviated

by α-MOSP). Their study was conducted on the context of unrelated machines ($R \,||\, C_{\max}$), opposed to the previous works on MOSP that always studied the problem of scheduling on independent machines ($P \,||\, C_{\max}$).

In the α-MOSP problem, the *local constraints* imposed in the classical MOSP problem are relaxed. Each organization allows a degradation of its initial makespan by a factor $\alpha \geq 1$ that represents the *degree of cooperativeness*. Ooshita et al. study how much the global makespan can be improved if the makespan obtained by each organization k is bounded by $\alpha C_{\max}^{(k)\ local}$. When $\alpha > 1$, each organization is less selfish and is more likely to sacrifice its local objective in order to improve the global makespan. When $\alpha = 1$, the problem corresponds to the classical MOSP problem defined in [PRT07].

The MOSP optimization problem rewritten to model the degree α of cooperativeness can be stated as follows:

$$\text{minimize } C_{\max} \text{ such that, for all } k \ (1 \leq k \leq N),\ C_{\max}^{(k)} \leq \alpha \cdot C_{\max}^{(k)\ local}$$

Their first contribution was to show that for any degree of cooperativeness $\alpha \geq 1$, there exists an instance of α-MOSP where the ratio between the makespan that respects the degree of cooperativeness ($C_{\max}^{\alpha}$) and the optimal makespan without the local constraints ($C_{\max}^{*}$) satisfies the relation $\frac{C_{\max}^{\alpha}}{C_{\max}^{*}} \leq \max\left\{\sum_{l=1}^{N} \frac{1}{\alpha^l}; \frac{(\alpha+1)}{\alpha}\right\} - \epsilon$. This ratio shows that, when $\alpha = 1$, the lack of cooperation can make the makespan be $N - \epsilon$ times greater then it could have been with unlimited cooperation.

The authors developed an algorithm called ToMoS, which provides a way to transform a schedule with unrestrained cooperation into one with degree of cooperativeness α. The algorithm guarantees that the ratio between the $C_{\max}$ of the solution without the constraints and the solution constructed with the α-MOSP constraints is less then or equal to $\sum_{l=1}^{N} \frac{1}{\alpha^l} < \frac{\alpha}{(\alpha-1)}$.

They also have studied the complexity and inapproximation bounds for the α-MOSP problem. They have shown that the problem is strongly NP-Hard for any $\alpha > 1$. Using the arguments given in the reduction used on the proof, they have calculated the inapproximation bounds for the problem when $\alpha < 2$. Under the assumption of $\mathbf{P} \neq \mathbf{NP}$, there is no ρ-approximation algorithm for α-MOSP for any $\rho < \frac{(\alpha+1)}{\alpha}$. If $\alpha > 2$, the classical inapproximation ratio of $\frac{3}{2}$ of the ∞-MOSP problem — which is equivalent to the classical $R \,||\, C_{\max}$ problem [LST90] on their study — holds.

In the next sections we will present our study of algorithms with guaranteed approximation ratios when the degree of cooperativeness is fixed. We denote the approximation ratios by $(\alpha; \beta)$, meaning that if an algorithm respects a degree of cooperativeness of α, then the algorithm is a β-approximation for the global $C_{\max}$. We start by presenting some improved inapproximability analysis for the problem.

4.2 Inapproximability analysis

A natural question that arises when studying a multi-objective optimization problem — like the relaxed version of MOSP — is how to determine the Pareto set that characterizes the set of Pareto optimal solutions[1]. In this section we study how to characterize the Pareto set of MOSP. While this problem is hard (and still open), we provide some inapproximability results that should help to better characterize the Pareto set.

We provide in this section some families of instances that clearly show the trade-off between the objectives being optimized. We will show through these examples that if we bound the approximation ratio of one criteria, no scheduling algorithm will be able to improve the approximation ratio of the other objective. Those inapproximability results are stronger than in [OII09] because the shape of the inapproximation curve is broader (see Family 1 of instances on Section 4.2) and because we prove that there is no algorithm with better performance ratio (since the Pareto optimal solutions of these instances reach these ratios). Hence, while previous works show that no polynomial algorithms with better ratios exist unless $P = NP$, we show that there are no feasible solutions with better ratios at all, which eliminates the possibility of any further improvements even with the use of non-polynomial algorithms.

Principle

To better understand the inapproximation ratios presented in this section, we first start with a simple example. Let us consider an instance with $N = 3$ organizations

[1] Pareto optimality is a concept originally used in economics and now widely utilized to indicate that a solution for a multi-objective problem cannot be improved on one objective without worsening another objective. See Definition 2.5 (p. 16) for a formal definition.

and four different jobs, as follows (see Figure 4.1):

$O^{(1)}$: 1 organization with 2 jobs of length 1;

$O^{(2)}$: 1 organization with 1 job of length $\frac{1}{3}$;

$O^{(3)}$: 1 organization with 1 job of length $\frac{2}{3}$.

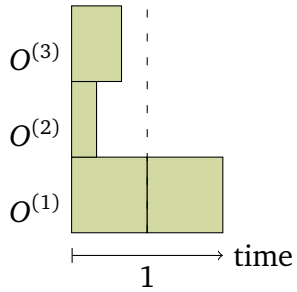


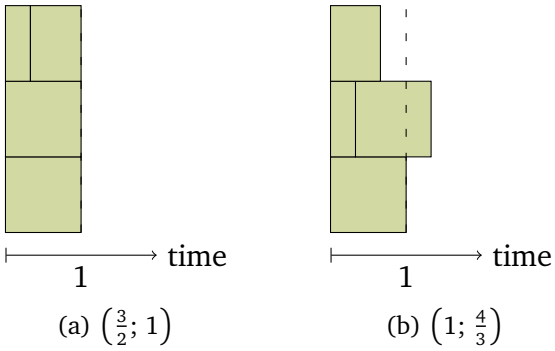
Figure 4.1: Simple instance with 3 organizations.

First, remark that if we want to obtain a makespan strictly better than 2, then it is impossible to schedule two jobs of organization $O^{(1)}$ on the same machine. Depending on the considered objective (global makespan and respect of the degree of cooperativeness α), a scheduling algorithm could be interested in either achieving the optimal global makespan ($C_{\max} = 1$) or respect α-MOSP relaxed local constraints with $\alpha = 1$ ($C_{\max}^{(k)} \leq 1 \cdot C_{\max}^{(k)\ local}, \forall k \in [1;N]$).

To achieve a makespan of 1, the jobs of organizations $O^{(2)}$ and $O^{(3)}$ must be scheduled together. In the best case, the job of $O^{(2)}$ is scheduled before the job of $O^{(3)}$ which leads to an approximation ratio of $\left(\frac{3}{2}; 1\right)$. This means that if the approximation ratio for the global makespan is bounded by 1, then no algorithm can construct a solution with a degree of cooperativeness better than $\frac{3}{2}$ (see Figure 4.2a).

On the other hand, if the schedule targets to achieve $C_{\max} < 2$ and a degree of cooperativeness $\alpha = 1$, then the jobs of $O^{(2)}$ and $O^{(3)}$ must start at time 0. Starting the first job of $O^{(1)}$ at time 0, the second one can start as soon as the job of organization $O^{(2)}$ finishes. In this case, the approximation ratio obtained is $\left(1; \frac{4}{3}\right)$ (see Figure 4.2b).

As these two schedules are Pareto optimal solutions for this instance, there is no algorithm with a performance ratio strictly better than $\frac{3}{2}$ on the degree of



(a) $\left(\frac{3}{2}; 1\right)$ (b) $\left(1; \frac{4}{3}\right)$

Figure 4.2: The two relevant schedules.

cooperativeness and $\frac{4}{3}$ on the makespan at the same time, as there are no solutions for this instance with these values.

The principles demonstrated in this example can be extended with the following instances.

Family 1

Let us consider the following instance of the MOSP problem (depicted in Figure 4.3):

- N organizations;
- $O^{(1)}$ has $n^{(1)} = N - 1$ identical jobs of length 1;
- For $2 \leq k \leq N$, $O^{(k)}$ has only one job of length $a_i = \frac{1}{N-1}$.

The total work to be done is equal to $\sum_{k,i} p_i^{(k)} = N$ and the optimal $C_{\max}$ for this instance *without the* MOSP *constraints* is obtained by scheduling each job of length 1 on different machines and then, scheduling all jobs of length $\frac{1}{N-1}$ on the remaining machine. The optimal $C_{\max}$ is equal to 1 (see Figure 4.4a).

Suppose now that we want to guarantee a value $1 \leq x < 2$ for the approximation ratio of the global makespan. This means that we must schedule all the jobs of $O^{(1)}$ in different organizations and that they must start at most at time $t = x - 1$.

If we set $x = 1 + \frac{1}{N-1}$, then we can schedule all the jobs of length a_i before the jobs of length 1 (see Figure 4.4b). This leads to a $\left(1; 1 + \frac{1}{N-1}\right)$-approximation. On the

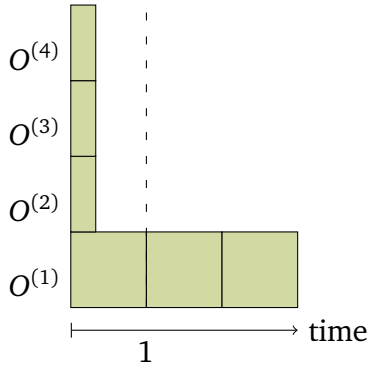
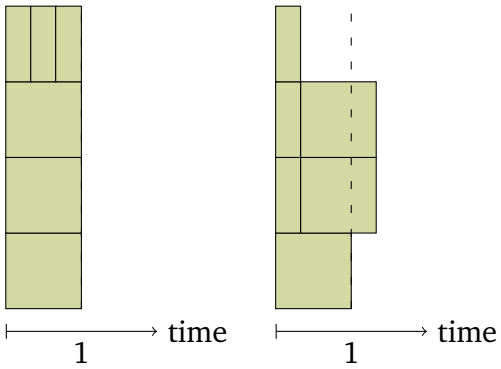


Figure 4.3: Instance of Family 1 for $N = 4$.



(a) $(N-1;\,1)$ (b) $\left(1;\,1+\frac{1}{N-1}\right)$

Figure 4.4: The two extreme Pareto optimal schedules of family 1 for $N = 4$.

other hand, if we set $x = 1+\frac{1}{N-1}-\epsilon$ then all jobs a_i must be scheduled on the machine that does not have any job of length 1. This schedule increases the makespan of the jobs a_i by a factor of at most $N-1$ and we get a $\left(N-1;\,1+\frac{1}{N-1}-\epsilon\right)$-approximation.

This family of instances shows that although the guarantee $\left(N-1;\,1+\frac{1}{N-1}\right)$ seems far from the two Pareto optimal solutions $\left(1;\,1+\frac{1}{N-1}\right)$ and $(N-1;\,1)$, we cannot improve simultaneously the solution for both objectives.

For $N = 3, 4, 5$ and 6, we have Pareto optimal guarantees of $\left(2;\,\frac{3}{2}\right)$, $\left(3;\,\frac{4}{3}\right)$, $\left(4;\,\frac{5}{4}\right)$, and $\left(5;\,\frac{6}{5}\right)$, respectively.

Family 2

Now consider the family of instances of the MOSP problem described as follows. Let j,k be integers such that $j > 1$ and $k > j-2$. We define three classes of organizations:

$O^{(A)}$: $(j-1)k$ organizations with only one job of length $\frac{j-1}{j}$;

$O^{(B)}$: k organizations with $j-1$ jobs of length $\frac{1}{j}$;

$O^{(C)}$: 1 organization with $k+1$ jobs of length 1.

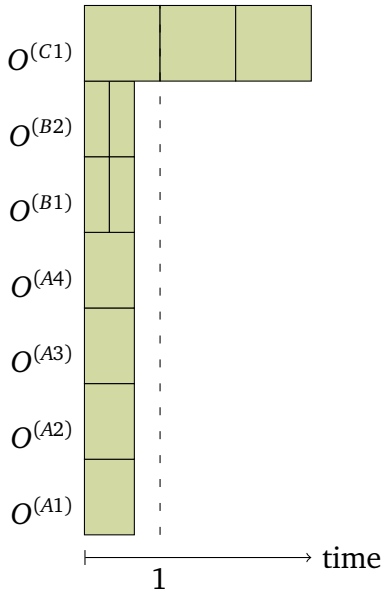


Figure 4.5: Instance of Family 2 for $(j = 3; k = 2)$.

In order to reach the optimal makespan, each job of organization $O^{(C)}$ must be scheduled alone on an organization. Each one of the jobs of $O^{(B)}$ must be scheduled together with a job from $O^{(A)}$, in such a way that each pair is scheduled alone on an organization and the job from $O^{(B)}$ is scheduled before the one from $O^{(A)}$. The global makespan is optimal ($C_{\max} = 1$) and the degree of cooperativeness is equal to $\frac{\frac{j-1}{j}+\frac{1}{j}}{\frac{j-1}{j}} = \frac{j}{j-1}$ (this configuration is shown in Figure 4.6a).

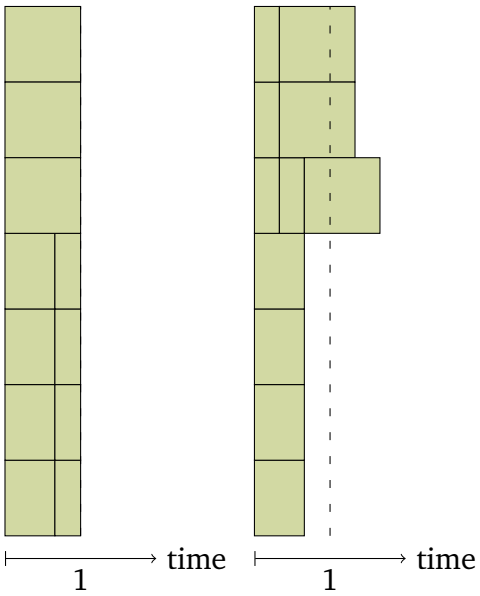
Proposition 4.1. *To improve the degree of cooperativeness to a value better than* $\frac{j}{j-1}$, *the makespan must be at least equal to* $1+\frac{j-1}{j}$.

Proof. To prove this, we need to look at what would be an implication of a lower degree of cooperativeness. If the degree of cooperativeness is lower than $\frac{j}{j-1}$, then each job of a type $O^{(A)}$ organization has to be scheduled without any other job of organizations $O^{(A)}$ or $O^{(B)}$. This only leaves $k+1$ machines to schedule the $k(j-1)$ jobs of organizations $O^{(B)}$. Furthermore, if the makespan is strictly lower than $1+\frac{j-1}{j}$, only one job of $O^{(C)}$ and at most $j-2$ jobs of $O^{(B)}$ can be scheduled on those $k+1$ machines (see Figure 4.6b).

However, as $k > j-2$, we have:

$$(k+1)(j-2) = k(j-2)+j-2 < k(j-2)+k = k(j-1)$$

Therefore, if the makespan is strictly lower than $1+\frac{j-1}{j}$ there is at least a machine with a job of type $O^{(B)}$ and one of type $O^{(A)}$, which leads to a cooperativeness ratio of at least $\frac{j}{j-1}$. □



(a) $\left(\frac{j}{j-1}; 1\right)$ (b) $\left(1; 1+\frac{j-1}{j}\right)$

Figure 4.6: Two Pareto optimal schedules of Family 2 for $(j=3; k=2)$.

The Pareto optimal guarantees are then $\left(\frac{j}{j-1}; 1+\frac{j-1}{j}\right)$. For $j=2,3,4$ and 5, we have Pareto optimal guarantees of $\left(2; \frac{3}{2}\right)$, $\left(\frac{3}{2}; \frac{5}{3}\right)$, $\left(\frac{4}{3}; \frac{7}{4}\right)$, and $\left(\frac{5}{4}; \frac{9}{5}\right)$, respectively.

Summary

In this section, we have studied the trade-offs between the degree of cooperativeness of the organizations and the best global makespan attainable that respects the relaxed MOSP constraints. We presented two families of instances and their respective sets of Pareto optimal approximation ratios.

The first family of instances, presented in Section 4.2, improves the previous known bounds established by Ooshita et al. [OII09] in two distinct ways.

First, we proved that what was previously though as not polynomially approximable (*unless* $P = NP$) is actually not approximable at all. We show that, for this family of instances, it is not possible to simultaneously improve the guarantee of $\left(N; 1 + \frac{1}{N-1}\right)$ on both criteria.

Second, despite the fact that the approximation ratios obtained in families 2 and 3 are in the curve $\frac{(\alpha+1)}{\alpha}$ presented in [OII09], better ratios are theoretically still possible if only one criterion is improved. The rectangles in Figure 4.7 mark the area that is known to be not attainable. However, points on the outline of the rectangles (that are not covered by other rectangles) are still attainable.

The second family of instances, presented in Section 4.2, shows that ratios of the form $\left(\frac{j}{j-1}; 1 + \frac{j-1}{j}\right)$ are Pareto optimal. This result also shows that for large values of j, we have $\lim_{j\to\infty}\left(1 + \frac{j-1}{j}; \frac{j}{j-1}\right) = (2; 1)$. This means that the ratio $(2; 1)$ obtained with the algorithms ILBA, LPT-LPT, SPT-LPT, and Less Helped First—presented in Section 3.3 — is also Pareto optimal.

Figure 4.7 summarizes the inapproximation bounds presented in this section.

4.3 Approximation algorithms

We have developed two algorithms that guarantees an approximation factor for the MOSP problem with relaxed local constraints of $\left(2; \frac{3}{2}\right)$ and $\left(3; \frac{4}{3}\right)$. It means that if the makespan of an organization is worsened by a factor at most of 2 (respectively 3), then the global makespan is no more than $\frac{3}{2}$ (respectively $\frac{4}{3}$) of the optimal.

Since the value for the optimal makespan is unknown, we use the dual approximation technique introduced by Hochbaum and Shmoys [HS88], that uses a binary search approach to estimate the value of the optimal makespan. For the sake of

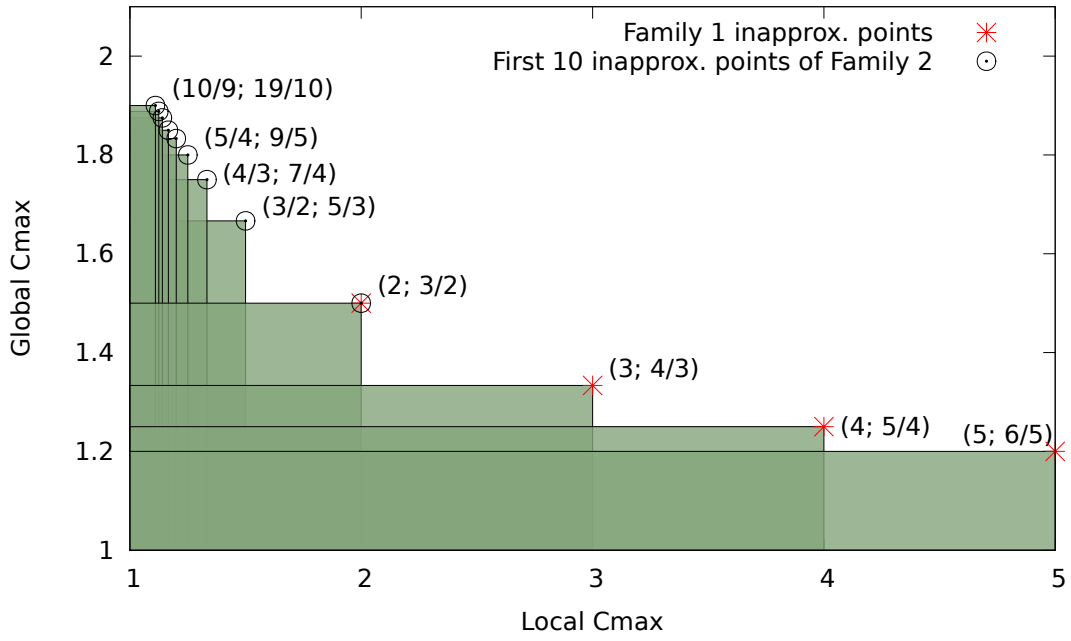


Figure 4.7: Inapproximation bounds defined by points belonging to families 1 and 2. The area inside the rectangles are not attainable.

clarity — and without loss of generality — we rescale the length of all jobs in such a way that 1 is the length of optimal makespan.

Principle

The main idea of the algorithms presented in this section is to allocate optimally all the *large* jobs. This placement will be used to determine where to schedule the remaining jobs of the organizations, with respect to the relaxed local constraints. The definition of what is a *large* job depends on the value of the target global makespan. Both presented algorithms are designed on the same basic structure of job partitioning. The principle can be decomposed into five successive phases as follows:

1. all organizations are classified according to the value of their initial makespan and presence of a *large* job;

2. an algorithm is used to determine the placement for the *large* jobs;

3. some of the large jobs are used to determine where to schedule all the jobs from their owners;

4. entire organizations are migrated to machines with total load less than the optimal makespan;

5. finally, the remaining organizations (the ones with initial makespans *too small* or *too large*) are allocated and the entire schedule is produced, starting all the jobs as soon as possible in the predetermined order to remove idle times.

Algorithm 1: $\left(2; \frac{3}{2}\right)$

The first approximation algorithm presented computes a schedule that is a $\frac{3}{2}$-approximation for the global makespan, while ensuring that no organization will have its makespan more than doubled if compared with its initial makespan.

Phase 1: Classify each organization into one of the following disjoint groups:

- $\mathcal{A} = \{O^{(k)} \mid C_{\max}^{(k)\ local} \leq \frac{1}{2}\}$;
- $\mathcal{B}_1 = \{O^{(k)} \mid \frac{1}{2} < C_{\max}^{(k)\ local} \leq \frac{3}{4}$ and $\nexists\ J_i^{(k)}$ such that $p_i^{(k)} > \frac{1}{2}\}$;
- $\mathcal{B}_2 = \{O^{(k)} \mid \frac{1}{2} < C_{\max}^{(k)\ local} \leq \frac{3}{4}$ and $\exists!\ J_i^{(k)}$ such that $p_i^{(k)} > \frac{1}{2}\}$;
- $\mathcal{C} = \{O^{(k)} \mid C_{\max}^{(k)\ local} > \frac{3}{4}\}$.

This classification gathers organizations that have small initial makespans ($\mathcal{A}$), that are hard to schedule with respect to the local constraints; organizations that are easy to schedule ($\mathcal{C}$), since within the global makespan of $\frac{3}{2}$ they will always fufill the local constraints; and intermediate organizations ($\mathcal{B}_1$ and $\mathcal{B}_2$) that need to be carefully scheduled. In this classification, *large* jobs are defined as jobs whose processing times $p_i^{(k)}$ are strictly greater than $\frac{1}{2}$.

