INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

/_/_/_/_/_/_/_/_/_/_/

THESE EN COTUTELLE INTERNATIONALE

pour obtenir le grade de

DOCTEUR DE L'Institut Polytechnique de Grenoble

et

de Polish-Japanese Institute of Information Technology

Spécialité : Informatique

préparée au Laboratoire d'Informatique de Grenoble

dans le cadre de l'École Doctorale Mathématiques, sciences et technologies de l'information, Informatique

présentée et soutenue publiquement

par

Krzysztof Rządca

Le 27 février 2008

Des modèles et des algorithmes pour la gestion des ressources dans les grilles de plusieurs organisations

DIRECTEURS DE THÈSE

Franciszek Seredyński

Denis Trystram

JURY

M. Lech Polkowski	Président
M. Jacek Błażewicz	Rapporteur
M. Roberto Di Cosmo	Rapporteur
M. Marek Tudruj	Rapporteur
M. Franciszek Seredyński	Directeur de thèse
M. Denis Trystram	Directeur de thèse
M. Witold Kosiński	Examinateur
M. Kazimierz Subieta	Examinateur

La tactique du démon c'est de toujours nous faire perdre du temps.

Marie-Dominique Philippe

Wespół zespół,
wespół zespół,
by żądz moc
móc wzmóc!

Kabaret Starszych Panów

The whole is more than the sum of its parts.

Aristotle, Metaphysics

Acknowledgements

This work would not have been possible without inspiration and help generously offered by many people. I owe a huge debt to my PhD advisors, Franciszek Seredyński and Denis Trystram, for the freedom they gave my in my research, but also for their helpful guidance whenever I was needing it. Franciszek Seredyński continually inspired, trusted and supported me during many years of our joint work. Thanks to Denis Trystram, I learned a lot about theoretical computer science, science, *hatha yoga* and life in general. *Dziękuję. Merci!*

Moltes gràcies a Francesc J. Ferri. Our joint work gave me the crucial credit allowing me to start my PhD.

I would like to thank Feryal Kamila Moulaï, Erik Saule and other friends from LIG for a warm welcome, help, advice, *des cours du français*, *la patience*, and finally many (not only scientific) discussions. Living in France was much easier thanks to you!

Precious remarks of Jaroslaw Żola helped me to formulate the model of grid that shares dedicated resources.

Fanny Pascual introduced me to the craft of designing approximation algorithms and worked with me on parallel job scheduling.

Adam Wierzbicki and Włodzimierz Ogryczak introduced me to the axiomatic theory of fairness.

Alexandru Iosup provided in the right time and in the right form the data that allowed me to make my experiments more realistic.

The support and confidence of my wife and my parents was crucial to overcome problems inherent to such long-lasting projects.

Abstract

Grids are large scale supercomputers that permit coordinated usage of resources owned and controlled by different parties. Acceptable performance of any supercomputer can be achieved only through efficient management of resources. Nevertheless, most of the associated algorithmic problems are computationally hard. A grid, by its decentralization, adds new, intriguing issues to these problems, such as ensuring fairness between the participating institutions or coping with their selfish behaviour.

The aim of this work is to study the effects of the increased decentralization in grid scheduling by means of simple mathematical modeling of the fundamental features that make grids different from classic parallel computers. We analyze the models with game theoretic approach in order to measure the consequences of decentralized decision making by selfish participants. We also employ the theory of equitable multicriteria optimization to guarantee that all the parties are treated fairly.

Our main conclusion is that grids without any form of centralized control or coordination work inefficiently. The resulting loss of performance can be proportional to the number of jobs in the system. Yet, with some centralized control and coordination, it is possible to share the pool of available resources fairly amongst participants, so no-one loses by cooperating. In this context, we propose a number of scheduling algorithms for various configurations of the grid.

Keywords: *grid, scheduling, game theory, fairness, multicriteria optimization*

Résumé

Les grappes sont des superordinateurs à grande échelle qui permettent l'utilisation coordonnée de ressources possédées et contrôlées par différentes organisations. La performance d'un super-ordinateur dépend de la gestion efficace de ses ressources, et en particulier de l'ordonnancement des tâches qu'il effectue. La plupart de ces problèmes sont des problèmes algorithmiques difficiles. Une grappe, par sa décentralisation, ajoute de nouvelles difficultés à ces problèmes, comme par exemple assurer l'équité entre les établissements participants, ou faire face à leur comportement égoïste.