Phase 2: We now assign all large jobs one per processor, from the end of the schedule under construction (time $= \frac{3}{2}$) to the beginning (time $= 0$).

Phase 3: The assignment calculated is adjusted as follows. Large jobs owned by organizations in $\mathcal{C}$ remain untouched, while each large job owned by an organization in $\mathcal{B}_2$ is replaced by all jobs of its owner (including itself).

Phase 4: Group all organizations from $\mathcal{B}_1$ and $\mathcal{B}_2$ into pairs ($O^{(i)}$ and $O^{(j)}$, where $C_{\max}^{(i)\ local} \leq C_{\max}^{(j)\ local}$) in such a way that each machine has only one pair and all jobs from $O^{(i)}$ are scheduled before the jobs from $O^{(j)}$. If $|\mathcal{B}_1| + |\mathcal{B}_2|$ is odd, then schedule the jobs from the last remaining organization on their original machines at the beginning of the schedule.

Phase 5: Assign all jobs owned by organizations in $\mathcal{A}$ and the remaining jobs from the last organization $\mathcal{B}_1$ (if any) into their original machines. Then, assign all remaining jobs from organizations in $\mathcal{C}$ into the scheduling using any list scheduling algorithm. Then, compact the schedule, i.e., remove any idle time by executing early the jobs starting after an idle time.

Analysis

The Phase 2 of the algorithm presented in the previous section schedules the *large* jobs, i.e. those with $p_i^{(k)} > \frac{1}{2}$, one per processor. The number of large jobs is limited to N as shown in the following lemma:

Lemma 4.2. *There are at most N jobs $J_i^{(k)}$ such that $p_i^{(k)} > \frac{1}{2}$.*

Proof. By contradiction. Suppose that there are N organizations with $N+1$ large jobs. Since the number of large jobs is larger than the number of organizations, two large jobs with processing times $p_i^{(k)}$ and $p_j^{(l)}$ must be assigned to a same machine on the optimal schedule. Then $C_{\max} \geq p_i^{(k)} + p_j^{(l)} > 1$, which contradicts the fact that the optimal schedule must be equal to 1. □

During Phase 4, pairs of organizations from $\mathcal{B}_2$ have all their jobs assigned to a same machine and they are scheduled one after the other. The following lemma shows why stacking two organizations does respect the α-MOSP constraints.

Lemma 4.3. *Given two organizations $O^{(i)}$ and $O^{(j)}$, if $C_{\max}^{(i)\ local} \leq C_{\max}^{(j)\ local}$ then all jobs from these organizations can be scheduled sequentially with respect to their relaxed local constraints.*

Proof. If all jobs from $O^{(i)}$ are followed by all jobs from $O^{(j)}$ on the same machine, we have $C_{\max}^{(i)} = C_{\max}^{(i)\ local}$ and

$$\begin{aligned} C_{\max}^{(j)} &= C_{\max}^{(i)\ local} + C_{\max}^{(j)\ local} \\ &\leq C_{\max}^{(j)\ local} + C_{\max}^{(j)\ local} \\ &= 2C_{\max}^{(j)\ local} \end{aligned}$$

□

At the end, organizations from $\mathcal{B}_2$ were coupled and assigned to machines with a total load greater than 1 and smaller than $\frac{3}{2}$.

Lemma 4.4. *Jobs of organizations from $\mathcal{A}$ always can be scheduled at the beginning of the schedule before large jobs from organizations in $\mathcal{C}$ during Phase 5.*

Proof. Since $|\mathcal{A}| + |\mathcal{B}_1| + |\mathcal{B}_2| \leq N$, it is sufficient to schedule organizations from $\mathcal{A}$ on the same machines where large jobs from $\mathcal{C}$ were assigned. Since a large job from $\mathcal{C}$ has processing time at most equal to 1, then the total load of the machine will be less than or equal to $\frac{1}{2} + 1 = \frac{3}{2}$. □

Theorem 4.5. *The schedule generated by Algorithm 1 is a $\frac{3}{2}$-approximation for the global makespan and no organization has its makespan more than doubled if compared with its initial makespan.*

Proof. First, we will show that the makespan obtained by the algorithm is a $\frac{3}{2}$-approximation for the global makespan.

We start by remarking that during Phase 4, organizations from $\mathcal{B}_1$ and $\mathcal{B}_2$ are coupled and each pair is assigned to a different machine. The total load on these machines is strictly greater than $2 \cdot \frac{1}{2} = 1$ and less than or equal to $2 \cdot \frac{3}{4} = \frac{3}{2}$. If $|\mathcal{B}_1| + |\mathcal{B}_2|$ is odd, the machine with the organization that is scheduled alone has a total load bounded by $\frac{3}{4}$.

From Lemma 4.4, it is sufficient to schedule all jobs from $\mathcal{A}$ before the large jobs from $\mathcal{C}$. Since $C_{\max}^{(\mathcal{A})\ local} \leq \frac{1}{2}$ and $\max_{i,k} p_i^{(k)} \leq 1$, then the load on these machines are bounded by $\frac{1}{2} + 1 = \frac{3}{2}$.

Finally, after Phase 5, all remaining small jobs ($p_i^{(k)} \leq \frac{1}{2}$) are scheduled using any list scheduling algorithm. Since there always exists a machine with load < 1

available (otherwise the total work to be done would be larger than N and the optimal makespan would be larger than 1), there is always a machine in which these jobs can be scheduled. The remaining jobs are smaller than $\frac{1}{2}$, therefore the load does not exceed $1+\frac{1}{2}=\frac{3}{2}$. This finishes the proof that the schedule generated is a $\frac{3}{2}$-approximation for the global makespan.

Furthermore, it is easy to remark that no organization will have its makespan more than doubled. Organizations from $\mathcal{A}$ will be scheduled at the beginning of the schedule and will not be delayed at all. Organizations from $\mathcal{B}_1$ and $\mathcal{B}_2$ will remain alone on their own machines or will be scheduled together. Lemma 4.3 guarantees that no stacked organization will be delayed. Organizations from $\mathcal{C}$ can be scheduled anywhere from $t=0$ to $t=\frac{3}{2}$, since $2\cdot C_{\max}^{(\mathcal{C})}\leq 2\cdot\frac{3}{4}=\frac{3}{2}$. □

Algorithm 2: $\left(3;\frac{4}{3}\right)$

Using the same frame as in Algorithm 1, we define below a new algorithm which achieves a $\frac{4}{3}$-approximation for the global makespan while guaranteeing that no organization will have its local makespan more than tripled.

This algorithm also has five phases described below.

Phase 1: Classify each organization into one of the following disjoint groups:

- $\mathcal{A}=\{O^{(k)} \mid C_{\max}^{(k)\ local}\leq\frac{1}{3}\}$;
- $\mathcal{B}_1=\{O^{(k)} \mid \frac{1}{3}<C_{\max}^{(k)\ local}\leq\frac{4}{9}$ and $\nexists\, J_i^{(k)}$ such that $p_i^{(k)}>\frac{1}{3}\}$;
- $\mathcal{B}_2=\{O^{(k)} \mid \frac{1}{3}<C_{\max}^{(k)\ local}\leq\frac{4}{9}$ and $\exists!\, J_i^{(k)}$ such that $p_i^{(k)}>\frac{1}{3}\}$;
- $\mathcal{C}=\{O^{(k)} \mid C_{\max}^{(k)\ local}>\frac{4}{9}\}$.

Phase 2: We first pair all the large jobs using an LPT scheduling order. Since a large job has $p_i^{(k)}>\frac{1}{3}$, we have at most $2N$ large jobs owned by organizations either in $\mathcal{B}_2$ or $\mathcal{C}$. Remark that if there are x large jobs with $x\leq N$, then there will be no pairing, and there will even be $N-x$ machines with no large jobs.

During this phase, we first allocate jobs from $\mathcal{B}_2$ on their own organizations, or if two jobs from $\mathcal{B}_2$ have to be scheduled together we do so on the organization with the largest index. We place paired jobs of $\mathcal{C}$ (or single jobs of $\mathcal{C}$) on the remaining organizations.

In the rest of the algorithm, we will note $(\mathcal{B}_2,\mathcal{C})$ the set of pairs created in this phase with one member of $\mathcal{B}_2$ and one member of $\mathcal{C}$. Similarly, we will use the sets $(\mathcal{C},\mathcal{C})$ and $(\mathcal{B}_2,\mathcal{B}_2)$ and if the number of large jobs is strictly lower than $2N$, the sets $(\mathcal{B}_2)$ and $(\mathcal{C})$ will denote respectively the jobs of $\mathcal{B}_2$ and $\mathcal{C}$ which were not matched in a pair.

Phase 3: As in the previous algorithm, the assignment calculated in Phase 2 is adjusted and each large job owned by an organization in $\mathcal{B}_2$ is replaced by all the jobs of its owner (including itself). These jobs will be tied together in the rest of the algorithm and treated as a unique job.

Phase 4: Since we have at most $2N$ large jobs, the schedule generated in Phase 2 can contain machines with jobs from only one $\mathcal{B}_2$ or $\mathcal{C}$ organization, or jobs from two organizations taken in $\mathcal{B}_2$ or $\mathcal{C}$.

As long as the set $(\mathcal{B}_2,\mathcal{C})$ has at least two elements, we consider two such pairs. Let $O^{(i)}$ be the organization from $\mathcal{B}_2$ whose job is in the first pair from $(\mathcal{B}_2,\mathcal{C})$, and $O^{(j)}$ be the organization from $\mathcal{B}_2$ whose job is in the second pair. Supposing that $i > j$, we put all the jobs from $O^{(i)}$ and $O^{(j)}$ on the machine i, and the two jobs from $\mathcal{C}$ they were paired with on machine j. Jobs on machine i are ordered according to their local makespan. The sets $(\mathcal{B}_2,\mathcal{B}_2)$, $(\mathcal{C},\mathcal{C})$ and $(\mathcal{B}_2,\mathcal{C})$ are then updated accordingly.

This leaves us with many pairs in $(\mathcal{B}_2,\mathcal{B}_2)$ and $(\mathcal{C},\mathcal{C})$ and at most one in $(\mathcal{B}_2,\mathcal{C})$. Remember that the sets $(\mathcal{B}_2)$ and $(\mathcal{C})$ are eventually not empty if there were less than $2N$ large jobs in Phase 2. Jobs in $(\mathcal{B}_2)$ are allocated to their own organization machine.

Using a similar argument from Lemma 4.3 we can show that up to three organizations can be scheduled sequentially if the jobs are scheduled from the organization with smaller initial makespan to the organization with the larger makespan.

As long as there are organizations of type $\mathcal{B}_1$ which are unaffected, these organizations are distributed, allocating one to each pair in $(\mathcal{B}_2,\mathcal{B}_2)$, and two to each single organization in $(\mathcal{B}_2)$. After this stage, if there are some organizations from $\mathcal{B}_1$ which are unaffected, then all the organizations from $\mathcal{B}_2$ are either in triplets from $\mathcal{B}_1$ and $\mathcal{B}_2$ or in the last pair of $(\mathcal{B}_2,\mathcal{C})$.

If there were strictly less than N large jobs in Phase 2, the empty machines are filled with triplets of organizations from $\mathcal{B}_1$, as long as that is possible. After this stage, either all the jobs from organizations in $\mathcal{B}_1$ and $\mathcal{B}_2$ have been allocated

somewhere, or all the machines either have a triplet from jobs in $\mathcal{B}_1$ and/or $\mathcal{B}_2$, or they have at least one large job from $\mathcal{C}$.

Phase 5: Assign all jobs owned by organizations in $\mathcal{A}$ to their original machines with an original pair of ($\mathcal{C}$,$\mathcal{C}$) from Phase 2 or a single job from ($\mathcal{C}$) if the remaining pairs in ($\mathcal{C}$,$\mathcal{C}$) are only pairs formed during Phase 4. Then, assign all remaining jobs from organizations $\mathcal{B}_1$ sequentially to any machine with less than 1 unit of workload.

Finally, assign all the remaining small jobs in $\mathcal{C}$ to the scheduling using any list scheduling algorithm. The jobs executions are ordered on each machine according to their original local makespan.

Analysis

Lemma 4.6. *There are at most $2N$ jobs $J_i^{(k)}$ such that $p_i^{(k)} > \frac{1}{3}$*

Proof. By contradiction, similar to the proof of Lemma 4.2. Suppose that there are N organizations with $2N+1$ large jobs. Since the number of large jobs is larger than twice the number of organizations, three large jobs must be assigned to a same machine on the optimal schedule. Then $C_{\max} \geq 3 \cdot p_i^{(k)} > 3 \cdot \frac{1}{3} > 1$, which contradicts the fact that the optimal schedule must be equal to 1. □

Lemma 4.7. *When considering a pair of ($\mathcal{B}_2$,$\mathcal{C}$) formed in Phase 2, the length of the jobs from $\mathcal{C}$ is less than $\frac{2}{3}$, and scheduling two such jobs together is possible within the targeted global makespan.*

Proof. By contradiction. Suppose that LPT scheduled the pair ($\mathcal{B}_2$,$\mathcal{C}$) together and that the lengths of the jobs from $\mathcal{C}$ are greater than or equal to $\frac{2}{3}$. Since the makespans of organizations in $\mathcal{B}_2$ are greater than $\frac{1}{3}$, the makespan obtained on the machine where the pair were assigned is greater than $\frac{2}{3} + \frac{1}{3} = 1$, which contradicts the fact that LPT scheduled the first two jobs optimally on each machine [Gra69]. □

Lemma 4.8. *Scheduling all the jobs from any triplet of organizations of type $\mathcal{B}_1$ or $\mathcal{B}_2$ on a single machine is always possible within the target bounds on global makespan and degree of cooperativeness.*

Proof. Since the makespan of any organization in $\mathcal{B}_1$ and $\mathcal{B}_2$ is at most $\frac{4}{9}$, scheduling a triplet of these organizations on a same machine produces a makespan of at most $3 \cdot \frac{4}{9} = \frac{4}{3}$, which respects the target bounds on global makespan.

Similarly to Lemma 4.3, this triplet can be scheduled sequentialy while respecting the degree of cooperativeness. Let $O^{(i)}$, $O^{(j)}$, and $O^{(k)}$ be the organizations from this triplet with $C_{\max}^{(i)\ local} \leq C_{\max}^{(j)\ local} \leq C_{\max}^{(k)\ local}$. If the organizations are scheduled from the one with smaller initial makespan to the one with larger makespan, we have:

- $C_{\max}^{(i)} = C_{\max}^{(i)\ local}$;
- $C_{\max}^{(j)} = C_{\max}^{(i)\ local} + C_{\max}^{(j)\ local} \leq 2C_{\max}^{(j)\ local}$;
- $C_{\max}^{(k)} = C_{\max}^{(i)\ local} + C_{\max}^{(j)\ local} + C_{\max}^{(k)\ local} \leq 3C_{\max}^{(k)\ local}$.

This shows that all organizations have their makespan at most tripled and that the targeted degree of cooperativeness $\alpha = 3$ is respected. □

Lemma 4.9. *There are always enough machines left to assign all the organizations of type $\mathcal{A}$ to a machine where there are no jobs of type $\mathcal{B}_1$ or $\mathcal{B}_2$, and no pair of ($\mathcal{C}$,$\mathcal{C}$) formed during Phase 4.*

Proof. During phase 4, pairs of ($\mathcal{C}$,$\mathcal{C}$) are formed by splitting two pairs of ($\mathcal{B}_2$,$\mathcal{C}$) and creating two new pairs by grouping the two organizations of $\mathcal{B}_1$ in one pair and the two organizations of $\mathcal{C}$ in the other. This means that the number of pairs ($\mathcal{C}$,$\mathcal{C}$) generated by Phase 4 is bounded by half of the number of organizations in $\mathcal{B}_2$ (as are pairs ($\mathcal{B}_2$,$\mathcal{B}_2$)) and these pairs ($\mathcal{C}$,$\mathcal{C}$) are actually assigned to machines originally owned by an organization in $\mathcal{B}_2$. Since $|\mathcal{A}| + |\mathcal{B}_1| + |\mathcal{B}_2| < N$, there is always a machine to assign the jobs from $\mathcal{A}$ not owned by an organization from $\mathcal{B}_1$ or $\mathcal{B}_2$. □

Lemma 4.10. *Whenever there is a large job of $\mathcal{C}$ scheduled on a machine and no more than 1 unit of workload, jobs from organizations $\mathcal{B}_1$ can be added to the machine while still respecting the target bounds on global makespan and degree of cooperativeness.*

Proof. First, let us prove that the bound on the global makespan is respected. Individual jobs $J_i^{(k)}$ belonging to an organization $\mathcal{B}_1$ are all strictly shorter than $\frac{1}{3}$. Therefore, when adding such a job to a machine with less than 1 unit of workload, the total workload is strictly lower than $\frac{4}{3}$. Hence, when scheduling all the jobs without idle time, the makespan is strictly lower than $\frac{4}{3}$.

Since the targeted degree of cooperativeness is 3, the inserted job $J_i^{(k)}$ must complete at most at time $3C_{\max}^{(k)\ local}$. Since k is an organization of type $\mathcal{B}_1$, its local

makespan is more than $\frac{1}{3}$ and $3C_{\max}^{(k)\ local}$ is greater than 1. As there is a large job (i.e., of length more than $\frac{1}{3}$) of $\mathcal{C}$ on the machine, and this large job will be executed last with a global makespan lower than $\frac{4}{3}$, any job scheduled on the same machine will complete before time 1. In particular, job $J_i^{(k)}$ will complete before time 1, which is lower than thrice its local makespan. □

Lemma 4.11. *Remaining small jobs from organization $\mathcal{C}$ always have a machine to be scheduled on, while still respecting the target bounds on global makespan and degree of cooperativeness.*

Proof. Within the targeted makespan of $\frac{4}{3}$, the degree of cooperativeness $\alpha = 3$ is always respected for any organization k in $\mathcal{C}$ because $C_{\max}^{(k)\ local} > \frac{4}{9} \Rightarrow 3C_{\max}^{(k)\ local} > \frac{4}{3}$. This means that the remaining small jobs from $\mathcal{C}$ can always be scheduled at the end of the schedule, after all other jobs were assigned, while respecting the degree of cooperativeness.

Since there is always a machine with load less than 1, after the addition of a small job from $\mathcal{C}$ we will have a makespan smaller than $1 + \frac{1}{3} = \frac{4}{3}$ and, therefore, the target bound on the global makespan is respected. □

Theorem 4.12. *Algorithm 2 provides a schedule which has a global makespan lower than or equal to $\frac{4}{3}$, and for which each task $J_i^{(k)}$ completes at most at time $3C_{\max}^{(k)\ local}$.*

Proof. To prove the theorem, it is sufficient to note that:

- During Phase 1, the organizations are just structured in groups;
- During Phase 2, we pair large jobs to distribute them evenly and ensure that the global makespan will be lower than $\frac{4}{3}$. We prove in Lemma 4.6 that this pairing is possible, and at this stage the workload affected on each machine is strictly lower than 1, as proved in [Gra69];
- During Phase 3, we add at most $\frac{1}{9}$ units of workload to pairs ($\mathcal{B}_2$,$\mathcal{C}$), and $\frac{2}{9}$ units of workload to pairs ($\mathcal{B}_2$,$\mathcal{B}_2$);
- Since the organizations of type $\mathcal{B}_2$ have a local makespan lower than $\frac{4}{9}$, and as we proved in Lemma 4.7 that we can bound the length of jobs from $\mathcal{C}$ paired with a job from $\mathcal{B}_2$, the transformation done in Phase 4 between two pairs of

($\mathcal{B}_2$,$\mathcal{C}$) into one pair of ($\mathcal{B}_2$,$\mathcal{B}_2$) and one pair of ($\mathcal{C}$,$\mathcal{C}$) keeps the workload of all machines under the $\frac{4}{3}$ bound;

- The triplets formed in the second part of Phase 4 can be scheduled within the targeted bounds for the global makespan and degree of cooperativeness according to Lemma 4.8;
- During Phase 5, jobs from organizations of type $\mathcal{A}$ are alloted to machines with a workload lower than 1, since the pairs of ($\mathcal{C}$,$\mathcal{C}$) are original pairs from Phase 2, as proved in Lemma 4.9. The workload on these machines is then lower than or equal to $\frac{4}{3}$;
- In the second part of Phase 5, the remaining jobs from organizations $\mathcal{B}_1$ are alloted to the least utilized machines such that they will complete before thrice their local makespan as proved in Lemma 4.10;
- Finally, Lemma 4.11 states that all the remaining small jobs from organizations $\mathcal{C}$ can be scheduled within the global makespan bound.

□

4.4 Concluding remarks

In this chapter we presented our study on a relaxed form of the scheduling problem known as the Multi-Organization Scheduling Problem (MOSP). We investigated how limited cooperation between organizations can greatly improve the global performance of grid computing platforms. This relaxed form of MOSP is known in the literature as the *α-Cooperative Multi-Organization Scheduling Problem* (α-MOSP). It models the scheduling problem where organizations accept a limited degradation on their perceived performance in order to improve the quality of the global performance.

We improved the previously known inapproximation bounds for α-MOSP by showing that it is actually not polynomially approximable *even if* $P = NP$. We designed two families of instances whose Pareto optimal points corroborate the presented inapproximation bounds. Then, two new algorithms with guaranteed

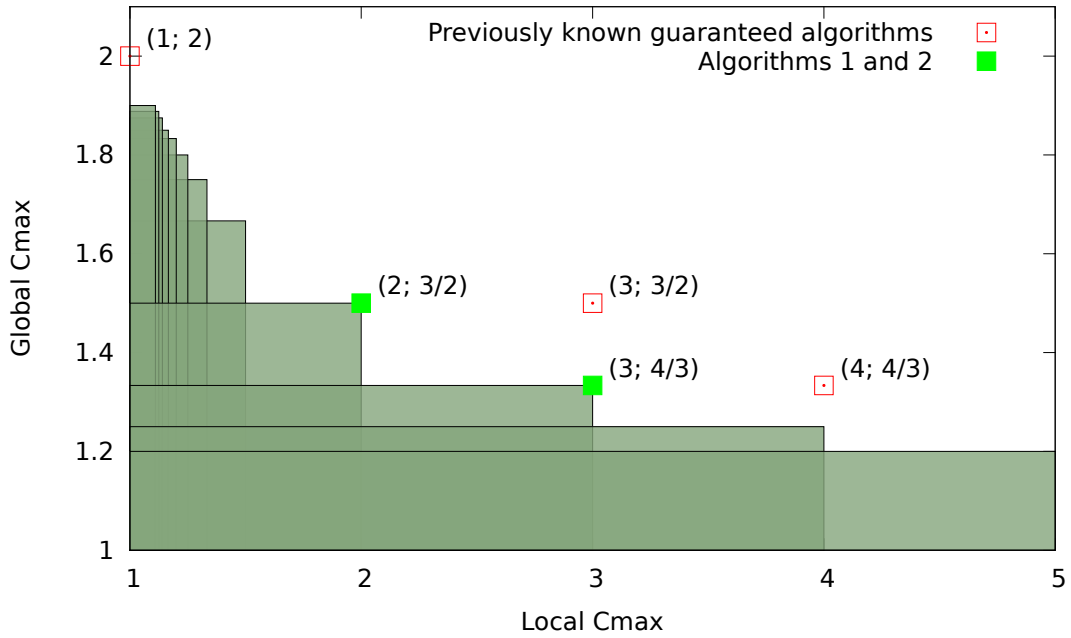


Figure 4.8: Graphical summary of the results. The figure shows the points that can be obtained by the currently known algorithms and the rectangles show the areas that are not attainable.

performance to solve the α-MOSP problem were developed and analyzed. We showed that the first one achieves a $\frac{3}{2}$-approximation for the obtained global makespan, while it guarantees that no organization will have its makespan more than doubled. The second one guarantees a $\frac{4}{3}$-approximation for the global makespan, while no organization has its makespan more than tripled.