Le but de ce travail est d'étudier les effets de la décentralisation sur le problème d'ordonnancement dans les grappes de calcul, et ce grâce à des modèles mathématiques simples capturant les caractéristiques fondamentales qui distinguent les grappes des ordinateurs parallèles classiques. Nous faisons notamment appel à la théorie des jeux afin de mesurer les conséquences de la prise de décision décentralisée par les participants égoïstes. Nous utilisons également la théorie de l'optimisation équitable pour garantir que toutes parties sont traitées de manière équitable.

Notre conclusion principale est que les grappes sans aucune forme de gestion ou de coordination centralisée ne fonctionnent pas efficacement. La perte de performance qui en résulte peut être proportionnelle au nombre de tâches dans le système. Cependant, avec un certain degré de coordination et une gestion centralisée, il est possible de partager de façon équitable l'ensemble des ressources disponibles entre les participants, afin que personne ne perde du fait de sa coopération. Dans ce contexte, nous proposons des algorithmes d'ordonnancement pour différentes configurations de grappes.

Mots clés : *grappes, ordonnancement, théorie des jeux, optimisation équitalbe, optimisation multicritère*

Streszczenie

Superkomputery wielkiej skali, nazywane gridami, pozwalają na skoordynowane korzystanie z zasobów znajdujących się pod kontrolą różnych instytucji. Właściwe zarządzanie zasobami kluczowe jest dla osiągnięcia wysokiej wydajności klasycznych superkomputerów. Problemy algorytmiczne związane z tym zagadnieniem mają dużą złożoność obliczeniową. Administracyjne zdecentralizowanie gridów dodaje do zagadnienia zarządzania zasobami szereg nowych, istotnych czynników, takich jak problem sprawiedliwego podziału zasobów pomiędzy uczestniczące instytucje, czy też radzenie sobie z ich egoistycznym zachowaniem.

Podstawowym celem niniejszej pracy było zbadanie, w jaki sposób organizacyjna decentralizacja gridów wpływa na wydajność systemu. Podstawowe różnice pomiędzy gridami a klasycznymi superkomputerami zostały przedstawione przy użyciu modeli matematycznych. Wykorzystując narzędzia teorii gier, zmierzyliśmy spadek wydajności związany ze zdecentralizowanym podejmowaniem decyzji przez niezależnych, egoistycznych użytkowników gridu. Korzystając z teorii sprawiedliwej optymalizacji wielokryterialnej, opisaliśmy jak powinien wyglądać sprawiedliwy podział zasobów.

Głównym wynikiem przeprowadzonych badań jest wykazanie, że całkowicie zdecentralizowane gridy działać będą niewydajnie. Spadek wydajności może być proporcjonalny do liczby zadań w systemie. Możliwe jest jednakże uzyskanie wydajnego i sprawiedliwego podziału zasobów, o ile tylko działania uczestników będą koordynowane, a ich kontrola nad lokalnymi zasobami zostanie ograniczona. Zakładając taką koordynację, w pracy proponujemy algorytmy szeregowania zadań dla różnych możliwych konfiguracji gridów.

Słowa kluczowe: *grid, szeregowanie, teoria gier, sprawiedliwość, optymalizacja wielokryterialna*

Preface

Nowadays, many computer systems gather independent individuals, who cooperate by sharing their resources. People share their photos and videos on web sites, their bookmarks in social bookmarking sites, their memories in blogs, their files in peer-to-peer systems and their computing power in volunteer computing projects. Similarly, laboratories share their computing resources in grids. The possibilities offered by hardware and software at last allow us to build such large-scale, complex systems. However, as such systems gather independent individuals, they face problems similar to the problems faced by modern democratic societies. Firstly, individuals are inherently selfish. Secondly, the access to shared resources should be equitable.

In early computer systems, sharing was forced by lack of means. Computers were expensive and the performance provided was not adequate to the needs. Then, because of the exponential drop in costs of hardware, computers became increasingly affordable. As a straightforward result, a number of isolated systems appeared. Yet, in recent years, these systems are being interconnected. Cooperation and sharing became popular in domains as different as high performance computing, on-line games or video storage. New possibilities have been created thanks to the tremendous increase of the network and computer performance, as well as the increased sophistication of software. Consequently, it is not difficult to get tools that considerably facilitate the creation of a virtual community. However, the development of such large scale systems seems to be driven more by a need, than by existing

possibilities. A well-functioning system is worth more than the simple sum of its components.