The results presented in this chapter are summarized in Figure 4.8. The figure evidences the improvements of the known results and shows how close the new approximation ratios are from the Pareto set.

Chapter 5

Coordination mechanisms for selfish multi-organization scheduling

Until this point we have studied the notions of cooperation and selfishness on the Multi-Organization Scheduling Problem using classical combinatorial optimization approaches. In this chapter, we extend the notions of independence and selfishness organizations by allowing each of them to rationally choose the best strategy for its jobs. The goal is to study the interactions between the independent organizations as the result of rational selfish players attempting to reach an equilibrium.

The main contribution presented in this chapter is a game theoretic model for the MOSP problem — contemplating the individualism and selfishness of organizations collaborating in a grid computing system — that leads to configurations with a cost as close as wanted to a pure Nash equilibrium, and with a bounded price of anarchy. The idea is to study the problem as a non-cooperative game, providing coordination mechanisms allowing each organization to continuously attempt to obtain the best possible makespan until an equilibrium is achieved.

The remaining of the chapter is organized as follows. Section 5.1 describes our game-theoretic model for the Multi-Organization Scheduling Problem. Section 5.2 studies a coordination mechanism based on classical scheduling algorithms, showing

that these kind of mechanisms do not admit ϵ-approximate equilibrium for $\epsilon < 2$ and that the problem of deciding whether a particular instance admits a pure Nash equilibrium is co-NP hard. In Section 5.3 we define a new coordination mechanism where priorities are given to organizations. We present an algorithm that constructs pure ρ-approximate equilibria and analyze the price of anarchy. Finally, some conclusion remarks are presented in Section 5.4.

5.1 Game-theoretic model

In the last few years, scheduling problems have been intensively studied by researchers interested in both algorithms and game-theory — a branch of game theory known as *algorithmic game theory* [Nis+07]. Scheduling problems are typically modeled as *selfish load balancing*, non-cooperative games.

Load balancing games were first studied by Koutsoupias and Papadimitriou [KP99]. In their seminal work, the authors studied load balancing in the form of a routing game, consisting of two nodes connected by parallel edges with different speeds. They were the first to formally study the inefficiency caused by the lack of coordination. Their study led to the definition of price of anarchy — see Definition 2.12 (p. 21) — comparing the cost of Nash equilibria to the cost of the optimal (social cost).

In the context of scheduling, a selfish load balancing game [Vöc07] is a game where each player k is responsible for one job J_k. The goal of each player is to assign its job to a processor with the lowest load possible. The load of a processor j is defined by $\ell_j = \sum_{i|\pi(J_i)=j} p_i$. Thus, the set of possible strategies for player k is the set of available processors, i.e., $S_k = \mathcal{M}$. A player can choose a strategy $s_k \in S_k$ that represents one of the available processors. The cost c_k for the player is the load of the chosen machine, i.e., $c_k = \ell_{s_k}$. The social cost for the collectivity is defined as the makespan $C_{\max} = \max_{m\in\mathcal{M}} \ell_m$.

Two interesting properties of load balancing games are:

Theorem 5.1 ([Fot+02]). *For parallel independent machines and uniform machines, every instance of the load balancing game admits at least one pure Nash equilibrium.*

Theorem 5.2 ([FKS05] and [EKM07]). *For parallel independent machines and uniform machines, it is always possible to construct a Nash equilibrium in polynomial time for every instance of the load balancing game.*

These results were further extended by other works that studied a similar game, with one player in charge of one job, but this time with the cost function defined to be the completion time of the player's job (C_k). See, e.g., [Car+06; CKV02; DT09; Imm+09; KP09; LO01; SV07]. Czumaj, Krysta, and Vöcking [CKV02] gave tight results ($\Theta(\log m / \log\log m)$) for the price of anarchy on pure Nash equilibria using uniform parallel machines. Please refer to Dürr and Thang [DT09] for a comprehensive summary of these results.

All these works have in common the fact that each player is in control of only one job. The nature of the MOSP problem imposes a different setup, with each player responsible for one or more jobs. We will see in the remaining of this chapter that this changes considerably the properties of this scheduling game.

We associate one selfish agent with each organization, i.e., jobs originally from organization $O^{(k)}$ are managed by agent k. Each agent can choose on which organization each one of its jobs will be executed, knowing that each selfish organization will schedule first its own jobs.

In other words, a pure strategy s_k for player k is a vector of $n^{(k)}$ elements such that $s_k(i)$ corresponds to the organization chosen by player k for job $J_i^{(k)}$. As before, we denote S_k the set of all the strategies for agent k.

A configuration (or profile) M is a vector $(s_1, s_2, \ldots, s_N)$ such that s_k is the strategy chosen by agent k. The cost of an agent k under configuration M — denoted by $c_k(M)$ — corresponds to the makespan obtained by organization $O^{(k)}$: $C_{\max}^{(k)}$.

Like we saw in Section 2.3, when all agents choose an assignment for their jobs in such way that no agent has incentive to change its strategy, we say that this configuration is a *pure Nash equilibrium* (see Definition 2.10, p. 20).

However, some games do not always feature pure Nash equilibria. Nevertheless, these games can still have their stability studied through the concept of *ϵ-approximate equilibrium*. In ϵ-approximate equilibrium, each selfish agent is satisfied when the chosen strategy results in an approximation of its best response.

Definition 5.3 (ϵ-approximate equilibrium [Rad80]). A strategy vector $s \in S$ is said

to be a *ϵ-approximate equilibrium* if for all players i and each alternate strategy $s'_i \in S_i$, we have that:

$$c_i(s_i, s_{-i}) \leq \epsilon \cdot c_i(s'_i, s_{-i})$$

Every schedule has some social cost, as well as individual costs for every agent. The *social cost* of a configuration is the global makespan obtained on the system ($C_{\max}$). Due to the lack of coordination, configurations in equilibrium may have higher costs if compared to the global social optimum. A measure of this inefficiency is the *price of anarchy*. It is defined as the ratio between the cost of the worst Nash equilibrium and the optimal cost, which in general is not an equilibrium.

Local scheduling policy

Differently from problems like the selfish load balancing problem, the set of strategies chosen by all agents to define where each job will be executed is not sufficient to compute the final schedule of all jobs. We also need a way to specify how jobs assigned to a same machine will be scheduled.

In order to study the price of anarchy in such games, Christodoulou et al. [CKN04] introduced the notion of *coordination mechanisms*. A *coordination mechanism* is a set of *local policies*, one for every machine, that specify how the jobs assigned to this machine will be scheduled, independently of how the other jobs were assigned to other machines.

A coordination mechanism for the MOSP game must reflect the selfish behavior of each organization. In our study, we assume that all organizations are individualist and selfish. We also assume that each organization has full control on the scheduling of the jobs assigned to its machines, which means that, in theory, it is possible for organizations to cheat the devised global schedule by re-inserting their jobs earlier in the local schedules[1].

Therefore, a coordination mechanism for the MOSP game must consider that an organization will always prioritize the execution of its own jobs before any job owned by other organizations, no matter how this decision will impact the makespan

[1]For a more detailed study about the impact on the quality of the obtained schedule caused by this selfish behaviour, please refer to [Coh+10].

of other organizations. In other words, each organization will apply a "my jobs first" policy, scheduling its own jobs before jobs migrated from other organizations.

As long as the coordination mechanism schedules the local jobs at the beginning of the schedule, the order they get scheduled will not change the makespan obtained by the organization. The remaining (foreign) jobs must be scheduled according to some criteria. We added to the game model the notion of *job priority*. Organizations will compute the local scheduling of the foreign jobs according to their scheduling priority. These jobs will be scheduled in decreasing priority order, i.e., jobs with higher priority will be scheduled first and jobs with the same priority will be scheduled in no particular order.

In the remaining sections, we study the game regarding two different priority policies. First, we study the MOSP game with *priority given to the jobs*, where all jobs have a distinct priority. Then, we study the game with *priority given to organizations*, where all jobs from a same organization have the same priority, but each organization has a distinct priority.

5.2 Game with priority to jobs

Classical scheduling algorithms usually compute the order in which the jobs will be executed according to some rule that assigns a different priority calculated separately for each job. For instance, the Longest Processing Time first (LPT) scheduling algorithm prioritizes the jobs with larger processing times (the priority of each job is its length), the Shortest Processing Time first (SPT) algorithm prioritizes the jobs with smaller processing times, etc.

For the general scheduling problem, Immorlica et al. [Imm+09] analyzed games with coordination mechanisms based on some classical scheduling algorithms (with priorities given to jobs). They studied the scenario where each job is controlled by a selfish agent that selects the machine that minimizes the expected job completion time. Their study shows that the set of pure Nash equilibria can always be computed by the LPT algorithm if the coordination mechanism used is to prioritize the largest jobs (and, analogously, that the SPT algorithm calculates the set of pure Nash equilibria if the coordination mechanism prioritizes the smallest jobs).

Due to MOSP constraints, these results do not hold for MOSP games. With coordination mechanisms prioritizing jobs according to a criteria that does not depend on information about the organization that owns the jobs, the MOSP game may not admit a pure Nash equilibrium, independently of the coordination mechanism being used.

We start by showing that with this priority model, even if we consider the notion of approximate equilibria, we cannot always have pure ϵ-approximate equilibrium for values of $\epsilon < 2$. Then we show that the decision problem of deciding whether a particular instance admits a pure Nash equilibrium is co-NP hard.

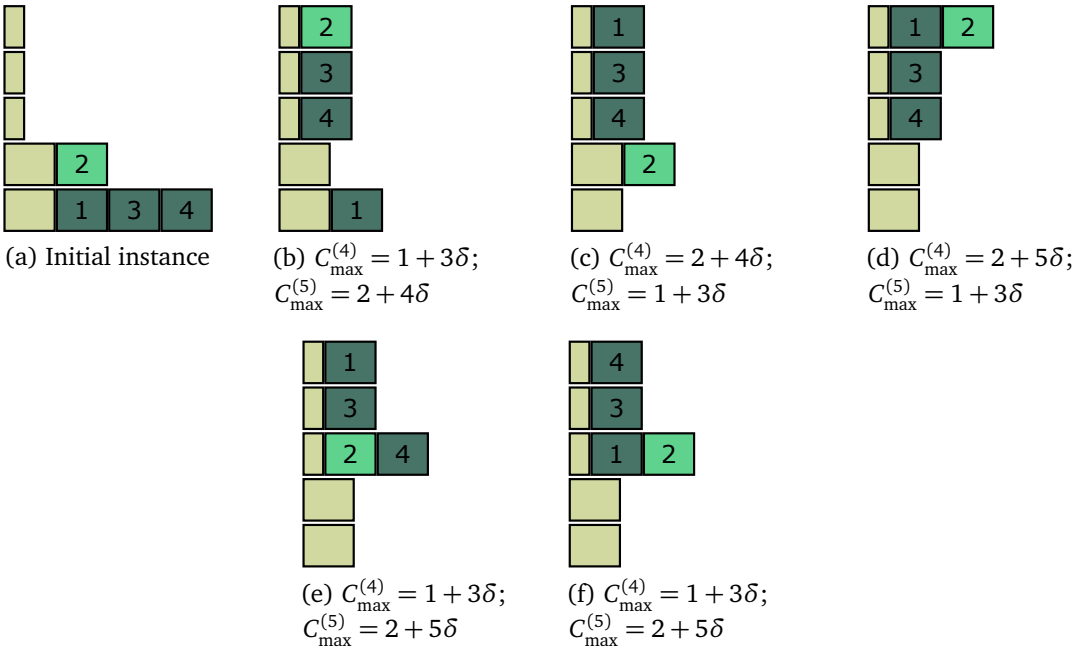
No pure ϵ-approximate equilibrium for $\epsilon < 2$

Theorem 5.4. MOSP *games defined with a coordination mechanism that assigns priorities to jobs independently of the owner organization do not admit a pure ϵ-approximate equilibrium for values of $\epsilon < 2$.*

Proof. We prove this theorem using the particular instance of the MOSP game depicted in Figure 5.1a. In this instance, organizations $O^{(1)}$, $O^{(2)}$, and $O^{(3)}$ have each one a job of length equal to δ, for some arbitrarily small constant $\delta > 0$. Organization $O^{(4)}$ has two jobs of length $1 + 2\delta$ and organization $O^{(5)}$ has four jobs also of length $1 + 2\delta$. We take an arbitrary coordination mechanism that prioritizes the jobs as follows: jobs $J_2^{(5)}$, $J_2^{(4)}$, $J_3^{(5)}$, and $J_4^{(5)}$ have priorities respectively equal to 1 (higher priority), 2, 3, and 4 (lower priority), while the remaining jobs have priorities lower than any of these four jobs. The priorities of these jobs are indicated in Figure 5.1.

MOSP local constraints impose that the low priority jobs $J_1^{(1)}$, $J_1^{(2)}$, and $J_1^{(3)}$ must be scheduled in their original organizations at time $t = 0$ (otherwise they would increase the makespan for their original organizations). Only organizations $O^{(4)}$ and $O^{(5)}$ are capable of improving their local makespans by changing the strategy for the higher priority jobs. The best makespan that organizations $O^{(4)}$ and $O^{(5)}$ can attain (while respecting MOSP's constraints) is equal to $1 + 3\delta$.

A best response policy for this game does not converge to an equilibrium. Figures 5.1d, 5.1e, and 5.1f show that both organizations can use their higher priority jobs to produce a configuration with the optimal makespan possible $(1 + 3\delta)$ by delaying a job from the other organization and increasing its makespan to $2 + 5\delta$.



(a) Initial instance

(b) $C_{\max}^{(4)} = 1+3\delta$; $C_{\max}^{(5)} = 2+4\delta$

(c) $C_{\max}^{(4)} = 2+4\delta$; $C_{\max}^{(5)} = 1+3\delta$

(d) $C_{\max}^{(4)} = 2+5\delta$; $C_{\max}^{(5)} = 1+3\delta$

(e) $C_{\max}^{(4)} = 1+3\delta$; $C_{\max}^{(5)} = 2+5\delta$

(f) $C_{\max}^{(4)} = 1+3\delta$; $C_{\max}^{(5)} = 2+5\delta$

Figure 5.1: Instance with no ϵ-approximate equilibrium for $\epsilon < 2$.

Among all the possible configurations respecting MOSP's constraints, no configuration is a pure Nash equilibrium. Figures 5.1b and 5.1c show the only *Pareto efficient* configurations for organizations $O^{(4)}$ and $O^{(5)}$, i.e., the configurations where the $C_{\max}$ of one organization cannot be improved without increasing the $C_{\max}$ of the other. When one of the organizations attain the optimal local makespan of $1+3\delta$, the best makespan that can be obtained by the other is $2+4\delta$.

In this example, the Pareto efficient configurations give the smallest ϵ needed to obtain an approximate pure equilibrium. No pure ϵ-approximate equilibrium exists unless $\epsilon \geq \frac{2+4\delta}{1+3\delta} \Rightarrow \epsilon \geq 2$.

This shows that MOSP games with coordination mechanisms that assign priorities to jobs do not always admit a pure ϵ-approximate equilibrium if $\epsilon < 2$. □

Note that the result of Theorem 5.4 shows that there is a large class of different coordination mechanisms that do not admit ϵ-approximate equilibrium for values of $\epsilon < 2$ for the MOSP game. Most notably, the theorem applies to games with coordination mechanisms based on classical scheduling algorithms like LPT, SPT, etc.

co-NP hardness

In this section, we show that deciding if a particular instance of MOSP game has a pure Nash equilibrium is co-NP-hard.

The idea of the proof is to build an instance of the MOSP problem with three types of jobs: (i) large jobs that cannot be scheduled on other organizations because of MOSP local constraint; (ii) a set of n jobs with integer lengths that are associated with the integers to be partitioned on two subsets with the same sum; (iii) three jobs — with the three (different) smallest priorities of all jobs of the instance — that will cause the disequilibrium when the jobs of type (ii) can be partitioned. More formally:

Theorem 5.5. *The decision problem of deciding whether* MOSP *has a pure Nash equilibrium is co-NP-hard when different priorities are given to jobs.*

Proof. The decision version of the problem of deciding whether a given instance of the problem MOSP has a pure Nash equilibrium can be stated as follows:

Instance: a set of N organizations $\{O^{(1)}, O^{(2)}, \ldots, O^{(k)}, \ldots, O^{(N)}\}$ and their respective jobs $J_i^{(k)}$.

Question: does the instance admit a pure Nash equilibrium?

In order to prove the theorem, we reduce the classical PARTITION problem [GJ79], known to be NP-complete, to the **complement** of the problem of deciding whether an instance of MOSP has a pure Nash equilibrium, i.e., the problem of knowing if the instance **does not** admit a pure Nash equilibrium. Remind that the PARTITION problem can be stated as follows:

Instance: a set of n integers $s_1, s_2, \ldots, s_n$.

Question: does there exist a subset $J \subseteq I = \{1, \ldots, n\}$ such that:

$$\sum_{i \in J} s_i = \sum_{i \in I \setminus J} s_i \ ?$$

Given an instance of the PARTITION problem, we construct an instance of MOSP problem with $N = 5$ organizations as follows:

- $O^{(1)}$ and $O^{(2)}$ have each one a job with length equal to 1;
- $O^{(3)}$ has $n+1$ jobs, where job $J_1^{(3)}$ has length equal to $\sum_{i\in I}\frac{s_i}{2}+3$ and, for $1 \leq i \leq n$, job $J_{i+1}^{(3)}$ has length s_i;
- $O^{(4)}$ has 3 jobs: $J_1^{(4)}$ with length equal to $\sum_{i\in I}\frac{s_i}{2}+3$, $J_2^{(4)}$ with length equal to 1, and $J_3^{(4)}$ with length 3;
- $O^{(5)}$ has 2 jobs $J_1^{(5)}$ and $J_2^{(5)}$ with lengths equal to $\sum_{i\in I}\frac{s_i}{2}+3$ and 2.5, respectively.

In order to simplify the notation used in the proof, we assume, without loss of generality, that the instance of the PARTITION problem is composed of integers s_i such that $s_i \geq 10$.

We fix a policy for this MOSP game, with different priority given to the jobs, in order to analyze the equilibria of the game. The policy used by each organization is to first schedule its own jobs, then schedule the jobs from $O^{(3)}$ (in no particular order) and finally schedule the remaining jobs in non-decreasing order of jobs' length.

This instance is constructed in polynomial time and is depicted in Figure 5.2.

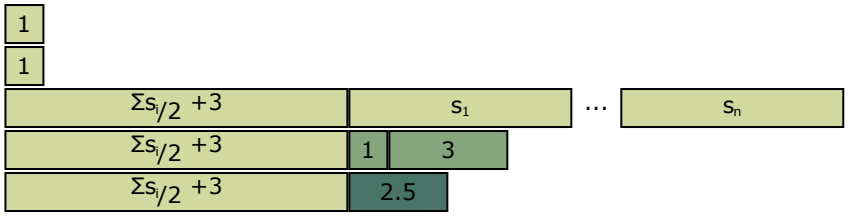
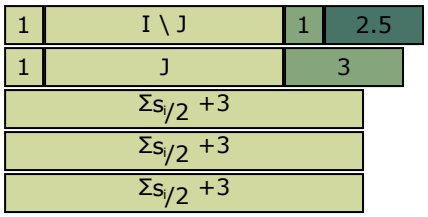


Figure 5.2: Instance of the MOSP problem used in the reduction.

We now prove that the PARTITION problem does have a solution if and only if the constructed instance of MOSP **does not admit** a pure Nash equilibrium.

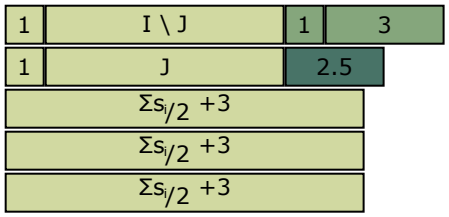
Assume that there exists a subset of indexes J that solves the PARTITION problem. Organizations $O^{(1)}$ and $O^{(2)}$ will always leave their jobs of length 1 on their own organizations due to the MOSP local constraint of not increasing the local makespan.

Jobs originally from organization $O^{(3)}$ have higher priority. Since the PARTITION problem admits a solution, the best response for $O^{(3)}$ is to equally divide jobs $J_2^{(3)},\ldots,J_{n+1}^{(3)}$ among the machines of organizations $O^{(1)}$ and $O^{(2)}$, and schedule its



(a) $C_{\max}^{(4)} = \sum s_i/2 + 4$;
$C_{\max}^{(5)} = \sum s_i/2 + 4.5$

(b) $C_{\max}^{(4)} = \sum s_i/2 + 6.5$;
$C_{\max}^{(5)} = \sum s_i/2 + 3.5$



(c) $C_{\max}^{(4)} = \sum s_i/2 + 5$;
$C_{\max}^{(5)} = \sum s_i/2 + 3.5$

Figure 5.3: Possible configurations for jobs $J_2^{(4)}$, $J_3^{(4)}$, and $J_2^{(5)}$, showing that one organization is always able to improve its local $C_{\max}$.

largest job $J_1^{(3)}$ on its own machine. The smaller jobs will be scheduled after jobs $J_1^{(1)}$ and $J_1^{(2)}$ (both of length 1), but before the jobs from any other organization.

Organizations $O^{(4)}$ and $O^{(5)}$ must also schedule their largest jobs ($J_1^{(3)}$, $J_1^{(4)}$, and $J_1^{(5)}$) on their own machines because of MOSP local constraint. However, they are free to change their strategies for jobs $J_2^{(4)}$, $J_3^{(4)}$, and $J_2^{(5)}$ (of lengths 1, 3, and 2.5).

Figure 5.3 shows that, no matter the strategy chosen, one of the organizations can always improve its local makespan, showing that a pure Nash equilibrium cannot be achieved for this game. In the figure, organization $O^{(5)}$ can improve its $C_{\max}$ by changing its strategy from the one in Figure 5.3a to the one in Figure 5.3b. After that, $O^{(4)}$ can improve its $C_{\max}$ by changing from the strategy 5.3b to 5.3c. Finally, we have a loop when organization $O^{(4)}$ changes its strategy to the one depicted in Figure 5.3a.

Conversely, assume that this instance of the MOSP game does not admit a pure Nash equilibrium. We want to show that in this case the PARTITION problem has a solution.

Suppose for contradiction that such solution for the PARTITION problem does not

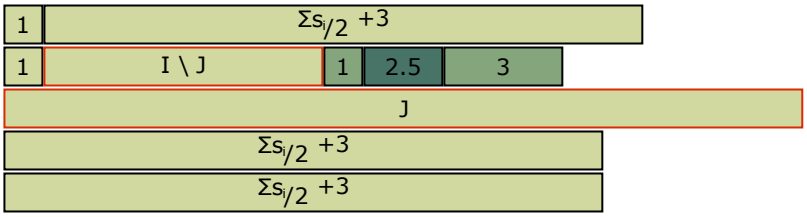


Figure 5.4: Equilibrium configuration if PARTITION does not have a solution.

exist, i.e., $\forall J \subseteq I = \{1,\ldots,n\}$, $\sum_{i\in J} s_i \neq \sum_{i\in I\setminus J} s_i$.

Let J be the set of indexes that minimizes the value of α in the expression $\sum_{i\in J} s_i = \alpha + \sum_{i\in I\setminus J} s_i$, with $\alpha > 0$.

Take the strategy depicted in Figure 5.4. Organizations $O^{(1)}$ and $O^{(2)}$ schedule their own jobs in their own machines. Organizations $O^{(4)}$ and $O^{(5)}$ schedule their large jobs ($J_1^{(4)}$ and $J_1^{(5)}$) also in their own machines.

Organization $O^{(3)}$ schedules its large job ($J_1^{(3)}$) on the machine of organization $O^{(1)}$. The remaining jobs are scheduled according to the set J that minimizes α. If $i \in J$, then job $J_{i+1}^{(3)}$ remains scheduled in organization $O^{(3)}$, otherwise the job is scheduled in the machine of organization $O^{(2)}$.

Finally, jobs $J_2^{(4)}$, $J_2^{(5)}$, and $J_3^{(4)}$ are scheduled in this (non-decreasing) order on organization $O^{(2)}$.

Organizations $O^{(1)}$ and $O^{(2)}$ have makespan equal to 1 and, therefore, have no incentive to change their strategy. $O^{(3)}$ also does not have incentive to change its strategy. The organization has a makespan equal to the load of the machine on organization $O^{(3)}$: $\sum_{i\in J_i} s_i$. Moving any job from this machine to organization $O^{(2)}$ or $O^{(1)}$ will increase the local makespan to at least $\sum_{i\in J_i} s_i + 1$. Jobs $J_2^{(4)}$, $J_2^{(5)}$, and $J_3^{(4)}$ finish before $\sum \frac{s_i}{2} + 3$, therefore neither organization $O^{(4)}$ nor $O^{(5)}$ has incentive to migrate any of these jobs to organization $O^{(1)}$ or to its own organization.

Since all organizations have no incentive to change their strategy, this configuration is a pure Nash equilibrium, which contradicts the assumption that this instance of the MOSP game does not admit a pure Nash equilibrium. This concludes the proof of the theorem. □

The results described in this section show that if the MOSP game is modeled with

a coordination mechanism based on priority on jobs, then not only ϵ-approximate equilibria with $\epsilon < 2$ may not exist for some instances, but also it is hard to know in advance if an instance admits at all a pure Nash equilibrium.

5.3 Game with priority to organizations

Section 5.2 showed that when priorities are given individually to jobs, some instances may not have pure Nash equilibria, and even the existence of the "weaker" ϵ-approximate equilibria is bounded to values of $\epsilon \geq 2$.

In this section, we study an alternative model for the assignment of priorities, where entire organizations are individually assigned with different scheduling priorities. In this model, each organization will locally schedule first its own jobs (the "my jobs first" policy) and then schedule the remaining jobs in non-increasing order of its owner priority. Two or more jobs belonging to a same organization are scheduled in no particular order.

Based on this game model, we present a parametric algorithm that computes a ϵ-approximate equilibrium for an instance of the MOSP problem. Given a ρ-approximation list scheduling algorithm, the algorithm computes a schedule that is a ρ-approximate equilibrium.

We also study the impact on the quality of the global $C_{\max}$ obtained by the selfish organizations modeled by our game. We show that the price of anarchy is asymptotically bounded by 2.

An algorithm to construct a pure ρ-approximate equilibrium

In this section, we present a parametric algorithm that constructs a pure ϵ-approximate equilibrium for instances of the MOSP game with one processor per organization and that has priority given to organizations. We show that using a ρ-approximation[2] list scheduling algorithm for the classical $P \,||\, C_{\max}$ scheduling problem, we can always construct a pure ρ-approximate equilibrium configuration for the MOSP game.

[2]The factor ρ of the algorithm used must have been analyzed for cases where machines starting times may be different from 0 like, for instance, Lee's MLPT [Guo98] (Lee's MLPT provides a 4/3-approximation, while standard LPT provides a 3/2-approximation). Any standard list scheduling algorithm have a guaranteed approximation ratio of 2.

The algorithm is an adaptation of the Interactive Load Balacing Algorithm (ILBA) presented in Section 3.3. The idea of this adapted algorithm is to rebalance the load of the organizations from the organizations with higher priority to the organizations with lower priority.

The construction works as follows. First, each organization schedules all its own jobs locally using the given ρ-approximation scheduling algorithm. The organizations are then enumerated by non-increasing priority (i.e., for any two organizations $O^{(i)}$ and $O^{(j)}$, $i < j$ if and only if $O^{(i)}$ has higher priority than $O^{(j)}$).

All organizations $O^{(1)}, \ldots, O^{(N)}$ are then rebalanced, one after the other, according to the order defined by the enumeration. The rebalancing of an organization $O^{(k)}$ is done by first unscheduling all jobs belonging to organization $O^{(k)}$, and then rescheduling all of them by executing the ρ-approximation list scheduling algorithm on all available processors on the platform. The algorithm must reassign a maximal set of jobs to the original organization, i.e., if at a given time the algorithm can choose between different organizations to schedule the job, then the original organization must be chosen.

At iteration k, the ρ-approximation algorithm will calculate a strategy for $O^{(k)}$'s jobs with cost no more than ρ times the *optimal* solution. In particular, this means that: $\forall s \in S$ and $\forall s_k \in S_k$, $c_k(s_k, s_{-k}) \leq \rho \cdot c_k(s)$, i.e., $O^{(k)}$ cannot improve its cost by a factor more than ρ.

We now show that:

Theorem 5.6. *The algorithm presented in this section always produces a pure ρ-approximate equilibrium configuration for the* MOSP *game when priorities are given to organizations.*

Proof. We prove the theorem by induction on k (the index of the organization being rescheduled by the algorithm). We show that after rescheduling the jobs of organization $O^{(k)}$, no organization $O^{(i)}$ with $i \leq k$ can unilaterally improve its cost by a factor greater than ρ.

For $k = 1$, the algorithm will rebalance all jobs of organization $O^{(1)}$ applying the ρ-approximation algorithm. Even if $O^{(1)}$ has the highest priority, the "my jobs first" constraint does not allow any further unilateral improvement.

Now assume that all organizations $1 \leq k \leq j-1$ cannot unilaterally improve their cost by a factor greater than ρ. We will show that after rescheduling the jobs from organization $O^{(j)}$, all already rescheduled organizations will not be able to improve their costs by a factor greater than ρ.

By contradiction, assume that an organization $O^{(i)}$ with $i < j$ can improve its makespan by a factor greater than ρ by changing only its own strategy.

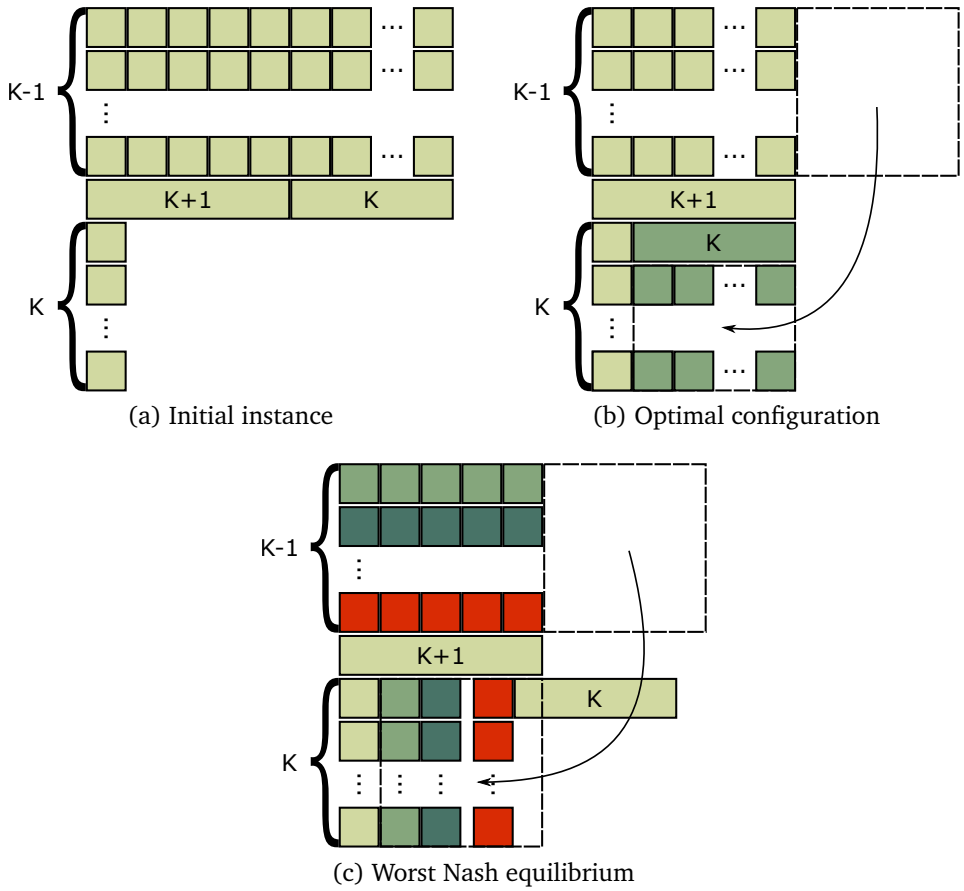
First, note that $O^{(i)}$ must have its makespan improved by migrating some of its jobs to organization $O^{(j)}$; otherwise, since $O^{(i)}$ has higher priority, it would have done that on some earlier interaction, contradicting the inductive hypothesis. So we can assume that at least one job from $O^{(i)}$ was migrated to $O^{(j)}$.

After rescheduling its jobs, if $O^{(j)}$ cannot improve its local makespan by migrating any job, then its jobs remain in its own organization. This means that any job from $O^{(i)}$ will start after the last job from $O^{(j)}$ because of the "my jobs first" constraint, which means that $O^{(i)}$ could have done that on an earlier iteration, which contradicts the inductive hypotheses.

On the other hand, if $O^{(j)}$ was able to improve its own makespan, then some of its jobs were migrated to other organizations. In particular, at least one of these jobs must have been rescheduled on a processor of a different organization to start either earlier or at the same time of the new value for the $C^{(j)}_{\max}$; otherwise this job would have been scheduled before at time $C^{(j)}_{\max}$. Let P be this processor. If $O^{(i)}$ can improve its makespan by migrating one of its jobs to start at time at most $C^{(j)}_{\max}$ on a processor in organization $O^{(j)}$, then it could have improved even more if the job was migrated to processor P in an earlier iteration, contradicting the inductive hypothesis that $O^{(i)}$ cannot improve its cost by a factor greater than ρ.

These two contradictions show that $O^{(i)}$ with $i < j$ cannot improve its makespan by a factor greater than ρ by changing only its own strategy. This fact finishes the proof of the inductive step, showing that after rescheduling $O^{(j)}$, no already rescheduled organizations will be able to improve their cost by a factor more than ρ. □

Note that if the ρ-approximation algorithm given as parameter is the optimal, all organizations $O^{(k)}$ have no incentive to change their strategies. In other words:



(a) Initial instance (b) Optimal configuration

(c) Worst Nash equilibrium

Figure 5.5: Price of anarchy of the MOSP game with priority given to organizations.

Corollary 5.7. *The* MOSP *game with priority given to organizations always induces a pure Nash equilibrium.*

The results presented in Chapter 3 indicate that computing the best response for each agent is NP-complete and, therefore, computing pure Nash equilibrium for the MOSP game remains a difficult problem.

Price of anarchy

The interest in searching configurations that result in equilibria — both pure Nash equilibria or ϵ-approximate equilibria — is a consequence of the selfish behavior of the agents. If on the one hand it guarantees that all agents are satisfied with the

result, on the other hand, uncoordinated, selfish behavior can potentially lead to suboptimal social outcomes (in our case, to suboptimal results for the social cost, i.e., the global $C_{\max}$ of the system).

We measure this effect by studying the *price of anarchy* of our game. The price of anarchy is defined as the ratio between the worst objective function value of an equilibrium of a game and the one of an optimal outcome.

Theorem 5.8. *The price of anarchy (PoA) of* MOSP *games with priorities given to organizations is lower or equal to* $2 - \frac{1}{N}$*, and this bound is asymptotically tight.*

Proof. The upper bound of the price of anarchy follows straightforwardly from the fact that a Nash equilibrium has no idle time. Consequently, the Nash equilibrium can be seen as a solution computed by an arbitrary list scheduling algorithm. By Theorem 2.2, an arbitrary list scheduling algorithm is a $\left(2 - \frac{1}{m}\right)$-approximation, where m corresponds to the number of machines.

We now show that this bound is asymptotically tight. Take the instance of the MOSP game (with priority given to organizations) depicted in Figure 5.5a. This game has $K - 1$ organizations having $2K + 1$ jobs of size 1, one organization having two jobs of size $K + 1$ and K, and K organizations having one job of size 1. In this game, an organization $O^{(i)}$ has higher priority over an organization $O^{(j)}$ if $i < j$.

Figure 5.5b shows the optimal configuration (profile) possible. Jobs originally starting after time $K + 1$ (highlighted in the figure) are migrated to the last K organizations. On this configuration, every organization (except the last K ones) achieves a local makespan equal to $K + 1$.

The worst Nash equilibrium possible is the configuration depicted in Figure 5.5c. In this configuration, jobs starting after time $K+1$ from organizations $O^{(1)}, \ldots, O^{(K-1)}$ are rescheduled on the last K organizations in such way that a job migrating from organization $O^{(i)}$ is rescheduled at time $i + 1$ on these machines. Different job colors on Figure 5.5c indicate the ownership of the jobs from organizations $O^{(1)}, \ldots, O^{(K-1)}$. The worst local makespan is given by organization $O^{(k)}$ and is equal to $1+(K-1)+K = 2K$.

This instance shows that the price of anarchy for the MOSP game is equal to $\frac{2K}{K+1}$, which for large values of K is asymptotically equal to 2. □

At first glance, assigning different priorities to organizations seems unfair. Even if MOSP local constraints are always respected for all organizations, an organization may never achieve its optimal value for the local makespan because of the chosen priorities. However, for the workloads considered in this work (composed of batches of *bags-of-tasks* jobs) fairness can be achieved in practice by rotating the priorities of each organization at each batch scheduling.

5.4 Concluding remarks

In this chapter we presented a game theoretic analysis of the Multi-Organization Scheduling Problem. We studied how selfish organizations can individually choose the best strategy for their jobs and still achieve a scheduling where all the organizations have incentive to cooperate with each other.

We proposed a model where each agent is responsible for all jobs belonging to a specific organization. An agent is free to choose in which organization each of its jobs will be executed. Unlike some previous works on selfish load balancing games, the algorithm used locally by each organization to schedule the jobs assigned to it is crucial in the final result obtained by each agent. In our proposed model, each selfish organization will first prioritize its own jobs. The remaining (foreign) jobs must be scheduled according to some given priority.

We showed that when this priority is given individually to the jobs — like in classical scheduling algorithms such as LPT or SPT — no pure ϵ-approximate equilibrium is possible for values of ϵ less than 2. We also proved that with this priority policy, the decision problem of deciding whether a given instance admits a pure Nash equilibrium is co-NP hard.

When priority is given to the organizations, however, we can show that the MOSP game can always converge to an equilibrium state. We provided a parametric algorithm that, given any ρ-approximation list scheduling algorithm, computes a pure ρ-approximate equilibrium for instances of the MOSP problem with one processor per organization — a first step towards a more general case. We also showed that with this priority policy, the price of anarchy — the ratio between the social cost (the global $C_{\max}$) of a worst-case Nash equilibrium and the social cost of an optimal assignment — is asymptotically equal to 2.

CHAPTER 6

The Multi-Users Scheduling Problem

SOMETIMES users must team-up to share computational resources that do not necessarily belong to them individually, but to a community of users. In such cases, the users do not have special privileges over some of the shared resources, which differs from the MOSP problems studied in the previous chapters. Still, they must collaborate with each other and coordinate the use of these resources in order to achieve high performances in the most fair way possible.

One of the examples most closely related with the MOSP problem is the problem of resource sharing in multi-users environments, like, for instance, in the context of cluster computing [KC03]. Several users must execute their jobs in the processors of the cluster, but none has special control over these processors.

An intricate problem that arises in these environments is how to schedule all the jobs, from all the users, while providing performance and fairness guarantees. More than just providing a guarantee, the scheduler should respect each user own interest, i.e., the scheduler should optimize the performance objective intrinsic of each user. To the best of our knowledge, this problem has not been completely solved yet, specially under the theoretical perspective.

These performance and fairness issues appear in other real world problems as well. Some examples are problems like bandwidth sharing in networking [MR02], negotiation between co-owners of a factory [Agn+00], application run-time with

hypergraph partitioning (partition balance and network usage) [PM07], etc.

Such optimization problems involve multiple complex parameters. To obtain a tractable way of solving the problem, one must often consider the optimization of a simpler objective. Sometimes, when optimizing one objective is not enough, multi-objective optimization is used to take into account multiple parameters of interest.

In this chapter, we present a generic method to optimize monotonic objective functions by approximating the Pareto set of the multi-objective problem where each parameter is an objective function. We exemplify our technique on a multi-user scheduling problem by optimizing the fairness among users.

The remaining of the chapter is organized as follows. In Section 6.1 we present the Multi-Users Scheduling Problem, its notations, and the previously known results that serve as basis of the work presented in this chapter. Section 6.2 shows how to construct Pareto approximations for some specific setups of the MUSP problem. A technique to optimize monotonic objective functions on multi-objective problems is presented in Section 6.3, together with a detailed description of how to apply this technique to the MUSP problem. Finally, Section 6.4 presents our concluding remarks.

6.1 Problem description and notations

The Multi-Users Scheduling Problem

We present in this section the approximation of multi-objective problems using the scheduling problem known in the literature as the *Multi-Users Scheduling Problem* (MUSP) [ST09].

MUSP models the scheduling problem that appears on computational platforms where different users, having different performance objectives, compete for the available resources in order to individually optimize their own performance. In such platforms, each user is interested in a particular performance objective for his/her jobs. Although each user has his/her own objective and set of jobs, the final schedule is devised by an authoritative centralized scheduler. The goal of the centralized scheduler is to serve the user interests as best as possible: it is never in the system

interest to degrade the performance of a user without improving the performance of another one. In other words, the goal of the scheduler can be modeled by a function which is monotonic in the users' interest.

We focus this study in the case where each user is interested in minimizing either the average completion time or the maximum completion time of his/her jobs. We assume the scheduling problem is offline: all the jobs are available at the beginning of the execution and the running time of the jobs are known in advance.

We adopt the same notation used in the previous chapter for the MOSP problem. In the Multi-Users Scheduling Problem we assume a parallel platform composed of m identical processors, shared by a total number of k users. Each user u aims at executing a set of $n^{(u)}$ jobs, where each job is denoted by $J_i^{(u)}$, $1 \leq i \leq n^{(u)}$. The processing time of $J_i^{(u)}$ is $p_i^{(u)}$. A user u is interested in minimizing some objective function denoted by $f^{(u)}$. The total number of jobs on the platform is $n = \sum_{u=1}^{k} n^{(u)}$.

A centralized scheduler will determine the starting time $\sigma(J_i^{(u)})$ of each job, that in turn will be executed without interruption until its completion at time $C_i^{(u)} = \sigma(J_i^{(u)}) + p_i^{(u)}$.

The makespan of user u, i.e., the maximum completion time of a job owned by this user, is defined as $C_{\max}^{(u)} = \max_{i=1}^{n^{(u)}} C_i^{(u)}$. Likewise, the average completion time of the jobs belonging to user u is defined as $\sum C_i^{(u)} = \frac{1}{n^{(u)}} \sum_{i=1}^{n^{(u)}} p_i^{(u)}$. We assume the function $f^{(u)}$ is either $C_{\max}^{(u)}$ or $\sum C_i^{(u)}$.

In line with MOSP, the Multi-Users Scheduling Problem with k' users interested in sum of completion time and k'' users interested in the makespan, with $k' + k'' = k$, is denoted by MUSP($k' : \sum C_i$; $k'' : C_{\max}$).

Problem definition

Using different setups for the MUSP problem, we study how to schedule all the jobs from the different users in order to simultaneously minimize the set of objective functions $f^{(1)}, f^{(2)}, \ldots, f^{(k)}$. The problem is defined as a multi-objective optimization problem. Potentially, all the solutions of the Pareto set are suitable. Recall that the Pareto set (Definition 2.6, p. 17) is the set of Pareto optimal solutions, i.e., solutions where no objective value can be improved without degrading another objective value.

Yet, the scheduler needs to select a unique schedule. Monotonic functions are

a wide class of functions that the scheduler might be interested in optimizing. Numerous fairness functions belong to this class. The main idea of the method we propose is to optimize a monotonic objective function by approximating the Pareto set of the multi-objective MUSP problem.