The crucial difference between early and modern systems is that nowadays members must *want* to cooperate, as opposed to being *forced* to do so in the past. Consequently, to create such incentives, the added value of the system, i.e. the difference between the value of the system and the total value of its components, must be *distributed equitably* amongst the members. A similar problem is faced by the modern human societies, in which added value is distributed amongst the citizens in a more or less equitable manner by means of complex chains of commercial relationships or social welfare systems.

The individuals that share resources in modern systems are independent. Consequently, in large scale systems, members tend not to identify themselves with other peers. It is thus natural that each member behaves *selfishly*. Each member takes every possible action that maximizes his/her gain, being aware of the current state of the system and having some expectations about actions of other players. As a result, using game theoretic terms, peers should end up in a state called *Nash equilibrium*. In human societies, this situation is equivalent to *anarchy*, a political system that permits selfish maximization of each member's gain without any form of global optimization. The *Price of Anarchy*, another game theoretical tool, can quantitatively measure the loss of performance in an anarchic system comparing to the best collaborative solution. If the Price of Anarchy is too high, some coordination mechanisms must exist in order to penalize users for their selfish behavior, and thus increase the motivation to behave in socially-optimal way. This situation is again analogous to modern democratic societies, in which partial coordination of independent citizens in achieved through formal law, or informal social conventions. Nobody is eager to pay taxes and thus to reduce his/her net income. However, everyone expects that the police and the justice system, paid by taxes, work efficiently. Consequently, taxes are enforced by law and backed up by large fines for the ones who try to dodge them.

In this thesis, we study grids, a particular class of modern computing

systems. Grids are large scale supercomputers that permit coordinated use of resources owned and controlled by different parties. The development of grids was mostly driven by large-scale scientific experiments. For instance, the Large Halidron Collider (LHC) will produce immense quantities of data that must be analyzed by and shared between physicists around the globe. Grids are currently more and more accepted both in other scientific projects and in industry.

Many working grids have shown that grids enable cooperation between the users. In EGEE grid, particle physicists from around the world jointly analyze data produced in CERN. In France, Grid'5000 project not only provides a large scale experimental platform, but also integrates parallel processing community, enabling scientists to share their knowledge. Users consider important this social effect of the grid.

It seems that grids reached their maturity concerning both the hardware and the software stack. Fast interconnection networks became available with the increased adoption of fiber-optic communication links. Many grid middleware implementations provide implementation of basic services, like large scale data transfer, remote execution of a job, or basic security tools.

Consequently, the available technologies make grids a community-like system that is facing the same problems as other communities face: selfishness of individual members and fair-sharing of the system's added value.

In this work, we employ the theory of equitable multicriteria optimization and game theory to study the problem of managing resources in grids. We prove that, similarly to other anarchic systems, grids without any form of centralized control or coordination will work inefficiently. Yet, with some level of coordination, it is possible to share the pool of available resources fairly amongst participants, so no-one loses by cooperating.

Contents

Preface **xiii**

1 Introduction **1**

1.1 Motivation 1
1.2 The Goal and the Scope of the Work 2
1.3 Tools 3
1.3.1 Multicriteria Optimization 3
1.3.2 Axiomatic Theory of Fairness 6
1.3.3 Game Theory 9
1.3.4 Approximation Algorithms 12
1.4 Related Work 12
1.4.1 Grid Systems 12
1.4.2 Scheduling Theory 15
1.4.3 Economic Approaches to Grid Resource Management . 16
1.4.4 Game Theory 18
1.5 Summary of Contributions 19

2 Multi-Organizational Grid Model **23**

2.1 Definitions and Notation 24
2.1.1 Elements of the Model 25
2.1.2 Additional Characteristics of the Model 25
2.1.3 Additional Notation 27
2.2 Performance Measures 27

2.3 Assumptions and Scope of the Model 28
2.4 Approaches for Optimization 30
2.5 The Model and Real-World Grids 32
2.6 Summary 34

3 Resource Sharing **37**
3.1 Grid Model 39
3.2 Optimization Approach 40
3.2.1 All Optimal Solutions 41
3.2.2 One Solution 48
3.3 Algorithms 52
3.3.1 Scheduling with Exhaustive Search 53
3.3.2 Scheduling with Dynamic Programming 55
3.3.3 Equitable Walk (EW) 59
3.4 Game-Theoretic Model 62
3.5 Non-cooperative Game 64
3.6 Constrained Optimization Approach 67
3.7 Experiments 68
3.7.1 Methods 68
3.7.2 Exact Algorithms 71
3.7.3 Greedy Heuristics 81
3.8 Equitable Criteria 92
3.9 Summary 93

4 Divisible Load Balancing **95**
4.1 Grid Model 97
4.2 Cost of Computation 98
4.3 Averaging Load Balancing: A Fair Optimization Perspective . 101
4.4 A Two-Player Sender-Receiver Game 103
4.5 Averaging Load Balancing Game 105
4.5.1 Description of Algorithms 106
4.5.2 Experimental Analysis 109

4.6 Load-Balancing Based on Queue Length . . . 112
4.7 Bounded, Iterative Load Balancing . . . 115
4.7.1 BILB Algorithm Description . . . 115
4.7.2 Algorithm Analysis . . . 117
4.7.3 Strategies . . . 118
4.7.4 Experimental Analysis . . . 119
4.8 Discussion . . . 125
4.9 Summary . . . 127

5 Rigid Load Balancing 129
5.1 Grid Model . . . 130
5.2 Scheduling on One Resource . . . 131
5.2.1 List Scheduling . . . 132
5.2.2 Highest First (HF) Job Order . . . 133
5.3 Optimization Approach . . . 135
5.3.1 Motivation . . . 135
5.3.2 Problem Complexity . . . 136
5.3.3 Approximation Ratio of List Scheduling . . . 136
5.3.4 Equitable Optimization . . . 138
5.4 Game-Theoretic Approach . . . 140
5.5 Multi-Organization Scheduling . . . 143
5.5.1 Impact of the Constraint . . . 144
5.5.2 Multi-Organization Load Balancing Algorithm . . . 144
5.5.3 Iterative Load Balancing Algorithm (ILBA) . . . 151
5.6 Experiments . . . 154
5.6.1 Methods . . . 155
5.6.2 Results . . . 156
5.7 Summary . . . 160

6 Summary and Conclusions 161

7 Résumé étendu en français **165**

7.1 Introduction 166

7.2 Modèle de grilles Multi-Organizations 168

7.3 Partage de Resources 170

7.4 Tâches Divisibles 172

7.5 Tâches parallèles 175

7.6 Conclusions 180

Bibliography **185**

Chapter 1

Introduction

1.1 Motivation

Grids introduce a number of new, fascinating issues to the problem scheduling and resource management. At the same time, many important applications require computing performance that nowadays can be delivered only by grids. In this section, we describe in detail our motivation to study grid resource management and scheduling.

Many high performance computing applications must on grids, as the quantity of data to analyze, or the sophistication of models rises faster than the Moore's law. Consequently, those applications are not feasible to compute on any single supercomputer.

At the same time, grids are a prominent example of a modern computer system based on sharing. Thus, our methods and conclusions can be adapted to related fields, such as peer-to-peer systems.

The grid hardware and software reached a certain level of maturity that allows us to analyze resource management on a certain level of abstraction.

The problem of resource management in grids is difficult. Even on the level of a single supercomputer, resource management is a key to achieve acceptable performance. At the same time, most of the associated algorithmic

problems are computationally hard. A grid, by its decentralization, adds new, intriguing issues, such as ensuring fairness between the participating institutions or coping with their selfish behaviour.

Existing approaches to grid resource management have some important drawbacks. A number of earlier works ignored the organizational decentralization of the grid. Broker-based approaches are unable to model the grid as a whole, as they rely on externally-set prices to control the access to the resources. Finally, in economic approaches, although the price forms part of the model, it is usually expressed in some kind of currency, a fact that may negatively influence sharing and cooperation, and thus the community of users participating in the grid.

1.2 The Goal and the Scope of the Work

The aim of this work is to study the effects of the increased decentralization in grid scheduling by means of simple mathematical modeling of the fundamental features that make grids different from classic parallel computers.