Several fairness functions of interest have been used in the literature [LT07]. To measure the fairness of a set of values $x = (x_1, x_2, \ldots, x_k)$, popular fairness functions are:

- Arithmetic mean $= \frac{1}{k}\sum_{i=1}^{k} x_i$;
- Geometric mean $= (\Pi_{i=1}^{k} x_i)^{1/k}$ (also known as proportional fairness);
- Jain index $= \frac{(\sum_{i=1}^{k} x_i)^2}{k\sum_{i=1}^{k} x_i^2}$;
- L^p norm, defined by $\|x\|_p = (\sum_{i=1}^{k} |x_i|^p)^{1/p}$, for any $p \geq 1$;
- Min-Max ratio $= \frac{\min_i\{x_i\}}{\max_j\{x_j\}} = \min_{i,j}\left\{\frac{x_i}{x_j}\right\}$;
- Variance $= \frac{1}{k-1}\sum_{i=1}^{k}(x_i - \mu)^2$, where $\mu = \frac{1}{n}\sum_{i=1}^{k} x_i$;

Note, however, that although popular fairness functions like the Jain index — used in our analysis on Section 3.4 — may present some interesting properties like independence on the scale and on the size of the values, continuity, etc., they may not be monotonic. The Variance, for instance, is optimized by making all the objective values equal, potentially worsening the performance of one user without improving the performance of another one. Min-Max is not monotonic as well. In the following, when discussing fairness functions, we mean a monotonic fairness function such as the L^p norm, the Arithmetic mean or the Geometric mean.

There are many more monotonic functions that a scheduler might be interested in optimizing. For instance, the users' objectives could be normalized with respect to a given configuration such as a default scheduling policy or a best case if the user was alone in the system. The scheduler could also apply some weight to make some users more important. Finally, financial incentive could also be modeled where the user pays a different amount of money to the system depending on the quality of service it receives. For example, $100 to have jobs completed before a conference

deadline, $10 if the jobs are completed in the day of the deadline, and $0 if the deadline is passed.

Previously known results

The Multi-Users Scheduling Problem was first studied on a single processor with two users by Agnetis et al. [Agn+04] and on multiple processors by Saule and Trystram [ST09].

When all users are interested in minimizing their *makespan*, MUSP cannot be approximated within a ratio better than $(1,2,\ldots,k)$ [ST09]. For this case, Saule and Trystram [ST09] presented an algorithm (called MULTICMAX) that was proven to be a $(\rho,2\rho,\ldots,k\rho)$-approximation of the zenith[1], where ρ is the approximation ratio of an arbitrary algorithm that minimizes the makespan for the single-user case. They also presented an adaptation of the MULTICMAX that provides a constant approximation of a Pareto optimal solution. Agnetis et al. [Agn+04] provided a $<\bar{1},\bar{1}>$-approximation of the problem for two users on a single processor. This result will be recalled and extended in Section 6.2.

If all users are interested in minimizing the *average completion time* of their jobs, then it is known that MUSP cannot be approximated from the zenith within a ratio better than $(k,\ldots,k)$ [ST09]. The problem has been shown to be weakly NP-complete on a single processor with two users and an exponential dynamic programming algorithm has been provided [Agn+04]. The MULTISUM algorithm is a $(k,\ldots,k)$-approximation of the zenith of the problem [ST09].

Both MULTICMAX and MULTISUM are optimal in the sense that no algorithm can achieve better approximation ratios to the zenith solution. Although tight, the "high" ratios guaranteed by the algorithms and their dependence on the number of users are explained by the fact that these are approximations from the zenith point, computed using the values for the objective function that each user could have obtained if it was alone on the system.

For the mixed case — MUSP($k' : \sum C_i$; $k'' : C_{\max}$)— the MULTIMIXED algorithm provides a $(k,\ldots,k)$-approximation of the zenith for the k' users interested in the

[1] Recall from Section 2.2 that the zenith point is the point where each objective has the minimum possible value and the nadir point, analogously, is the point where each objective has its maximum value.

average completion time, and a $\left(\frac{k}{k''}\rho, \frac{2k}{k''}\rho, \ldots, k\rho\right)$-approximation of the zenith for the k'' users interested in the makespan [ST09].

The problem is computationally hard even when few users are using the system. In the simplest case, where only one user ($k = 1$) wants to minimize his/her makespan using all the m available machines, the problem is equivalent to the classical scheduling problem known as $P \,||\, C_{\max}$. In this case, all the classical results presented in Section 2.1 hold.

Pareto set approximation algorithm

In classical single-objective scheduling problems, we say that a schedule S is a ρ-approximation [Hoc97] if the value of the objective function obtained with this schedule $f(S)$ is at most ρ times the optimal value f^* for this objective, i.e., $f(S) \leq \rho f^*$.

Generally, two approaches are used to approximate multi-objective problems like MUSP. The first one consists in approximating the zenith point (point usually not feasible, but that gives lower bounds for the values of each objective function). In this case, we say that a solution S is a $(\rho_1, \ldots, \rho_k)$-approximation of the zenith if the value of the objective function obtained by each user u in the schedule S is such that $f^{(u)}(S) \leq \rho_u f^{(u)*}$.

Many problems do not admit a constant zenith approximation algorithm. Actually, there might be no solution at a constant relative distance of the zenith [GST09]. In these cases, the other approach is to find an approximation of the Pareto set. Informally, a set of solutions P is a ρ-approximation of the Pareto set P^* if each point $p^* \in P^*$ is ρ-approximated by at least a point $p \in P$. Formally:

Definition 6.1 (from [PY00])**.** A set of solutions P is a $(\rho_1, \ldots, \rho_k)$-*approximation of the Pareto set* P^* if $\forall S^* \in P^*, \exists S \in P$, we have for all users u: $f^{(u)}(S) \leq \rho_u f^{(u)}(S^*)$.

The most popular way to obtain a Pareto set approximation algorithm is to provide an approximation algorithm for the variant of the problem where some of the objectives are constrained by some thresholds, while some others have to be optimized. This concept is somehow similar to the dual approximation algorithms [HS88]. The formal approximation property is stated as follows:

Definition 6.2 (from [JST08]). Given $\omega = (\omega_{x+1}, \ldots, \omega_k)$ thresholds for the values of the objective functions $f^{(x+1)}, \ldots, f^{(k)}$, a $<\rho_1, \rho_2, \ldots, \rho_x, \overline{\rho_{x+1}}, \overline{\rho_{x+2}}, \ldots, \overline{\rho_k}>$-approximation algorithm delivers a solution where $f^{(u)} \leq \rho_u \omega_u$, for $x+1 \leq u \leq k$, and $f^{(u)} \leq \rho_u f^{(u)*,\omega}$, for $1 \leq u \leq x$, where $f^{(u)*,\omega}$ is the best value possible for the objective function given that the thresholds ω were respected. The algorithm can return no solution if no solution that respects the constraint ω exists.

Let us assume a lower bound $f^{(u)}_{\min}$ and an upper bound $f^{(u)}_{\max}$ on the values of the objective function of the solutions of the Pareto set are known. A simple lower bound on the mono objective optimization problem is enough to obtain $f^{(u)}_{\min}$. The upper bound $f^{(u)}_{\max}$ is often obtained in scheduling by assuming all the jobs complete at a worst case time without introducing unnecessary idle times. In MUSP, this upper bound can be obtained by scheduling the tasks of user u after all the other tasks. Notice that $(f^{(1)}_{\min}, f^{(2)}_{\min}, \ldots, f^{(k)}_{\min})$ is Pareto dominated by the zenith. While $(f^{(1)}_{\max}, f^{(2)}_{\max}, \ldots, f^{(k)}_{\max})$ is Pareto dominated by the nadir.

Papadimitriou and Yannakakis [PY00] proposed the following algorithm to find an approximation of the Pareto set. First, each interval $[f^{(u)}_{\min}, f^{(u)}_{\max}]$ is partitioned in intervals of geometrically increasing length[2]:
$[f^{(u)}_{\min}, (1+\epsilon)f^{(u)}_{\min}] \cup [(1+\epsilon)f^{(u)}_{\min}, (1+\epsilon)^2 f^{(u)}_{\min}] \cup [(1+\epsilon)^2 f^{(u)}_{\min}, (1+\epsilon)^3 f^{(u)}_{\min}] \cup \ldots \cup [(1+\epsilon)^{\left\lfloor \log_{1+\epsilon} \frac{f^{(u)}_{max}}{f^{(u)}_{min}} \right\rfloor} f^{(u)}_{min}; f^{(u)}_{max}]$.

This decomposition of each interval decomposes the hyperbox delimited by the zenith and the nadir in $\prod_{u=1}^{k} \left(1 + \left\lfloor \log_{1+\epsilon} \frac{f^{(u)}_{max}}{f^{(u)}_{min}} \right\rfloor\right)$ hyperboxes. An interesting property of this decomposition is that the ratio between the coordinates of the nadir and the zenith of each hyperbox is $(1+\epsilon)$. Since, informally, the ϵ-approximate Pareto set is a set of solutions which are not dominated by any other by a ratio of more than $1+\epsilon$, taking one solution from each hyperbox (if such a solution exists) builds an ϵ-approximation of the Pareto set [PY00]. For a fixed number of objective k and for most optimization problems, this approximation of the Pareto set is of polynomial size. An example of such decomposition is presented in Figure 6.1.

However, taking a solution inside each hyperbox might not be algorithmically easy. If one has a $<\rho_1, \rho_2, \ldots, \rho_x, \overline{\rho_{x+1}}, \overline{\rho_{x+2}}, \ldots, \overline{\rho_k}>$-approximation algorithm A, then it

[2]This decomposition assumes $f^{(u)}_{\min}$ is not equal to 0. This assumption is common in the theory of approximation algorithms.

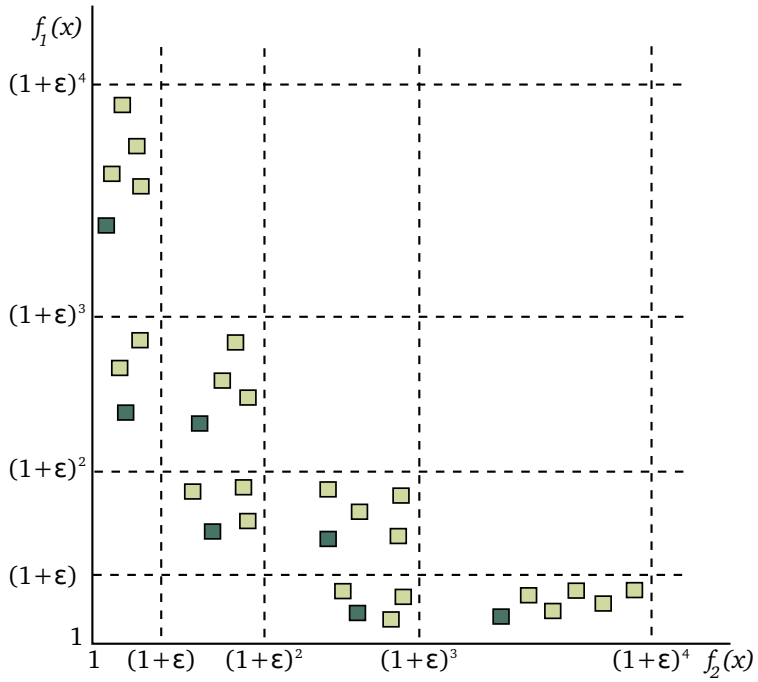


Figure 6.1: Example of an ϵ-approximate Pareto set computed using the algorithm proposed by Papadimitriou and Yannakakis [PY00]. The darker points represent a possible set of points that can be chosen by the algorithm.

is possible to compute a $(\rho_1, \ldots, \rho_k)$-approximation of the nadir of the hyperboxes that contain solutions. Since the nadir is a $(1+\epsilon)$-approximation of the zenith of the hyperbox, A returns a $((1+\epsilon)\rho_1, \ldots, (1+\epsilon)\rho_k)$-approximation of the zenith. And the set of solutions returned by the procedure is a $((1+\epsilon)\rho_1, \ldots, (1+\epsilon)\rho_k)$-approximation of the Pareto set. Partitioning the solution space among all the objectives is not necessary, it is sufficient to decompose the space among the objectives $x+1, \ldots, k$ that are constrained. This technique was used in [JST08].

6.2 Pareto approximation of MUSP

In this section, we study the Pareto approximation of the MUSP problem using different setups for the problem. We start by giving some general properties of the set of valid MUSP solutions.

Proposition 6.3. *In a Pareto optimal schedule, all jobs from a user interested in minimizing his/her makespan that are assigned to a same processor are scheduled to be executed one after the other.*

Proof. Let us consider the case where user u is interested in minimizing his/her makespan and a user u' is interested in either makespan or average completion time. Suppose, for contradiction, a Pareto optimal schedule where there exists at least one job ($J^{(u')}$) scheduled between two jobs from user u ($J_1^{(u)}$ and $J_2^{(u)}$) in the same processor and that $C_{\max}^{(u)}$ is given by the completion time of job $J_2^{(u)}$. If we change the schedule and execute $J^{(u')}$ before $J_1^{(u)}$, then user u does not have his/her makespan changed, but user u' is able to improve his/her objective (whenever it is to minimize the makespan or average completion time), which contradicts the fact that the schedule was Pareto optimal. □

Proposition 6.4. *In a Pareto optimal solution, the jobs of a user interested in the sum of completion time are arranged in SPT order on a given processor.*

Proof. The proof is by contradiction. Suppose a Pareto optimal schedule where a user u, interested in the average completion time, has two jobs $J_i^{(u)}$ and $J_j^{(u)}$ — with $p_i^{(u)} < p_j^{(u)}$ — scheduled on the same processor. Suppose also that these jobs are not scheduled in SPT order, i.e., $J_j^{(u)}$ is scheduled in an earlier time than $J_i^{(u)}$. It is easy to see that by switching jobs $J_j^{(u)}$ and $J_i^{(u)}$ the average completion time of user u will decrease (by the optimality of SPT on the scheduling problem $1 \,||\, \sum C_i$, see Theorem 2.1, p. 12) and, since $p_i^{(u)} < p_j^{(u)}$, all jobs (from any user) originally scheduled between $J_j^{(u)}$ and $J_i^{(u)}$ will now start at an earlier time, which means that those users will also have their objective values improved, no matter if the objective value is makespan or average completion time. This means that all users were able to improve their objective values, which contradicts the fact that the original schedule was Pareto optimal. □

The core of the scheduling algorithm is to set some constraint on the objective of some user while optimizing the other one. Setting up a constraint on a user interested in the makespan is equivalent to trying to complete all the tasks of that user before a given time. This relates to the problem of optimizing deadlines, which are well-studied in the literature. However, setting a constraint on a user interested

in the sum of completion time has a less intuitive practical meaning. Therefore, all the presented techniques rely on setting a constraint on the users interested in optimizing their makespan.

In the following we will present some examples of approximation algorithms for specific setups of the MUSP problem that will be used in conjunction with the algorithm presented in Section 6.3.

MUSP($k : C_{\max}$) on $m = 1$ processor

Proposition 6.5. *The Earliest Deadline First (EDF) algorithm is a* $<\bar{1}, \bar{1}, \ldots, \bar{1}>$*-approximation algorithm for the problem* MUSP($k : C_{\max}$) *on* $m = 1$ *processor and, therefore, a* $(1+\epsilon, 1+\epsilon, \ldots, 1+\epsilon)$*-approximation of the Pareto set.*

Proof. Restrictions on the maximum completion time of each user interested in minimizing his/her $C_{\max}$ are equivalent to setting deadlines for the jobs of this user. The Earliest Deadline First algorithm [GRS96] — a list scheduling algorithm that always chooses the next available task with the earliest deadline to be executed — is known to be optimal with respect of the deadlines for $m = 1$ processors. If there is a feasible solution that respects all the thresholds, then EDF will produce a $<\bar{1}, \bar{1}, \ldots, \bar{1}>$-approximation and, therefore a $(1+\epsilon, 1+\epsilon, \ldots, 1+\epsilon)$-approximation of the Pareto set. □

Remark that Proposition 6.3 shows that, in practice, we can efficiently implement the algorithm by considering all jobs from a particular user u as one large job with length $\sum_{i=1}^{n^{(u)}} p_i^{(u)}$.

MUSP($1 : \sum C_i; k-1 : C_{\max}$) on $m = 1$ processor

Agnetis et al. [Agn+04] were the first to study this sub-case of the problem limited to the case where $k' = k'' = 1$. We show with a slightly different argument that:

Proposition 6.6. *The Latest Starting Time (LST) algorithm is a* $<1, \bar{1}, \ldots, \bar{1}>$*-approximation algorithm for the problem* MUSP($1 : \sum C_i; k-1 : C_{\max}$) *on* $m = 1$ *processor.*

Proof. The thresholds are given to the objectives of users interested in minimizing their makespan. Again, this is equivalent to set deadlines for these users. Proposi-

tion 6.3 also holds, so each user u interested in makespan can be treated as having only one large job of length $\sum_{i=1}^{n^{(u)}} p_i^{(u)}$.

The Latest Starting Time (LST) algorithm works as follows. First, the users interested in $C_{\max}$ have their large job scheduled at the latest moment possible before its deadline (which usually happens at time $d_i^{(u)} - p_i^{(u)}$, but could happen earlier if there is another job already scheduled in the interval $[d_i^{(u)} - p_i^{(u)}, d_i^{(u)}]$). Secondly, the jobs from the user interested in $\sum C_i$ is scheduled in SPT order at the first idle time where the job fits. Finally, the scheduled is compacted, shifting all jobs to start earlier in order to leave no idle times.

It is easy to show that the approximation ratios are guaranteed. Like in the analysis of Proposition 6.5, if all the deadlines can be met, then the $\bar{1}, \ldots, \bar{1}$ ratio of the constrained objectives is guaranteed for the users interested in $C_{\max}$. Proposition 6.4 and the optimality of SPT guarantee that scheduling the jobs in SPT order in the remaining idle times is the best that one can do while respecting the constraints given by the jobs from users interested in $C_{\max}$. Thus, the algorithm is a $<1, \bar{1}, \ldots, \bar{1}>$-approximation . □

MUSP($k : C_{\max}$) on $m = N$ processors

The extension of Proposition 6.5 to the case where we have a fixed number of processors is straightforward.

Theorem 6.7. *The EDF algorithm is a $<\bar{2}, \bar{2}, \ldots, \bar{2}>$-approximation and therefore leads to a $(2+\epsilon, 2+\epsilon, \ldots, 2+\epsilon)$-approximation of the Pareto set.*

Proof. The algorithm is very simple. Just apply the List Scheduling algorithm with the list of jobs sorted according to the EDF rule.

The proof of the ratio follows the reasoning for proving the approximation guarantee of List Scheduling algorithms, presented in the proof of Theorem 2.2 (p. 14). If a job ℓ starts after its deadline, it means that until time $\sigma(\ell)$ all processors were busy. Since there is no idle time, jobs with deadlines less than or equal to the deadline of job ℓ represent a total work of $N\sigma(\ell)$. Thus, if one job starts after the deadline, it means that it is not possible to meet all the deadlines of these jobs and

the schedule is not feasible. If it starts before, then the 2-approximation guaranteed by Theorem 2.2 holds. □

Pareto efficient solutions does not imply fairness. Any global optimization function can lead to solutions that can be perceived as unfair by the participants. In the next section, we discuss how to find fair solutions between the solutions presenting good trade-offs.

6.3 Optimizing fairness functions

In this section, we describe a novel general algorithm to optimize monotonic functions of multiple variables, where each variable is an objective function of a multi-objective problem. In this work, we consider the fairness function as the monotonic function of interest. The algorithm optimizes the fairness function by approximating the Pareto set of the multi-objective problem.

The main idea of this new algorithm is to perform a search in the k-dimensional objective space of the multi-objective problem (the algorithm is generic, but is explained with the MUSP($k' : \sum C_i; k'' : C_{\max}$) problem). We divide the objective space in several hyperrectangles and, guided by the value of the fairness function computed for the solutions inside each hyperrectangle, we either further refine the hyperrectangle and search for solutions with better fairness — that dominate the nadir point of the hyperrectangle — or just discard it if the fairness at the zenith point is worse than the best fairness found so far.

In the next sections will formally describe the algorithm and provide some examples of utilization.

Algorithm

Given an instance of the multi-objective problem, and assuming that there is a $<\rho_1, \ldots, \rho_x, \overline{\rho_{x+1}}, \ldots, \overline{\rho_k}>$-approximation algorithm for the problem, we start by calculating the points that will delimit the objective space of this instance to be searched: the zenith and the nadir points.

The zenith and the nadir points define a hyperrectangle that contains the entire attainable objective space of this instance. Notice that some problems have nadir

point at infinity. For instance, in a scheduling problem one can always delay some of the tasks indefinitely. We only need the nadir to be Pareto dominated by every Pareto optimal solution.

Starting with the hyperrectangle that covers the entire attainable objective space, the algorithm will iteratively decompose the objective space, searching for schedules of the multi-objective problem with better fairness.

At each iteration, for each hyperrectangle that was not discarded on the previous iterations, the algorithm computes the $<\rho_1, \ldots, \rho_x, \overline{\rho_{x+1}}, \ldots, \overline{\rho_k}>$-approximation algorithm *constrained by the uppermost point of the hyperrectangle*. The fairness function is applied to the point in the objective space corresponding to the result obtained by the approximation algorithm. The result with best fairness so far is saved.

The hyperrectangle is discarded if: (i) the approximation algorithm constrained by the uppermost point of the hyperrectangle does not produce an attainable result, or (ii) if the fairness of the result obtained at the lowermost point is worst than or equal to the best known fairness.

Otherwise, the non-discarded hyperrectangles are each decomposed in 2^k hyperrectangles, by cutting each dimension x (bounded by x_{low} and x_{high}) in the point x_{middle} such that $\frac{x_{\text{high}}}{x_{\text{middle}}} = \frac{x_{\text{middle}}}{x_{\text{low}}}$. It is the same decomposition proposed by Papadimitriou and Yannakakis [PY00], but with a different ratio between the zenith and the nadir, instead of fixed ratio of $(1+\epsilon)$ proposed by them.

Each iteration can potentially find a better value for the monotonic function been optimized and the number of iterations can be chosen at will.

Example

To exemplify how the previous algorithm can be applied to a concrete multi-objective problem, we present some simple simulations aiming to improve the fairness of some instances of the Multi-Users Scheduling Problem.

For the purpose of this example, we narrow the possible instances of the Multi-Users Scheduling Problem to the ones where k users — one interested in his/her average completion time and all others interested in minimizing their makespan — must share one single processor. We saw in Section 6.2 that for this case Propo-

sition 6.6 guarantees that the Latest Starting Time (LST) algorithm provides a $<1, \bar{1}, \ldots, \bar{1}>$-approximation algorithm for these instances.

The experiments were conceived to simulate a batch scheduler that intents to iteratively improve the fairness in a short amount of time, while the target system is being set up to receive and execute the jobs. The jobs and users used in our experiments were chosen to represent a workload of scientific computing. The basic fairness function chosen is the L^2 norm.