Grids are composed of resources being under different administrative domains [Foster, 2002]. We claim that grids must be studied with approaches capable of seeing this organizational heterogeneity. We propose two kinds of mathematical frameworks: *game theory* in systems in which owners of individual resources have complete control over their machines; *equitable multi-criteria optimization* in more centralized systems, in which a grid resource manager can schedule jobs on resources, yet different stakeholders must be treated fairly.

We do not propose a fully-featured software that schedules jobs in grids for a number of reasons. Firstly, a number of such programs is already available [Berman et al., 2003, Dumitrescu et al., 2005, Kurowski et al., 2004a] [Capit et al., 2005]. Secondly, at the current level of development of grids, a scheduler cannot be used in systems other than exactly the one for which it has been created, as the software stack, the architecture and even the underly-

ing assumptions and predicted usage scenarios differ. Thirdly, the relatively newness of grids and, consequently, the lack of widely adopted validation techniques causes scientific validation of such programs hard. Instead, we analyze our models with means similar to these used in scheduling theory. We provide a theoretical analysis of worst-case scenarios, lower bounds on performance of algorithms and simulation results, where appropriate.

1.3 Tools

In this section we provide a summary of the theoretical tools used in this work. We discuss multicriteria optimization in Section 1.3.1, the axiomatic theory of fairness in Section 1.3.2, game theory in Section 1.3.3 and approximation algorithms in Section 1.3.4.

1.3.1 Multicriteria Optimization

Multicriteria optimization problems are specific optimization problems in which more than one function is to be optimized. Therefore, before defining multicriteria optimization approaches, we consider a standard, monocriterion optimization problem.

Definition 1.1. *[Papadimitriou and Steiglitz, 1998] An* instance of an optimization problem *is a pair (X, f) where X is any set, the domain of feasible points; f is the cost function, a mapping $f : X \to \mathbb{R}^1$. The problem is to find $x^* \in X$ for which $\forall x \in X : f(x^*) \leq f(x)$.*

In multicriteria optimization, there are a number of cost functions $f_1, \ldots, f_N$ to be optimized, corresponding to different criteria.

Definition 1.2. *An* instance of an multicriteria optimization problem *(IMOP) is a pair $(X, \boldsymbol{f})$ where X is any set, the domain of feasible points; $\boldsymbol{f} = [f_1, \ldots, f_N]$ is a vector-function that maps X into the criterion space of* outcome vectors *$Y = \mathbb{R}^N$; f_i is the ith criterion, a mapping $f_i : X \to \mathbb{R}^1$.*

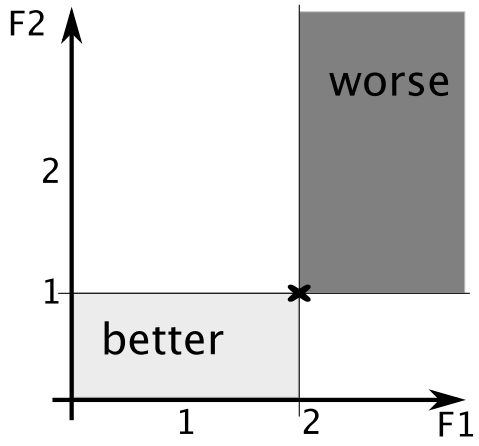


Figure 1.1: Relation of Pareto-dominance. Point $[2, 1]$ (black cross) is dominated by all the points in the light-gray rectangle where both criteria have lower values. $[2, 1]$ dominates all the points in the dark-gray rectangle where both criteria have higher values.

The problem is to find $\boldsymbol{x}^*$ *that minimizes all the objective functions* $\boldsymbol{f} = [f_1, \ldots, f_N]$.

Note that the above definition specifies that we are interested in optimizing all the objective functions. We have to assume a certain *solution concept* [Kostreva et al., 2004] that precises what is the meaning of minimization of multiple functions. Usually, a optimality is defined as Pareto-optimality, which, in turn, is defined through the relation of Pareto-dominance.

Definition 1.3. *An asymmetric, irreflexive and transitive relation* $(\mathbb{R}^N, \mathbb{R}^N, \prec_P)$ *is the* relation of Pareto-dominance *iff* $\boldsymbol{y}' \prec_P \boldsymbol{y}'' \Leftrightarrow \forall_{1 \leq i \leq N} : y'_i \leq y''_i \wedge \exists_{1 \leq j \leq N} : y'_j < y''_j$.

Informally, $\boldsymbol{y}' \prec_P \boldsymbol{y}''$ iff $\boldsymbol{y}'$ is as good as $\boldsymbol{y}''$ regarding all criteria and better regarding at least one (see Figure 1.1).

Definition 1.4. *If* $\boldsymbol{y}' \prec_P \boldsymbol{y}''$, $\boldsymbol{y}''$ *is* Pareto-dominated *by* $\boldsymbol{y}'$.

Definition 1.5. $\boldsymbol{y}'$ *is* Pareto-optimal *iff it is not dominated by any other point, i.e.* $\nexists \boldsymbol{y}'' : \boldsymbol{y}'' \prec_P \boldsymbol{y}'$.

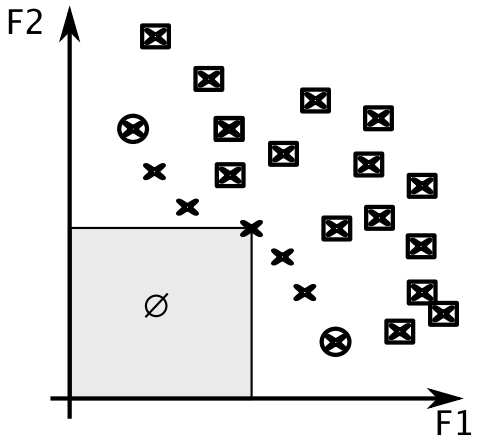


Figure 1.2: Pareto-dominance and Pareto-optimality. All the feasible solutions (crosses) of an instance plotted in two-dimensional criteria space $[f_1, f_2]$. Boxes denote Pareto-dominated points. Circles denote points that are optimal for one of the criteria. For each of Pareto-optimal points $[y_1^*, y_2^*]$, there are no points in a rectangle spanning between $[0, 0]$ and $[y_1^*, y_2^*]$ (denoted by a gray rectangle).

We use the notion of Pareto-optimity to define a solution of an IMOP.

Definition 1.6. *A feasible solution* $\boldsymbol{x}^* \in X$ *is* Pareto-optimal solution *to an instance of the multicriteria optimization problem* $(X, \boldsymbol{f})$ *if the resulting outcome vector* $\boldsymbol{f}(\boldsymbol{x}^*)$ *is not Pareto-dominated, i.e.* $\nexists \boldsymbol{x} \in X : \boldsymbol{f}(\boldsymbol{x}) \prec_P \boldsymbol{f}(\boldsymbol{x}^*)$.

Usually, there is a huge number of Pareto-optimal solutions to an IMOP. In order to solve an IMOP completely, all such solutions must be enumerated. Figure 1.2 illustrates this concept.

Definition 1.7. *The* solution *to an instance of the multicriteria optimization problem* $(X, \boldsymbol{f})$ *is the set of all Pareto-optimal solutions* $X^* \subseteq X$. *Each member is a Pareto-optimal solution* $\forall \boldsymbol{x}^* \in X^* \nexists \boldsymbol{x} \in X : \boldsymbol{f}(\boldsymbol{x}) \prec_P \boldsymbol{f}(\boldsymbol{x}^*)$. *Moreover, all Pareto-optimal solutions belong to* X^*, *i.e.* $\forall \boldsymbol{x} \in X :$ *Pareto-optimal*$(\boldsymbol{x}) \Leftrightarrow \boldsymbol{x} \in X^*$.

1.3.2 Axiomatic Theory of Fairness

Although extensively studied [Young, 1994], fairness is a complex concept that depends much on, among others, cultural values, precedents, and the context of the problem. However, a precise definition is needed to use the concept of fairness in optimization. In this work, fairness is identified with *distributive fairness*, a meaning narrower than social justice [Rawls, 1971].

Distributive fairness is usually related to the question of distribution of some goods, resources or costs, be it kidneys for transplantation, parliament mandates, or the costs of water and electricity. In distributive fairness, we assume that each criterion corresponds to the quantity of goods assigned to one of independent agents. Thus, the criteria are uniform, which permits direct comparisons between their values.