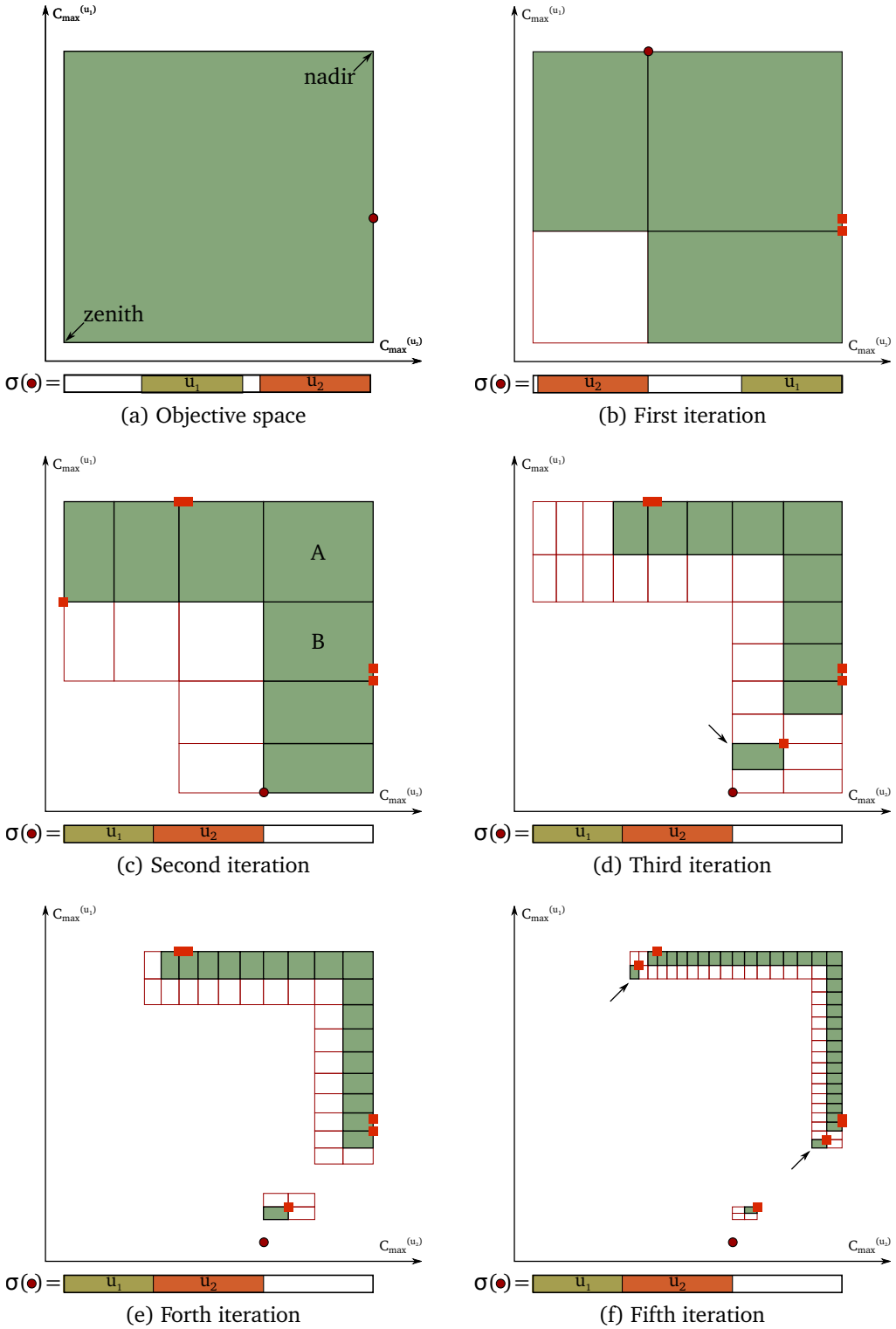
We used a public available workload of a Blue Gene/P system in production at the Argonne National Laboratory [Fei09]. From all the jobs available on the workload (jobs executed in the system from January 2009 to September 2009), we select the 10% largest jobs of each of the most active users.

All results presented represent the average of 30 executions of each experiment in a standard Intel Core i5 system with 5 GB of RAM and 8 MB of cache for a simulation period fixed in five minutes.

Figure 6.2 shows a graphical representation of the earlier stages of the execution of our algorithm. Each figure shows an iteration of the algorithm executed for an instance of the problem MUSP($1 : \sum C_i; 2 : C_{\max}$) and the respective schedule of the solution with best fairness found so far. Since $k'' = 2$ in this example, each hyperrectangle is a two-dimensional rectangle. The third dimension — the one that should represent the values obtained for the user interested in $\sum C_i$— is not explicitly represented in the figure since the Pareto approximation algorithm does not put thresholds on the possible values of $\sum C_i$, but it can be inferred from the values of the other two dimensions.

For this particular case, our search algorithm will use the uppermost point of each rectangle as different values of deadlines for the $k'' = 2$ users interested in $C_{\max}$. The search algorithm will look among these values for the ones that could lead to a solution with better fairness. Each of these different deadlines will be used by the LST algorithm to schedule these jobs at the latest moment possible and will schedule the jobs from the user wanting $\sum C_i$ in the remaining available times (in SPT order).

Figure 6.2 also shows the schedule found for the jobs of the users interested in $C_{\max}$ on the most fair solution found at each iteration. Since, in our example, all users will share a single processor, by Proposition 6.3 we have that the LST algorithm will treat each user interested in its makespan as if each user have only a single large



(a) Objective space (b) First iteration

(c) Second iteration (d) Third iteration

(e) Forth iteration (f) Fifth iteration

Figure 6.2: Decomposition of the search space during the first iterations of the algorithm and the respective schedule found with the best fairness so far.

job. The jobs belonging to the user wanting $\sum C_i$ are hidden in the figure (they fill the entire white area) for clarity.

At the first iteration of the algorithm, the search space is delimited by the nadir point (the uppermost point of the rectangle) and the zenith (the lowermost point). The algorithm starts by calculating the nadir and the zenith points for this instance of the problem. Recall that in MUSP, the nadir can be calculated by measuring the $C_{\max}$ of each user u if his/her tasks are scheduled after all the other tasks. Similarly, the zenith can be calculated by scheduling all the tasks before the others. Figure 6.2a depicts the rectangle that represents the entire search space.

The algorithm uses each coordinate of the nadir point as a constraint for a user interested in $C_{\max}$. The LST algorithm calculates an approximation from the Pareto set based on these constraints and the result (another point in the search space) is kept as the one having the best fairness so far. Figure 6.2 depicts the point with the best fairness found so far using the symbol • and its respective schedule $\sigma(\bullet)$.

Figure 6.2b shows the result of the next iteration. The original rectangle was divided in four rectangles. For each rectangle, the LST algorithm is applied taking the coordinates of the uppermost point of each rectangle as constraints. The point obtained with each rectangle is represented by the symbol ■. The darker rectangles was kept for the next iteration, while the white one was discarded because the fairness obtained using the lowermost point of this rectangle as constraints to the LST algorithm was worst than the best known fairness.

During the second iteration (Figure 6.2c) we can see for the first time rectangles where the point calculated by the LST lies outside the rectangle. The LST calculated the same point for both rectangles A and B. Since the fairness obtained using the LST taking the lowermost point as constraints is not worst than or equal to the best known fairness, rectangle A could not be discarded yet.

Finally, figures 6.2d and 6.2f show that the refinements of the rectangles lead to the discovery of new points of interest. The rectangles highlighted on these figures show the new approximations of the Pareto set found. We finish the example at the fifth iteration, but one could iterate more in order to find more points.

Our experiments showed an improvement of 27.69% on the fairness of the best point found if compared to the fairness obtained when applying the LST algorithm to the nadir point. Increasing the number of users to 4, 5 and 6 leads to improvements

of the fairness in 23.52%, 28.12%, and 22.12%, respectively.

6.4 Concluding remarks

In this chapter, we presented a different perspective on how we can model collaboration on parallel and distributed platforms. Instead of users sharing their resources, we consider the problem of users executing jobs on a set of resources belonging to the collectivity.

Our study is motivated by the problem known in the literature as the Multi-Users Scheduling Problem (MUSP), where users with different performance objectives execute their jobs in a shared set of resources.

First, we studied how to construct Pareto approximations of the MUSP problem, studying different setups with varying number of users, objectives and resources. Then, we presented a new method to address problems where the optimization function is monotonic and the arguments of this function can be optimized. Essentially, the method enumerates the frontier of best compromise solutions between these arguments and selects the solution that brings the best value for the function to optimize.

We presented a detailed description of how the technique works by demonstrating its application to the MUSP($1 : \sum C_i; 2 : C_{\max}$) problem, for which we proved that the Latest Starting Time (LST) algorithm provides a $<1, \bar{1}, \ldots, \bar{1}>$-approximation of the Pareto set.

Although the algorithm is presented under the specific context of fairness optimization on the MUSP problem, the technique is generic and can be applied for a large number of multi-objective problems and monotonic functions.

Chapter 7

Conclusion and perspectives

Computer science is deeply changing methodological aspects of the discovery process in different areas of knowledge. Researchers have at their disposal today new capabilities that can create novel research opportunities and also accelerate the discovery process. The question now is how to make these new capabilities accessible to every researcher at every level and not only to the few ones participating in bigger (and wealthy) scientific projects.

Our work was motivated by the belief that this could be accomplished through the collaboration between individuals and organizations, even if in practice the participants may have different (potentially conflicting) expectations for the results of the collaboration.

In this thesis, we studied how scheduling theory can be applied in order to create a parallel or distributed platform where all participants always feel stimulated to share their own computational resources in exchange of being able to execute their own jobs more efficiently. Each chapter explored a different facet of the rules that govern how these organizations engage in collaboration in practice and showed how to obtain schedules with good trade-offs between the results got by the participants under this set of rules.

The first kind of interaction was studied on Chapter 3. We studied what happens when the participants cannot be entirely trusted in the sense that they may not exactly follow the orders given by a central scheduler. We assumed that the participants of the platform are organizations that act in an individualist and selfish way, capable

of taking advantage of the fact that they have complete control over their own resources.

We showed how a centralized scheduler can compute an execution order for the jobs that does not allow a situation where a selfish organization could be tempted to change the schedule devised in order to execute its own jobs more efficiently at the expense of the others. Still, the algorithm is capable of guaranteeing a 2-approximation for the global $C_{\max}$ while ensuring that no organization will worsen its own results by sharing its resources with the others.

The effort to ensure that the results will not be worsen if compared to what the participant could obtain using only its resources is important to guarantee that the participant will always have incentive to cooperate. But these individual guarantees impose a performance penalty on the global performance of the platform. In Chapter 4, we studied the correlation between the guarantees that we can provide for each participant, in order to incentive collaboration, and the guarantees we can provide for the collectivity, in order to improve the global performance.

We presented algorithms that allow more altruistic participants to tolerate a bounded amount of degradation on their results in exchange of better global performances. We showed that the approximation ratio for the global makespan of 2 (for the case where no degradation is tolerated) can be improved to $\frac{3}{2}$ if each participant allows its makespan to be at most doubled and improved to $\frac{4}{3}$ if each participant allows its makespan to be at most tripled.

We also studied inapproximability bounds for the problem and showed that the results obtained with our algorithms are not too far from the Pareto set of the problem. The analysis suggested that algorithms with guaranteed approximation ratios of $\left(2; \frac{4}{3}\right)$ or $\left(3; \frac{5}{4}\right)$ are still possible. The development of such algorithms is part of the perspectives for future works.

On all situations presented above, the decision-making process is made by a centralized agent that computes a scheduling respecting some predefined rules. In Chapter 5 we used algorithmic game-theory to extend the notions of independence and selfishness of each organization. We gave to the participants the freedom to choose the best strategy for their jobs. The goal was to study the interactions between the independent organizations as the result of rational selfish players attempting to reach an equilibrium.

The model represents the problem as a game, where each participant can choose on which organization each one of its jobs will be executed. The selfishness of each organization is expressed with coordination mechanisms that assume that each organization will always execute its own jobs first and only after the jobs from other organizations (that in turn will be scheduled according some predefined rule).

If the local scheduling of the foreign jobs mimics classic list scheduling algorithms, fixing a priory for each job individually, then we found that no pure ϵ-approximate equilibrium is possible for values of ϵ less than 2. In this case, even the problem of determining if an instance admits a pure Nash equilibrium is co-NP hard.

If the local scheduling uses the information about the organization that own the job to prioritize the jobs, then we can show how to make the game converge to an equilibrium state. We presented an algorithm that can compute an approximate pure equilibrium that produces results as close as wanted to a pure Nash equilibrium. We also showed that with this local scheduling policy it is possible to bound the price of anarchy — the ratio between the social cost (the global $C_{\max}$) of a worst-case Nash equilibrium and the social cost of an optimal assignment.

This game-theoretic model opens the possibility for new future works to explore more coordination mechanisms for the problem. It would be interesting to study new coordination mechanisms leading to results with additional guarantees like fairness or a better social welfare, for instance. Another interesting research opportunity is to use this game-theoretic model to explore the more altruistic way of collaboration studied in Chapter 4.

The last aspect of collaboration on parallel and distributed platforms studied in this work is relative to the ownership of the resources being shared. Unlike the other cases — where each participant contributes with a portion of the available resources — we studied in Chapter 6 the case where the resources belong to the community and no one has special control or privileges over them.

In this context, we have a set of different users competing for a common set of resources in order to achieve the best performance possible. There is no “best solution” possible, but actually a set of solutions that offer different compromises for the users.

There are several properties on such solutions that can be of interest. One that naturally come to light in this context is how fair are the results obtained by

each participant. We presented a generic method to optimize monotonic objective functions (like fairness) that depend on arguments that can be optimized. Essentially, the method enumerates the frontier of best compromise solutions between these arguments and selects the solution that brings the best value for the function to optimize.

We have illustrated the use of the method on the scheduling problem where different users — interested in minimizing either the makespan or the average completion time of their jobs — compete for a limited set of resources. Using new Pareto set approximations for the problem, we showed how our method can be used to find a fair scheduling that offers a good compromise between the performance perceived by the users.

The method can be applied to a wide variety of multi-objective problems that we expect to explore in future works. We also hope to extend the Pareto set approximations in order to apply them for more generic instances of the problem.

Bibliography

[AF06] David P. Anderson and Gilles Fedak. “The Computational and Storage Potential of Volunteer Computing”. In: *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid*. (Singapore). Los Alamitos, USA: IEEE Computer Society, May 2006, pp. 73–80. DOI: 10.1109/CCGRID.2006.101.

[Agn+00] Alessandro Agnetis, Pitu B. Mirchandani, Dario Pacciarelli, and Andrea Pacifici. “Nondominated Schedules for a Job-Shop with Two Competing Users”. In: *Computational & Mathematical Organization Theory* 6.2 (2000), pp. 191–217. DOI: 10.1023/A:1009637419820.

[Agn+04] Allesandro Agnetis, Pitu B. Mirchandani, Dario Pacciarelli, and Andrea Pacifici. “Scheduling Problems with Two Competing Agents”. In: *Operations Research* 52.2 (Apr. 2004), pp. 229–242. DOI: 10.1287/opre.1030.0092.

[And+02] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. “SETI@home: an experiment in public-resource computing”. In: *Communications of the ACM* 45 (11 Nov. 2002), pp. 56–61. DOI: 10.1145/581571.581573.

[And04] David P. Anderson. “BOINC: A System for Public-Resource Computing and Storage”. In: *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. (Pittsburgh, USA). Washington, DC, USA:

IEEE Computer Society, Nov. 2004, pp. 4–10. DOI: 10.1109/GRID.2004.14.

[BCR80] Brenda S. Baker, Edward G. Coffman Jr., and Ronald L. Rivest. "Orthogonal Packings in Two Dimensions". In: *SIAM Journal on Computing* 9.4 (Nov. 1980), pp. 846–855. DOI: 10.1137/0209064.

[BCS74] John L. Bruno, Edward G. Coffman Jr., and Ravi Sethi. "Scheduling independent tasks to reduce mean finishing time". In: *Communications of the ACM* 17.7 (July 1974), pp. 382–387. ISSN: 0001-0782. DOI: 10.1145/361011.361064.

[Bou+10] Marin Bougeret, Pierre-François Dutot, Klaus Jansen, Christina Otte, and Denis Trystram. "Approximation Algorithms for Multiple Strip Packing". In: *Proceedings of the 7th International Workshop on Approximation and Online Algorithms*. (Copenhagen, Denmark). Ed. by Evripidis Bampis and Klaus Jansen. Vol. 5893. Lecture Notes in Computer Science. Heidelberg: Springer, Sept. 2010, pp. 37–48. DOI: 10.1007/978-3-642-12450-1_4.

[Bra+08] Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Słowiński, eds. *Multiobjective optimization: interactive and evolutionary approaches*. Vol. 5252. Lecture Notes in Computer Science. Springer, 2008. ISBN: 978-3-540-88907-6. DOI: 10.1007/978-3-540-88908-3.

[Bru07] Peter Brucker. *Scheduling Algorithms*. 5th ed. Heidelberg: Springer, 2007. ISBN: 978-3-540-69515-8.

[Car+06] Ioannis Caragiannis, Michele Flammini, Christos Kaklamanis, Panagiotis Kanellopoulos, and Luca Moscardelli. "Tight Bounds for Selfish and Greedy Load Balancing". In: *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming*. (Venice, Italy). Ed. by Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener. Vol. 4051. Lecture Notes in Computer Science. Heidelberg: Springer, July 2006, pp. 311–322. DOI: 10.1007/11786986_28.

[CD06] Xi Chen and Xiaotie Deng. "Settling the Complexity of Two-Player Nash Equilibrium". In: *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*. (Berkeley, CA, USA). Washington, DC, USA: IEEE Computer Society, Oct. 2006, pp. 261–272. DOI: 10.1109/FOCS.2006.69.

[Cir+06] Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro B. Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray. "Labs of the World, Unite!!!" In: *Journal of Grid Computing* 4.3 (2006), pp. 225–246. DOI: 10.1007/s10723-006-9040-x.

[CKN04] George Christodoulou, Elias Koutsoupias, and Akash Nanavati. "Coordination Mechanisms". In: *Proceedings of the 31st International Colloquium on Automata, Languages, and Programming*. (Turku, Finland). Ed. by Josep Diaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella. Vol. 3142. Lecture Notes in Computer Science. Heidelberg: Springer, July 2004, pp. 45–56. DOI: 10.1007/978-3-540-27836-8_31.

[CKV02] Arthur Czumaj, Piotr Krysta, and Berthold Vöcking. "Selfish traffic allocation for server farms". In: *Proceedings of the 34th annual ACM symposium on Theory of computing*. (Montreal, Canada). Ed. by John Reif. New York, NY, USA: Association for Computing Machinery, May 2002, pp. 287–296. DOI: 10.1145/509907.509952.

[Coh+10] Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. "Analysis of Multi-Organization Scheduling Algorithms". In: *The 16th International Conference on Parallel Computing (Euro-Par)*. Ed. by Pasqua D'Ambra, Mario Guarracino, and Domenico Talia. Vol. 6272. Lecture Notes in Computer Science. Heidelberg: Springer, 2010, pp. 367–379. DOI: 10.1007/978-3-642-15291-7_34.

[Coh+11a] Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. "Coordination Mechanisms for Selfish Multi-Organization Scheduling". In: *Proceedings of the 18th annual IEEE International Conference on High Performance Computing (HiPC)*. (Bangalore, India). To appear. Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2011.

[Coh+11b] Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. “Multi-organization scheduling approximation algorithms”. In: *Concurrency and Computation: Practice and Experience* 23 (17 2011), pp. 2220–2234. DOI: 10.1002/cpe.1752.

[Cor+09] Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frédéric Wagner. “Comment rater la validation de votre algorithme d’ordonnancement”. In: *Proceedings of the 19th Rencontres francophones du Parallélisme (RenPar)*. (Toulouse, France). Poster. Sept. 2009.

[Cor+10] Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frédéric Wagner. “Random graph generation for scheduling simulations”. In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTOOLS)*. (Torremolinos, Spain). Brussels, Belgium: Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2010, 60:1–60:10. ISBN: 978-963-9799-87-5. DOI: 10.4108/ICST.SIMUTOOLS2010.8667.

[Cor+11] Daniel Cordeiro, Pierre-François Dutot, Grégory Mounié, and Denis Trystram. “Tight Analysis of Relaxed Multi-Organization Scheduling Algorithms”. In: *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. (Anchorage, AL, USA). Los Alamitos, CA, USA: IEEE Computer Society, May 2011, pp. 1177–1186. DOI: 10.1109/IPDPS.2011.112.

[DT09] Christoph Dürr and Nguyen Kim Thang. “Non-clairvoyant Scheduling Games”. In: *Proceedings of the 2nd International Symposium on Algorithmic Game Theory*. (Paphos, Cyprus). Ed. by Marios Mavronicolas and Vicky G. Papadopoulou. Vol. 5814. Lecture Notes in Computer Science. Heidelberg: Springer, Oct. 2009, pp. 135–146. DOI: 10.1007/978-3-642-04645-2_13.

[Dut+09] Pierre-François Dutot, Krzysztof Rządca, Érik Saule, and Denis Trystram. “Multi-objective scheduling”. In: *Introduction to scheduling*. Ed.

by Yves Robert and Frédéric Vivien. Boca Raton, FL, USA: Chapman & Hall/CRC Press, Nov. 2009. Chap. 9. ISBN: 978-1-4200-7273-0.

[Dut+11] Pierre-François Dutot, Fanny Pascual, Krzysztof Rządca, and Denis Trystram. "Approximation Algorithms for the Multiorganization Scheduling Problem". In: *IEEE Transactions on Parallel and Distributed Systems* 22.11 (Nov. 2011), pp. 1888–1895. DOI: 10.1109/TPDS.2011.47.

[DY07] Ilias Diakonikolas and Mihalis Yannakakis. "Small Approximate Pareto Sets for Bi-objective Shortest Paths and Other Problems". In: *Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*. Ed. by Moses Charikar, Klaus Jansen, Omer Reingold, and José Rolim. Lecture Notes in Computer Science. Heidelberg: Springer, 2007, pp. 74–88. DOI: 10.1007/978-3-540-74208-1_6.

[EKM07] Eyal Even-Dar, Alex Kesselman, and Yishay Mansour. "Convergence time to Nash equilibrium in load balancing". In: *ACM Transactions on Algorithms* 3.3 (Aug. 2007). DOI: 10.1145/1273340.1273348.

[Emm+06] Stephen Emmott et al. *Towards 2020 Science*. Working Group Report. Microsoft Research Cambridge, 2006. URL: http://research.microsoft.com/towards2020science/ (visited on 10/13/2011).

[Fei09] Dror G. Feitelson. *The ANL Intrepid log. The Parallel Workloads Archive*. 2009. URL: http://www.cs.huji.ac.il/labs/parallel/workload/l_anl_int/ (visited on 11/03/2011).

[FK04] Ian Foster and Carl Kesselman. *The Grid. Blueprint for a New Computing Infrastructure*. 2nd ed. The Elsevier Series in Grid Computing. Morgan Kaufmann, 2004. ISBN: 978-1-55860-933-4.

[FKS05] Dimitris Fotakis, Spyros Kontogiannis, and Paul Spirakis. "Selfish unsplittable flows". In: *Theoretical Computer Science* 348.2 (Dec. 2005): *Automata, Languages and Programming: Algorithms and Complexity*, pp. 226–239. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2005.09.024.

[Fos11] Ian Foster. “Globus Online: Accelerating and Democratizing Science through Cloud-Based Services”. In: *IEEE Internet Computing* 15 (3 May 2011). Ed. by George Pallis, pp. 70–73. ISSN: 1089-7801. DOI: 10.1109/MIC.2011.64.

[Fot+02] Dimitris Fotakis, Spyros Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul Spirakis. “The Structure and Complexity of Nash Equilibria for a Selfish Routing Game”. In: *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming*. Ed. by Peter Widmayer, Stephan Eidenbenz, Francisco Triguero, Rafael Morales, Ricardo Conejo, and Matthew Hennessy. Vol. 2380. Lecture Notes in Computer Science. Heidelberg: Springer, July 2002, pp. 785–785. DOI: 10.1007/3-540-45465-9_12.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, Jan. 1979. ISBN: 0-716-71045-5.

[Gra66] Ronald L. Graham. “Bounds for certain multiprocessing anomalies”. In: *Bell System Technical Journal* 45.9 (Nov. 1966), pp. 1563–1581.

[Gra69] Ronald L. Graham. “Bounds on Multiprocessing Timing Anomalies”. In: *SIAM Journal on Applied Mathematics* 17.2 (Mar. 1969), pp. 416–429. DOI: 10.1137/0117039.

[Gra+79] Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. “Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey”. In: *Annals of Discrete Mathematics* 5 (2 1979). Ed. by Peter L. Hammer, Ellis L. Johnson, and Bernhard H. Korte, pp. 287–326. DOI: 10.1016/S0167-5060(08)70356-X.

[GRS96] Laurent George, Nicolas Rivierre, and Marco Spuri. *Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling*. Tech. rep. RR-2966. INRIA, 1996. OAI: hal.inria.fr:inria-00073732.

[GST09] Alain Girault, Érik Saule, and Denis Trystram. “Reliability versus performance for critical applications”. In: *Journal of Parallel and Distributed*

Computing 69.3 (Mar. 2009), pp. 326–336. DOI: j.jpdc.2008.11.002.

[Guo98] Lin Guo-Hui. "The exact bound of Lee's MLPT". In: *Discrete Applied Mathematics* 85.3 (July 1998), pp. 251–254. DOI: 10.1016/S0166-218X(97)00139-X.

[Hoc97] Dorit S. Hochbaum, ed. *Approximation Algorithms for NP-Hard Problems*. Boston, MA, USA: PWS Publishing Company, 1997. ISBN: 978-0-534-94968-6.

[HS88] Dorit S. Hochbaum and David B. Shmoys. "A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach". In: *SIAM Journal on Computing* 17.3 (June 1988), pp. 539–551. DOI: 10.1137/0217033.

[Imm+09] Nicole Immorlica, Li (Erran) Li, Vahab S. Mirrokni, and Andreas S. Schulz. "Coordination mechanisms for selfish scheduling". In: *Theoretical Computer Science* 410.17 (Apr. 2009): *Internet and Network Economics*, pp. 1589–1598. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2008.12.032.

[Inc10] Distributed Computing Technologies Inc. Oct. 13, 2010. URL: http://www.distributed.net/ (visited on 10/13/2011).

[Ios+06] Alexandru Iosup, Catalin Dumitrescu, Dick Epema, Hui Li, and Lex Wolters. "How are Real Grids Used? The Analysis of Four Grid Traces and Its Implications". In: *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*. (Barcelona, Spain). Ed. by Dennis Gannon, Rosa M. Badia, and Rajkumar Buyya. Los Alamitos, CA, USA: IEEE Computer Society, Sept. 2006, pp. 262–269. DOI: 10.1109/ICGRID.2006.311024.

[JCH84] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. *A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems*. Tech. rep. Digital Equipment Corporation, Sept. 1984. arXiv:cs/9809099.

[JST08] Emmanuel Jeannot, Érik Saule, and Denis Trystram. “Bi-objective Approximation Scheme for Makespan and Reliability Optimization on Uniform Parallel Machines”. In: *Proceedings of the 14th International Euro-Par Conference*. (Las Palmas de Gran Canaria, Spain). Ed. by Emilio Luque, Tomàs Margalef, and Domingo Benítez. Vol. 5168. Lecture Notes in Computer Science. Heidelberg: Springer, Aug. 2008, pp. 877–886. DOI: 10.1007/978-3-540-85451-7_94.

[KC03] Stephen D. Kleban and Scott H. Clearwater. “Fair Share on High Performance Computing Systems: What Does Fair Really Mean?” In: *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*. (Tokyo, Japan). Los Alamitos, CA, USA: IEEE Computer Society, May 2003, pp. 146–153. DOI: 10.1109/CCGRID.2003.1199363.

[Kel98] Hans Kellerer. “Algorithms for multiprocessor scheduling with machine release times”. In: *IIE Transactions* 30.11 (Nov. 1998), pp. 991–999. DOI: 10.1023/A:1007526827236.

[Kol+08] Mariana Kolberg, Daniel Cordeiro, Gerd Bohlender, Luiz Gustavo Fernandes, and Alfredo Goldman. “A Multithreaded Verified Method for Solving Linear Systems in Dual-Core Processors”. In: *Proceedings of the 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA)*. (Trondheim, Norway). May 2008.

[Kor+11] Eric J. Korpela, David P. Anderson, Robert Bankay, Jeff Cobb, Andrew Howard, Matt Lebofsky, Andrew P. V. Siemion, Joshua von Korff, and Dan Werthimer. “Status of the UC-Berkeley SETI Efforts”. In: *Proceedings of SPIE. Instruments, Methods, and Missions for Astrobiology XIV*. Ed. by Richard B. Hoover, Paul C. W. Davies, Gilbert V. Levin, and Alexei Y. Rozanov. Vol. 8152. 1. San Diego, California, USA: SPIE, Aug. 2011. DOI: 10.1117/12.894066.

[KP09] Elias Koutsoupias and Christos Papadimitriou. “Worst-case equilibria”. In: *Computer Science Review* 3.2 (May 2009), pp. 65–69. DOI: 10.1016/j.cosrev.2009.04.003.

[KP99] Elias Koutsoupias and Christos Papadimitriou. “Worst-Case Equilibria”. In: *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*. (Trier, Germany). Ed. by Christoph Meinel and Sophie Tison. Vol. 1563. Lecture Notes in Computer Science. Heidelberg: Springer, Mar. 1999, pp. 404–413. DOI: 10.1007/3-540-49116-3_38.

[Lee91] Chung-Yee Lee. “Parallel machines scheduling with nonsimultaneous machine available time”. In: *Discrete Applied Mathematics* 30.1 (Jan. 1991), pp. 53–61. DOI: 10.1016/0166-218X(91)90013-M.

[Leu04] Joseph Y. T. Leung, ed. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. 1st ed. Computer and Information Science Series. Boca Raton, FL, USA: Chapman & Hall/CRC Press, 2004. ISBN: 1-58488-397-9.

[LLM88] Michael J. Litzkow, Miron Livny, and Matt W. Mutka. “Condor – a hunter of idle workstations”. In: *Proceedings of the 8th International Conference on Distributed Computing Systems*. (San Jose, CA, USA). Los Alamitos, USA: IEEE Computer Society, June 1988, pp. 104–111. DOI: 10.1109/DCS.1988.12507.

[LO01] Lavy Libman and Ariel Orda. “Atomic Resource Sharing in Noncooperative Networks”. In: *Telecommunication Systems* 17 (4 Aug. 2001), pp. 385–409. ISSN: 1018-4864. DOI: 10.1023/A:1016770831869.

[LST90] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. “Approximation algorithms for scheduling unrelated parallel machines”. In: *Mathematical Programming* 46 (1 1990). 10.1007/BF01585745, pp. 259–271.

[LT07] Arnaud Legrand and Corinne Touati. *How to measure efficiency?* Tech. rep. RR-6216. INRIA, June 2007. OAI: hal.inria.fr:inria-00153720.

[MR02] Laurent Massoulié and James Roberts. “Bandwidth sharing: objectives and algorithms”. In: *IEEE/ACM Transactions on Networking* 10.3 (June 2002), pp. 320–328. DOI: 10.1109/TNET.2002.1012364.

[Nas51] John Nash. “Non-Cooperative Games”. In: *The Annals of Mathematics* 2 (Sept. 1951), pp. 286–295. JSTOR: 1969529.

[Nis+07] Noam Nisam, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. New York, NY, USA: Cambridge University Press, Sept. 2007. ISBN: 978-0-521-87282-9.

[OII09] Fukuhito Ooshita, Tomoko Izumi, and Taisuke Izumi. “A Generalized Multi-Organization Scheduling on Unrelated Parallel Machines”. In: *Proceedings of the 10th International Conference on Parallel and Distributed Computing, Applications and Technologies*. (Higashi Hiroshima, Japan). Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2009, pp. 26–33. DOI: 10.1109/PDCAT.2009.26.

[OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. Cambridge, MA, USA: MIT press, July 1994. ISBN: 978-0-262-65040-3.

[Pap94] Christos H. Papadimitriou. “On the complexity of the parity argument and other inefficient proofs of existence”. In: *Journal of Computer and System Sciences* 48.3 (1994), pp. 498–532. DOI: 10.1016/S0022-0000(05)80063-7.

[Pil+10] Laércio L. Pilla, Christiane Pousa Ribeiro, Daniel Cordeiro, and Jean-François Méhaut. *Charm++ on NUMA Platforms: the impact of SMP Optimizations and a NUMA-aware Load Balancer*. Presented at: The 4th workshop of the INRIA-Illinois Joint Laboratory on Petascale Computing. Urbana, IL, USA. Nov. 2010.

[Pil+11] Laércio L. Pilla, Christiane Pousa Ribeiro, Daniel Cordeiro, Abhinav Bhatele, Philippe O. A. Navaux, Jean-François Méhaut, and Laxmikant V. Kalé. *Improving Parallel System Performance with a NUMA-aware Load Balancer*. Tech. rep. TR-JLPC-11-02. INRIA-Illinois Joint Laboratory on Petascale Computing, July 2011. HDL: 2142/25911.

[Pin08] Michael L. Pinedo. *Scheduling. Theory, Algorithms, and Systems*. 3rd ed. Springer, 2008. ISBN: 978-0-387-78934-7.

[PM07] David A. Papa and Igor L. Markov. “Hypergraph partitioning and clustering”. In: *Handbook of approximation algorithms and metaheurististics*. Computer and Information Science Series. Boca Raton, FL, USA: Chapman & Hall/CRC Press, 2007. Chap. 61. ISBN: 978-1-58488-550-4. DOI: 10.1201/9781420010749.ch61.

[PRT07] Fanny Pascual, Krzysztof Rządca, and Denis Trystram. “Cooperation in Multi-organization Scheduling”. In: *Proceedings of the 13th International Euro-Par Conference*. (Rennes, France). Ed. by Anne-Marie Kermarrec, Luc Bougé, and Thierry Priol. Vol. 4641. Lecture Notes in Computer Science. Heidelberg: Springer, Aug. 2007, pp. 224–233. DOI: 10.1007/978-3-540-74466-5_25.

[PRT09] Fanny Pascual, Krzysztof Rządca, and Denis Trystram. “Cooperation in multi-organization scheduling”. In: *Concurrency and Computation: Practice and Experience* 21.7 (May 2009): *Special Issue: Euro-Par 2007*. Ed. by Luc Bougé and Christian Lengauer, pp. 905–921. ISSN: 1532-0626. DOI: 10.1002/cpe.1378.

[PY00] Christos H. Papadimitriou and Mihalis Yannakakis. “On the approximability of trade-offs and optimal access of Web sources”. In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. (Redondo Beach, CA, USA). Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2000, pp. 86–92. DOI: 10.1109/SFCS.2000.892068.

[Rad80] Roy Radner. “Collusive behavior in noncooperative epsilon-equilibria of oligopolies with long but finite lives”. In: *Journal of Economic Theory* 22.2 (Apr. 1980), pp. 136–154. ISSN: 0022-0531. DOI: 10.1016/0022-0531(80)90037-X.

[Ram+11] Juan Ramírez-Alcaraz, Andrei Tchernykh, Ramin Yahyapour, Uwe Schwiegelshohn, Ariel Quezada-Pina, José González-García, and Adán Hirales-Carbajal. “Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids”. In: *Journal of Grid Computing* 9 (1 2011), pp. 95–116. DOI: 10.1007/s10723-011-9179-y.

[ST09] Érik Saule and Denis Trystram. “Multi-Users Scheduling in Parallel Systems”. In: *Proceedings of the 23rd International Parallel and Distributed Processing Symposium*. (Rome, Italy). Los Alamitos, CA, USA: IEEE Computer Society, May 2009, pp. 1–9. DOI: 10.1109/IPDPS.2009.5161037.

[STY08] Uwe Schwiegelshohn, Andrei Tchernykh, and Ramin Yahyapour. “Online scheduling in grids”. In: *Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium*. (Miami, FL, USA). Los Alamitos, USA: IEEE Computer Society, Apr. 2008, pp. 1–10. DOI: 10.1109/IPDPS.2008.4536273.

[SV07] Petra Schuurman and Tjark Vredeveld. “Performance Guarantees of Local Search for Multiprocessor Scheduling”. In: *INFORMS Journal on Computing* 19.1 (Feb. 2007), pp. 52–63. ISSN: 1526-5528. DOI: 10.1287/ijoc.1050.0152.

[Tay06] John Taylor. *UK National e-Science Centre definition of e-Science*. Sept. 18, 2006. URL: http://www.nesc.ac.uk/nesc/define.html (visited on 10/13/2011).

[Vöc07] Berthold Vöcking. “Selfish Load Balancing”. In: Noam Nisam, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. New York, NY, USA: Cambridge University Press, Sept. 2007. Chap. 20. ISBN: 978-0-521-87282-9.

[Voo03] Mark Voorneveld. “Characterization of Pareto Dominance”. In: *Operations Research Letters* 31.1 (Jan. 2003), pp. 7–11. DOI: 10.1016/S0167-6377(02)00189-X.

[WK11] George Woltman and Scott Kurowski. *The Great Internet Mersenne Prime Search*. Feb. 23, 2011. URL: http://www.mersenne.org/ (visited on 10/13/2011).

[YHZ09] Deshi Ye, Xin Han, and Guochuan Zhang. “On-Line Multiple-Strip Packing”. In: *Proceedings of the 3rd International Conference on Combinatorial Optimization and Applications*. (Huangshan, China). Ed. by Ding-Zhu Du, Xiaodong Hu, and Panos M. Pardalos. Vol. 5573. Lecture Notes in

Computer Science. Heidelberg: Springer, June 2009, pp. 155–165. DOI: 10.1007/978-3-642-02026-1_14.

[Zar05] Christos Zaroliagis. “Recent Advances in Multiobjective Optimization”. In: *Stochastic Algorithms: Foundations and Applications*. Ed. by Oleg Lupanov, Oktay Kasim-Zade, Alexander Chaskin, and Kathleen Steinhöfel. Vol. 3777. Lecture Notes in Computer Science. Heidelberg: Springer, 2005, pp. 45–47. DOI: 10.1007/11571155_5.

[Zhu06] S. N. Zhuk. “Approximate algorithms to pack rectangles into several strips”. In: *Discrete Mathematics and Applications* 16.1 (Jan. 2006), pp. 73–85. DOI: 10.1515/156939206776241264.

Résumé étendu en français

Introduction

LABS OF THE WORLD, UNITE ! ! ! Ce titre provocateur de l'article écrit par CIRNE et al. [Cir+06] allégorise comment les changements récents dans l'informatique ont transformé la quête d'une puissance de calcul toujours plus importante (à plus faible coût) en un effort de collaboration.

Aujourd'hui, la communauté scientifique a la capacité sans précédent de combiner des différentes ressources de calcul (éventuellement réparties autour du globe) en un système distribué puissant, capable d'analyser des volumes massifs de données. Les défis et opportunités créés par l'utilisation de ces nouvelles technologies dans des projets de recherche appliquées à grande échelle — habituellement dénotés par e-Science [Tay06] — est en train de changer la façon dont les chercheurs s'engagent dans ce processus de découverte [Fos11].

Des concepts, outils et théorèmes de l'informatique sont maintenant intégrés aux aspects méthodologiques dans plusieurs domaines de recherche. Climatologie, sciences de la terre, dynamique des fluides, génomique, protéomique, chimie théorique, astrophysique et physique des hautes énergies sont quelques exemples [Emm+06] des domaines de recherche qui poussent les limites de ce qui est possible avec l'informatique.

Les informaticiens ont appris depuis longtemps que la nécessité pour des montants plus élevés de puissance de calcul demandée par les laboratoires de recherche devient

de plus en plus importante. Le projet Condor [LLM88] a joué un rôle pionnier dans cette tendance. C'était l'une des premières initiatives axées sur l'utilisation de la puissance de calcul vacante de nombreux ordinateurs personnels (banalisés), visant à fournir des grandes quantités de puissance de calcul maintenue sur une longue période de temps (*High Throughput Computing*).

Avec le développement des technologies des réseaux, les techniques utilisées par le projet Condor furent rapidement adaptées pour travailler avec toutes les machines connectées à Internet. C'est ce qui a été appelé le "calcul volontaire" — *volunteer computing* [AF06], aussi connu sous le nom "calcul pair-à-pair" (*peer-to-peer computing*) ou "systèmes de calcul global" (*global computing*) — les chercheurs ont commencé à étudier comment utiliser la puissance de calcul non utilisée (accessible via Internet) dans la résolution de problèmes spécifiques.

Plusieurs travaux de recherche — universitaires ou non — ont exploré l'idée de l'utilisation de calcul volontaire dès ses premières étapes. Des projets comme "The Great Internet Mersenne Prime Search" (1996) [WK11] (qui cherche les nombres premiers de type Mersenne), le distributed.net (1997) [Inc10] (qui a démontré le déchiffrement par brute force), et le projet SETI@home (1999) [And+02] (qui analyse de grands ensembles de données de signaux radio, à la recherche de signes d'intelligence extra-terrestre) ont montré les possibilités offertes pour de telles techniques en terme de puissance de calcul.

SETI@home est sans doute le plus emblématique de tous les projets de calcul volontaire. Lancé en mai 1999, SETI@home a attiré l'attention non seulement de la communauté scientifique, mais aussi du public en général. La possibilité de collaborer dans un projet scientifique a attiré plus de 6 millions de participants, situés dans 226 pays [Kor+11]. Le projet a évolué et il utilise aujourd'hui la plate-forme appelé *Berkeley Open Infrastructure for Network Computing* (BOINC [And04]), un intergiciel pour l'informatique volontaire distribuée. SETI@home reste encore l'un des plus grands super-calculateurs en fonctionnement, ayant une performance moyenne de 3,5 PFLOPS.

À peu près au même moment, Ian Foster et Carl Kesselman ont développé le concept de Grille de Calcul (*grid computing*) [FK04], dans lequel la puissance de calcul est fournie sur demande, comme un service. Comme dans les projets précédents, différentes organisations s'associent pour partager leurs ressources de calcul. Mais,

désormais, dans une grille de calcul la plate-forme complète est disponible comme un service pour tous les participants. Chaque organisation participante peut agir en tant que producteur ou consommateur de ressources.

Comme dans les “réseaux électriques” traditionnels — où différentes centrales électriques sont reliées de façon à ce que les utilisateurs puissent accéder à l’électricité sans se soucier ni où, ni comment, l’électricité est générée — les plates-formes de grille de calcul permettent aux utilisateurs d’utiliser différentes ressources de calcul, au sein et entre les organisations, sans avoir connaissance où ces ressources se trouvent physiquement.

La vision proposé par Foster et Kesselman est maintenant une réalité. Des plates-formes de grille de calcul sont utilisées dans des projets académiques et commerciaux. FOSTER [Fos11] a présenté quelques exemples contemporains qui montrent l’ampleur des projets qu’utilisent les technologies de grille de calcul. Parmi les initiatives académiques, on peut citer la plate-forme de grille de calcul du *Large Hadron Collider* (LHC) (qui distribue régulièrement des dizaines de téraoctets à des équipes de recherche du monde entier), la plate-forme du *Earth System Grid* (qui donne accès à plus de 25.000 utilisateurs à une grande base de données des simulations de changement climatique), le *US InCommon trust federation* (qui permet à plus de 5 millions de personnes d’accéder aux ressources distantes en utilisant des informations d’identification locales), entre autres. Parmi les systèmes commerciaux nous avons aujourd’hui les fournisseurs de *cloud computing* commerciales. Ces fournisseurs offrent de la puissance de calcul et du stockage mémoire à des échelles irréalisables dans les milieux universitaires.

Bien que le calcul volontaire et les grilles informatiques soient très semblables dans le sens où les deux permettent la création d’une plate-forme composée d’un grand nombre de ressources de calcul, la façon dont les projets utilisent ces plates-formes est complètement différent.

Dans le calcul volontaire, chaque volontaire choisit effectivement le projet qu’il veut aider. L’exécution du projet repose sur la bonne volonté des nœuds volontaires, qui habituellement ne gagnent rien pour leur aide. Ils sont généralement anonymes et ne peuvent pas être tenus responsables de la qualité des ressources ou des résultats qu’ils fournissent.

Toutefois, dans l’informatique en grilles, les interactions entre les participants sont

plus complexes. Une organisation choisit d'intégrer une plate-forme informatique en grille non seulement pour contribuer avec ses ressources de calcul à d'autres projets. Leurs utilisateurs s'attendent également à être capable d'exécuter leurs propres tâches d'une façon plus efficace avec l'aide des autres.

On pourrait avoir un conflit d'intérêts si les ressources de la grille informatique ne sont pas partagées de manière équitable. Par exemple, si une organisation suppose que moins de ressources sont allouées à ses propres projets (par rapport aux ressources allouées aux autres), l'organisation peut être contrainte de quitter la grille et de cesser de partager ses ressources.

Un autre exemple serait celui où l'une des organisations se comporte mal, pour une raison ou pour une autre, dans le but d'améliorer l'exécution de ses propres projets. Par exemple, une organisation malveillante pourrait refuser l'exécution d'un projet autre que ses propres projets. Ces organisations doivent être tenues pour responsables de leur mauvaise conduite. Selon les circonstances, l'organisation pourrait voir son accès aux ressources des autres refusé ou même avoir à répondre légalement pour son mauvais comportement...

De grands projets de recherche tels que ceux cités par FOSTER [Fos11] peuvent éviter ces problèmes en créant et gérant des infrastructures de grille dédiées, où chaque détail de l'exécution peut être contrôlé. Mais afin d'offrir une plate-forme en grille informatique vraiment collaborative — une plate-forme où les équipes de recherche (avec des différents budgets, tailles et objectifs) peuvent avoir les mêmes avantages et la même compétitivité que les grands projets de recherche — il faut respecter les intérêts de chaque utilisateur.