Distributive fairness can be precisely described by axioms, which define a relation of *equitable dominance* [Kostreva et al., 2004]. Similarly to the relation of Pareto-dominance, equitable dominance characterizes the set of solutions to a multicriteria optimization problem.

Definition 1.8. *[Kostreva et al., 2004] An asymmetric, irreflexive and transitive relation* $(\mathbb{R}^N, \mathbb{R}^N, \prec_e)$ *is the* relation of equitable dominance *iff:*

anonymity $\prec_e$ *ignores the ordering of the outcome values. Given any permutation* π *of* $(1, \ldots, N)$, $\neg\big(\ [y_{\pi(1)}, y_{\pi(2)}, \ldots, y_{\pi(N)}] \prec_e [y_1, y_2, \ldots, y_N]\ \big)$ *and* $\neg\big(\ [y_1, y_2, \ldots, y_N] \prec_e [y_{\pi(1)}, y_{\pi(2)}, \ldots, y_{\pi(N)}]\ \big)$

monotony *a outcome that is Pareto-dominated is also equitably-dominated i.e.* $\boldsymbol{y}' \prec_P \boldsymbol{y}'' \Rightarrow \boldsymbol{y}' \prec_e \boldsymbol{y}''$

principle of transfers *A transfer of any small amount from an outcome to any other relatively worse-off outcome results in a more preferred outcome vector, i.e.* $y_i > y_j \Rightarrow [y_1, \ldots, y_i - \epsilon, \ldots, y_j + \epsilon, \ldots, y_N] \prec_e [y_1, \ldots, y_i, \ldots, y_j, \ldots, y_N]$

The notion of *equitable dominance*, *equitable efficiency* and *equitably-optimal solution* are defined similarly as in Section 1.3.1.

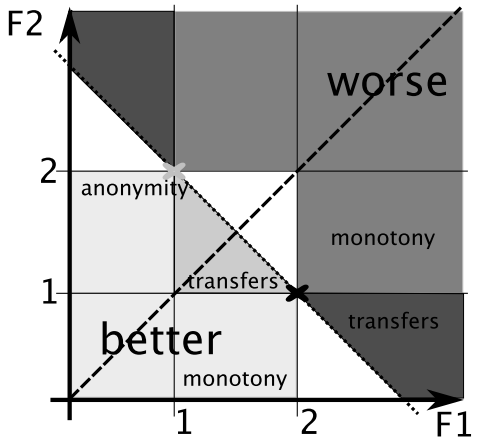


Figure 1.3: Relation of equitable dominance. Point $[2, 1]$ (black cross) and its symmetric counterpart $[1, 2]$ (gray cross). The plot is symmetrical regarding $f_1 = f_2$ dashed line because of the anonymity. $[2, 1]$ is dominated by all the points in the light-gray rectangle areas because of the principle of monotony. $[2, 1]$ is dominated by the points in the light-gray triangle because of principle of transfers: there, the improvement of the worse-off criterion is greater than the degradation of the other criteria. Areas painted in dark-gray are equitably-dominated by $[2, 1]$: the rectangle areas because of monotony; the trapezoid areas because of the principle of transfers (the degradation of the worse-off criterion is greater than the improvement of the better-off criterion)

Definition 1.9. *If* $\boldsymbol{y}' \prec_e \boldsymbol{y}''$, $\boldsymbol{y}''$ *is* equitably-dominated *by* $\boldsymbol{y}'$.

Definition 1.10. $\boldsymbol{y}'$ *is* equitably efficient *iff it is not equitably-dominated by any other point, i.e.* $\nexists \boldsymbol{y}'' : \boldsymbol{y}'' \prec_e \boldsymbol{y}'$.

Definition 1.11. *A feasible solution* $\boldsymbol{x}^* \in X$ *is* equitably-optimal solution *of an IMOP* $(X, \boldsymbol{f})$ *if the resulting outcome vector* $\boldsymbol{f}(\boldsymbol{x})$ *is equitably-efficient, i.e.* $\boldsymbol{x}^* \in X^* \nexists \boldsymbol{x} \in X : \boldsymbol{f}(\boldsymbol{x}) \prec_e \boldsymbol{f}(\boldsymbol{x}^*)$.