Contenu général

Dans cette thèse, nous étudions comment encourager la collaboration entre différents utilisateurs et organisations. Nous modélisons les variantes de ce problème comme un problème d'ordonnancement multi-objectifs et, en utilisant des différents outils théoriques, nous étudions comment le comportement individuel de ces utilisateurs peut influencer à la fois la performance perçue par les composantes individuelles de la plate-forme et l'efficacité globale de la plate-forme.

Dans un premier temps, nous donnons le cadre théorique qui sous-tend les études de ce travail de doctorat et le contexte général. En particulier, la Section 2.1 présente rapidement des rappels en théorie de l'ordonnancement classique (mono-objectif).

Puis, la Section 2.2 introduit les concepts fondamentaux en optimisation multi-objectifs. L'idée ici est de présenter les techniques qui permettent l'étude des compromis pour choisir un ordonnancement qui satisfasse les différents objectifs individuels.

Finalement, la Section 2.3 présente les concepts de base de la Théorie algorithmique des jeux (*pure-strategy games*) et comment les interactions entre individus sont analysées sous l'hypothèse qu'ils agissent de façon rationnelle.

Nous présentons dans les quatre sections suivantes un résumé des contributions de chaque chapitre. Les détails sont disponibles en anglais dans le document complet.

Algorithmes d'approximation pour le problème d'ordonnancement multi-organisation

Les utilisateurs de grilles de calcul contribuent à ces systèmes en fournissant de la puissance de calcul. Ils attendent en retour que leurs propres jobs soient exécutés plus efficacement grâce au partage de ces ressources avec les autres.

Les caractéristiques telles que l'hétérogénéité des ressources disponibles, le grand nombre de processeurs et les différentes demandes des utilisateurs rendent difficile le problème d'ordonnancement sur de telles plates-formes. Pour pouvoir exploiter au mieux de tels systèmes il faut concevoir des algorithmes d'ordonnancement sophistiqués qui doivent à la fois encourager les utilisateurs à partager leurs ressources et respecter les intérêts propres de chaque utilisateur.

Dans ce chapitre, nous présentons une manière de traiter ce problème. Notre contribution principale est l'extension et l'analyse du problème dans le cas de jobs séquentiels soumis par des organisations individuelles qui ont chacune un objectif particulier : *makespan* (maximum des temps de complétion) ou *mean flow time* (temps de complétions moyens).

Nous introduisons de nouvelles restrictions sur l'ordonnancement pour tenir compte du concept d'ordonnancement égoïste (*selfish organizations*), c'est-à-dire,

lorsque les organisations refusent de coopérer si leur propre objectif peuvent être améliorés en exécutant un de leur job plus tôt sur une machine de leur organisation.

Nous proposons ici une étude de l'équité (*fairness*) en utilisant plusieurs métriques. Notre but ici est d'évaluer si les résultats obtenus par les organisations améliorent de façon égale tous les objectifs locaux en construisant des ordonnancements équitables par rapport aux métriques considérés. Perdre l'équité pour une organisation particulière peut être considéré comme une raison d'arrêter la coopération avec les autres.

Les résultats plus précis sont les suivants : la Section 3.2 montre que n'importe quel algorithme qui respecte les restrictions d'égoïsme ne peuvent être approximés à mieux qu'un facteur 2 de l'optimal. de plus, ce problème est NP-difficile que les organisations soient intéressées par minimiser leur makespan ou le temps de complétion moyen. Puis, nous présentons plusieurs algorithmes d'approximation qui atteignent ce rapport de 2 dans la Section 3.3. Ce algorithmes sont analysés vis à vis des métriques d'équité dans la Section 3.4. Enfin, des expérimentations menées sur des simulations montrent les bons résultats de ces algorithmes en moyenne et plus particulièrement, comment chaque organisation perçoit l'équité dans la Section 3.5.

Algorithmes relâchés pour le problème d'ordonnancement multi-organisation

Les résultats présentés au chapitre précédent montrent que d'une part, il est possible aux ordonnancements d'encourager la coopération en garantissant des performances pour chaque organisation indépendantes. D'autre part, ils montrent aussi que de tels encouragements ont un impact sur la performance globale.

Tout d'abord, on distingue l'impact sur les contraintes locales qui induisent une dégradation du makespan global. En particulier, nous avons montré (voir l'exemple de la Figure 3.2) que tout algorithme qui respecte ces contraintes locales a au moins un facteur d'approximation de $\frac{3}{2}$ par rapport à l'optimum global obtenu avec des organisations altruistes (c'est-à-dire, celles qui sont prêtes à collaborer en dégradant leurs propres jobs).

Puis, nous étudions l'influence des contraintes d'une attitude égoïste sur la per-

formance globale. En l'absence de telles contraintes, nous montrons à travers la Proposition 3.1 qu'il n'existe pas d'algorithme d'approximation avec un rapport d'approximation asymptotique meilleur que 2. Enfin, en mixant les deux types de contraintes, il est impossible d'obtenir un rapport meilleur que $2 - \frac{2}{N}$, en vertus de la Proposition 3.2.

Ainsi, il existe une corrélation claire entre les garanties que l'on peut attendre de chaque organisation pour inciter à la collaboration et la garantie obtenue pour l'optimisation globale. L'objectif de ce chapitre est d'étudier de telles corrélations lorsque les organisations locales acceptent de dégrader leurs performances locales d'un certain facteur fixé.

Plus précisément, la Section 4.1 présente comment nous avons modélisé la corrélation à partir du problème MOSP relaxé (noté α-MOSP) et quelques résultats existant autour de modèles poches. Dans la Section 4.2, nous montrons comment améliorer la borne précédente d'inapproximation en montrant en particulier (contrairement à l'intuition) il n'existe pas d'algorithmes polynomiaux pour obtenir de telles bornes (sauf si $P = NP$). Nous présentons également deux familles d'instances dont les points de Pareto optimaux atteignent les bornes d'inapproximation. Deux nouveaux algorithmes sont proposés et analysés dans la Section 4.3. Le premier est une $\frac{3}{2}$-approximation pour l'objectif global, avec la garantie qu'aucune organisation ne dégrade son makespan de plus d'un facteur 2. Cette solution est Pareto optimale selon [OII09]. Le second garantie une $\frac{4}{3}$-approximation pour le makespan global avec une dégradation locale limitée à 3. Cette solution appartient à la frontière de la courbe d'inapproximabilité de la seconde famille et, ainsi, est Pareto-optimale.

Mécanismes de coordination pour l'ordonnancement multi-organisation dans un cadre égoïste

Jusqu'à présent, nous avons étudié les notions de coopération et d'égoïsme appliquées au problème d'ordonnancement multi-organisations en utilisant des approches classiques de l'optimisation combinatoire. Dans ce chapitre, nous proposons une extension des notions d'indépendance et d'égoïsme des organisations en les autorisant de choisir rationnellement la meilleure stratégie possible pour exécuter leurs jobs.

Le but ici est d'étudier les interactions entre organisations indépendantes comme résultat d'un jeu et la recherche d'équilibres.

La principale contribution de ce chapitre est d'avoir modéliser le problème d'ordonnancement multi-organisation sous forme de jeu qui conduit à des configurations qui sont aussi prêts possible d'un équilibre de Nash avec un prix de l'anarchie borné par une constante. L'idée est d'étudier le problème comme un jeu non coopératif avec des mécanismes de coordination qui permettent à chaque organisation de déterminer le meilleur makespan possible jusqu'à obtenir un équilibre.

Plus précisément, le chapitre est organisé comme suit : la Section 5.1 décrit le modèle lui-même, puis la Section 5.2 étudie le mécanisme de coordination basé sur des résultats classiques en ordonnancement. On montre qu'utiliser ce genre de mécanismes ne permet pas d'admettre des équilibres à ϵ-approximation avec $\epsilon < 2$ et que décider s'il existe une instance particulière qui admet un équilibre de Nash pur est co-NP difficile. Dans la Section 5.3, on définit un autre mécanisme de coordination où la priorité est donnée aux organisations, et non plus aux tâches. Nous proposons alors un algorithme qui construit une ρ-approximation d'un équilibre pur et nous analysons le prix de l'anarchie correspondant.

Ordonnancement multi-utilisateurs

Il est fréquent que des utilisateurs doivent se regrouper pour partager leurs ressources de calcul. Ces ressources appartiennent ainsi non plus à des utilisateurs individuels, mais à une communauté. Dans ce cas, les utilisateurs n'ont plus de privilèges spéciaux sur une partie de ces ressources. Ceci diffère des problèmes multi-organisations étudiés dans les chapitres précédents. Ils doivent également collaborer et coordonner l'utilisation de ces ressources pour atteindre les meilleures performances de façon la plus équitable possible.

Un exemple typique de ces problèmes est l'ordonnancement multi-utilisateurs dans les grappes de calcul [KC03], où plusieurs utilisateurs doivent exécuter leurs jobs sur des processeurs de la grappe où aucun n'a le contrôle spécifique sur ces machines.

Une question délicate qui se pose dans ce type d'environnements porte sur comment ordonnancer tous les jobs en garantissant une certaine équité entre les

utilisateurs. Plus que garantir certaines performances, l'ordonnanceur devrait pouvoir respecter les intérêts de chacun. A notre connaissance, ce problème n'est pas résolu à ce jour, en particulier, aucune étude théorique n'a été menée.

Ce type d'interrogation mêlant performances et équité se pose également du point de vue plus pratique lors de l'exécution sur des plates-formes parallèles. Quelques problèmes proches ont été étudiés, tels que le partage de la bande passante dans les réseaux [MR02], ou la négociation entre des partenaires industriels [Agn+00], ou le partitionnement d'applications par hyper-graphes [PM07], etc.

De tels problèmes d'optimisation mettent en jeu de nombreux paramètres complexes. Pour concevoir une méthode raisonnable de résolution, on doit la plupart du temps considérer une fonction objective plus simple. Cependant, souvent il n'est pas suffisant d'optimiser un seul objectif. Ainsi, nous présentons dans ce chapitre une méthode générique multi-objective pour optimiser le front de Pareto où chaque paramètre est une fonction objectif particulière.

Cette méthode est mise en œuvre sur le problème MUSP avec optimisation de l'équité entre utilisateurs. Plus précisément, la Section 6.1 présente le problème d'ordonnancement multi-utilisateurs sur les mêmes bases que le multi-organisations et les notations associées. La Section 6.2 montre comment construire des approximation de Pareto pour des instances particulières du problème MUSP. Puis, nous introduisons une nouvelle méthode pour optimiser des fonctions objectifs monotones pour une formulation multi-objectifs du problème à la Section 6.3. Cette technique fonctionne à l'aide de bornes inférieure et supérieure qui servent à guider l'espace des solutions autour de l'ensemble de Pareto. Nous détaillons comment appliquer cette technique pour MUSP pour trois utilisateurs.

Conclusion et perspectives

L'informatique a changé profondément les aspects méthodologiques du processus de découverte dans les différents domaines du savoir. Les chercheurs ont à leur disposition aujourd'hui de nouvelles capacités qui permettent d'envisager la résolution de nouveaux problèmes. La question que l'on se pose maintenant est de comment rendre ces nouvelles capacités, auparavant limitée aux projets scientifiques les plus grands (et les plus riches), à tous les chercheurs.

Notre travail a été motivé par la conviction que cela pourrait être accompli grâce à la collaboration entre les individus et les organisations, même si dans la pratique, les participants peuvent avoir des besoins et attentes différentes (potentiellement contradictoires) quant à leur niveau de coopération.

Dans ce document qui regroupe les résultats obtenus pendant cette thèse, nous avons étudié comment la théorie de l'ordonnancement peut être appliquée afin de créer une plate-forme parallèle ou distribuée, où tous les participants se sentent toujours encouragés à partager leurs propres ressources de calcul en échange d'être en mesure d'exécuter leurs propres tâches de manière plus efficace. Chaque chapitre explore une facette différente de la façon dont les organisations s'engagent dans une collaboration et montre comment obtenir des ordonnancements qui garantissent de bons compromis entre les performances obtenues par chaque participant.

Le premier type d'interaction a été étudiée dans le Chapitre 3. Nous avons étudié ce qui se passe lorsque les participants ne peuvent pas se faire confiance mutuellement, dans le sens où ils ne peuvent pas suivre exactement les ordres donnés par un ordonnanceur centralisé. Nous avons supposé que les participants de la plate-forme sont des organisations qui agissent de façon individualiste et égoïste, capable de prendre avantage du fait qu'ils ont le contrôle complet sur leurs propres ressources.

Nous avons montré comment un ordonnanceur centralisé peut calculer un ordre d'exécution pour les tâches qui ne permet pas une situation où une organisation égoïste pourrait être tentée de modifier l'ordonnancement calculé afin d'exécuter ses propres tâches plus efficacement (au détriment de la performance des autres organisations). Néanmoins, l'algorithme est capable de garantir une 2-approximation pour le $C_{\max}$ global, tout en s'assurant qu'aucune organisation ne sera pas pénalisée pour partager ses ressources avec les autres.

L'effort pour garantir que la performance obtenue pour une organisation n'est pas dégradée (par rapport à celle que le participant pourrait obtenir en utilisant seulement ses propres ressources) est très important pour garantir que le participant aura toujours intérêt à coopérer. Mais ces garanties pour la performance individuelle imposent une pénalité sur la performance globale de la plate-forme. Dans le Chapitre 4, nous avons étudié la corrélation entre les garanties que nous pouvons offrir à chaque participant et les garanties que nous pouvons fournir à la collectivité, afin

d'améliorer la performances globale.

Nous avons présenté des algorithmes qui permettent aux participants plus altruistes de tolérer une dégradation limitée de leurs résultats en échange de meilleures performances globales. Nous avons montré que le rapport d'approximation pour le makespan global de 2 (pour le cas où aucune dégradation est tolérée) peut être amélioré à $\frac{3}{2}$ si chaque participant permet que son makespan soit au plus doublé ; et amélioré à $\frac{4}{3}$ si chaque participant permet que son makespan soit au plus triplé.

Nous avons également étudié de bornes d'inapproximabilité pour le problème. Nous avons montré que les résultats obtenus avec nos algorithmes ne sont pas trop loin de l'ensemble de Pareto du problème. L'analyse suggère que les algorithmes d'approximation avec des rapports d'approximation garanties de $\left(2; \frac{4}{3}\right)$ ou $\left(3; \frac{5}{4}\right)$ sont encore possibles. Le développement de tels algorithmes fait partie des perspectives pour des travaux futurs.

Dans toutes les situations présentées ci-dessus, le processus de prise de décision est fait par un agent centralisé qui calcule un ordonnancement qui respecte certaines règles prédéfinies. Dans le Chapitre 5, nous avons utilisé la théorie des jeux algorithmique afin d'étendre les notions d'indépendance et d'égoïsme de chaque organisation. Nous avons donné aux participants la liberté de choisir la meilleure stratégie pour leurs tâches. L'objectif était d'étudier les interactions entre les organisations indépendantes comme résultat de la tentative d'atteindre une situation d'équilibre fait par des joueurs égoïstes et rationnels.

Le modèle représente le problème comme un jeu, où chaque participant doit choisir l'organisation où chacune de ses tâches seront exécutées. L'égoïsme de chaque organisation est exprimée avec des mécanismes de coordination qui supposent que chaque organisation ira toujours exécuter d'abord ses propres tâches avant celles des d'autres organisations (qui seront ordonnancés selon une règle prédéfinie).

Si l'algorithme d'ordonnancement utilisé localement pour ordonnancer les tâches étrangères est un algorithme d'ordonnancement de liste, alors aucun équilibre de Nash ϵ-approché n'est possible pour $\epsilon < 2$. Dans ce cas, même le problème de déterminer si une instance admet un équilibre de Nash pur est co-NP dur.

Si l'algorithme d'ordonnancement utilise des informations sur l'organisation à laquelle la tâche appartient, alors on peut montrer comment faire converger le jeu vers un état d'équilibre. Nous avons présenté un algorithme qui peut calculer un

équilibre approché pur qui produit des résultats aussi proches que l'on veut d'un équilibre de Nash pur. Nous avons également montré que grâce à cette politique d'ordonnancement local, il est possible de majorer le prix de l'anarchie — le rapport entre le coût social (le $C_{\max}$ global) du pire équilibre de Nash et le coût social d'un ordonnancement optimal.

Ce modèle de théorie des jeux ouvre des nouvelles pistes de recherche, qui pourront explorer d'autres mécanismes de coordination pour le problème. Il serait intéressant d'étudier de nouveaux mécanismes de coordination menant à des résultats avec des garanties supplémentaires telles que l'équité ou d'un meilleur $C_{\max}$ global, par exemple. Une autre possibilité de recherche intéressante est d'utiliser ce modèle de théorie des jeux pour étudier une manière plus altruiste de collaboration, comme cela a été étudié dans le Chapitre 4.

Le dernier aspect de la collaboration sur les plates-formes parallèles et distribués étudiées dans ce travail est relatif à la notion de propriété des ressources qui sont partagées. Contrairement aux cas précédents — où chaque participant contribue à une partie des ressources disponibles — nous avons étudié dans le Chapitre 6 le cas où les ressources appartiennent à la communauté et personne n'a de contrôle spécial ni de privilège sur ces ressources.

Dans ce contexte, nous avons considéré un ensemble d'utilisateurs différents qui sont en concurrence pour l'exécution de leurs tâches sur un ensemble commun de ressources, afin d'atteindre la meilleure performance possible. Il n'y a pas de "meilleure solution" possible, mais un ensemble de solutions qui offrent différents compromis pour les utilisateurs.

Il y a plusieurs propriétés sur de telles solutions qui peuvent faire l'objet d'étude. La première propriété que l'on pourrait imaginer dans ce contexte est l'équité des résultats obtenus par chaque participant. Nous avons présenté une méthode générique pour optimiser des fonctions objectives monotones (comme, par exemple, l'équité) qui dépendent des arguments à optimiser. Essentiellement, la méthode énumère la frontière des solutions de meilleurs compromis entre ces arguments et sélectionne la solution qui apporte la meilleure valeur pour la fonction à optimiser.

Nous avons illustré l'utilisation de la méthode sur le problème d'ordonnancement où différents utilisateurs — intéressés à minimiser soit le makespan, soit la durée moyenne du temps d'exécution de leurs tâches — sont en concurrence pour un

nombre limité de ressources. En utilisant de nouvelles approximations de l'ensemble de Pareto pour le problème, nous avons montré comment notre méthode peut être utilisée pour trouver un ordonnancement équitable et qui offre un bon compromis entre la performance perçue par les utilisateurs.

Cette méthode peut être appliquée à une grande variété de problèmes multi-objectifs (que nous nous attendons à explorer dans des travaux futurs). Nous espérons également d'étendre les algorithmes d'approximations de l'ensemble de Pareto afin de les appliquer pour des cas plus génériques du problème.

Abstract

Computer science is deeply changing methodological aspects of the discovery process in different areas of knowledge. Researchers have at their disposal new capabilities that can create novel research opportunities. Parallel and distributed platforms composed of resources shared between different participants can make these new capabilities accessible to every researcher at every level, delivering computational power that was restricted before to bigger (and wealthy) scientific projects.

This work explores four different facets of the rules that govern how organizations engage in collaboration on modern parallel and distributed platforms. Using classical combinatorial tools, multi-objective scheduling and game-theory, we showed how to compute schedules with good trade-offs between the results got by the participants and the global performance of the platform. By ensuring fair results and guaranteeing performance improvements for the participants, we can create an efficient platform where everyone always feels encouraged to collaborate and to share its resources.

First, we study the collaboration between selfish organizations. We show how the selfish behavior between the participants imposes a lower bound on the global makespan. We present algorithms that cope with the selfishness of the organizations and that achieve good fairness in practice.

The second study is about collaboration between organizations that can tolerate a limited degradation on their performance if this can help ameliorate the global makespan. We improve the existing inapproximation bounds for this problem and present new algorithms whose guarantees are close to the Pareto set.

The third form of collaboration studied is between rational participants that can independently choose the best strategy for their jobs. We present a non-cooperative game-theoretic model for the problem and show how coordination mechanisms allow the creation of approximate pure equilibria with bounded price of anarchy.

Finally, we study collaboration between users sharing a set of common resources. We present a method that enumerates the frontier of best compromise solutions for the users and selects the solution that brings the best value for the global performance function.

Résumé

L'informatique a changé profondément les aspects méthodologiques du processus de découverte dans les différents domaines du savoir. Les chercheurs ont à leur disposition aujourd'hui de nouvelles capacités qui permettent d'envisager la résolution de nouveaux problèmes. Les plates-formes parallèles et distribuées composées de ressources partagées entre différents participants peuvent rendre ces nouvelles capacités accessibles à tout chercheur et offrent une puissance de calcul qui a été limitée jusqu'à présent aux projets scientifiques les plus grands (et les plus riches).

Dans ce document qui regroupe les résultats obtenus pendant cette thèse, nous explorons quatre facettes différentes de la façon dont les organisations s'engagent dans une collaboration sur de plates-formes parallèles et distribuées. En utilisant des outils classiques de l'analyse combinatoire, de l'ordonnancement multi-objectif et de la théorie des jeux, nous avons montré comment calculer des ordonnancements avec un bon compromis entre les résultats obtenus par les participants et la performance globale de la plate-forme. En assurant des résultats justes et en garantissant des améliorations de performance pour les différents participants, nous pouvons créer une plate-forme efficace où chacun se sent toujours encouragé à collaborer et à partager ses ressources.

Tout d'abord, nous étudions la collaboration entre organisations égoïstes. Nous montrons que le comportement égoïste entre les participants impose une borne inférieure sur le makespan global. Nous présentons des algorithmes qui font face à l'égoïsme des organisations et qui présentent des résultats équitables.

La seconde étude porte sur la collaboration entre les organisations qui peuvent tolérer une dégradation limitée de leur performance si cela peut aider à améliorer le makespan global. Nous améliorons les bornes d'inapproximabilité connues sur ce problème et nous présentons de nouveaux algorithmes dont les garanties sont proches de l'ensemble de Pareto (qui regroupe les meilleures solutions possibles).

La troisième forme de collaboration étudiée est celle entre des participants rationnels qui peuvent choisir la meilleure stratégie pour leur tâches. Nous présentons un modèle de jeu non coopératif pour le problème et nous montrons comment l'utilisation de "coordination mechanisms" permet la création d'équilibres approchés avec un prix de l'anarchie borné.

Finalement, nous étudions la collaboration entre utilisateurs partageant un ensemble de ressources communes. Nous présentons une méthode qui énumère la frontière des solutions avec des meilleurs compromis pour les utilisateurs et sélectionne la solution qui apporte la meilleure performance globale.