The axioms of anonymity, monotony and principle of transfers have straightforward interpretation (Figure 1.3 presents these axioms for a bi-objective problem). Anonymity states that no criteria is preferred a priori, e.g. a solution $\mathbf{y}' = [2, 1]$ is equally good as a solution $\mathbf{y}'' = [1, 2]$. Through monotony we state that equitably-optimal solutions must be Pareto-optimal, e.g. $\mathbf{y}' = [2, 1]$ is preferred to $\mathbf{y}'' = [2, 2]$. Finally, the principle of transfers

states that a solution improving a worse-off outcome, at the expense of deteriorating a better-off outcome is preferred, e.g. $\mathbf{y}' = [2, 1]$ is preferred to $\mathbf{y}'' = [3, 0.5]$.

Note that equitably efficient solutions are a subset of Pareto-optimal solutions to an IMOP.

An Instance of Equitable Multicriteria Optimization Problem (IEMOP) can be formulated on the basis of an IMOP, such that the Pareto-optimal solutions to the IEMOP will be equitably efficient solutions to the IMOP. The original vector of outcomes of IMOP is transformed into a vector of *cumulative ordered outcomes*.

The construction of IEMOP is as follows. First, introduce the ordering map $\boldsymbol{\theta} : R^N \to R^N$ such that $\boldsymbol{\theta}(\mathbf{y}) = [\theta_1(\mathbf{y}), \theta_2(\mathbf{y}), \ldots, \theta_N(\mathbf{y})]$. $\boldsymbol{\theta}$ is a permutation of $\{1, \ldots, N\}$ that sorts an outcome vector $\mathbf{y}$ according to non-increasing values of individual outcomes, i.e. $\theta_1(\mathbf{y}) \geq \theta_2(\mathbf{y}) \geq \cdots \geq \theta_N(\mathbf{y})$.

Next, apply a linear cumulative map to $\boldsymbol{\theta}(\mathbf{y})$, resulting in the *cumulative ordering map* $\bar{\boldsymbol{\theta}}(\mathbf{y}) = [\bar{\theta}_1(\mathbf{y}), \bar{\theta}_2(\mathbf{y}), \ldots, \bar{\theta}_N(\mathbf{y})]$ defined as $\bar{\theta}_k(\mathbf{y}) = \sum_{i=1}^{k} \theta_i(\mathbf{y})$. The coordinates of vector $\bar{\boldsymbol{\theta}}(\mathbf{y})$ express, respectively: $\bar{\theta}_1$ the largest outcome, $\bar{\theta}_2$ the total of the two largest outcomes, $\bar{\theta}_3$ the total of the three largest outcomes, ..., $\bar{\theta}_N$ the total of all outcomes.

The following theorem unites the relations of equitable- and Pareto-dominance.

Theorem 1.1. *[Kostreva et al., 2004] Outcome vector* $\mathbf{y}' \in \mathbb{R}^N$ *equitably dominates* $\mathbf{y}'' \in \mathbb{R}^N$*, iff* $\bar{\boldsymbol{\theta}}(\mathbf{y}') \prec_P \bar{\boldsymbol{\theta}}(\mathbf{y}'')$.

Note that Theorem 1.1 permits to express equitable efficiency for IMOP in terms of the Pareto-optimality for the IEMOP.

Corollary 1.1. *A feasible solution* $x \in X$ *is an equitably efficient solution of the IMOP* $(X, \mathbf{f})$*, iff it is a Pareto-optimal solution of the IEMOP* $(X, \bar{\boldsymbol{\theta}})$.

1.3.3 Game Theory

Game theory [Osborne, 2004, Osborne and Rubinstein, 1994] studies mathematical games, that is situations in which many independent agents take decisions. For each agent, the result of the game depends not only on the decision made by the agent, but also by the other agents.

The agents participating in the game are called *players*. The decision player P_k makes is called a *strategy* σ_k used by the player. Finally, for each player P_k, the result of the game is called the *outcome* u_k. The outcome is a function of player's strategy σ_k, but also of strategies of other players $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_N]$. With loss of generality, we assume that u_k are to be minimized.

Note that in this work we consider mostly so-called *pure strategies*, in which the decision of the player is deterministic. Thus, for brevity, we use the word *strategy* for *pure strategy*. In contrast, a *mixed strategy* is defined as a probability distribution over the set of pure strategies of each players, i.e. the set of decisions available to each player. In [Osborne and Rubinstein, 1994], a pure strategy is called an action in this context.

The players are assumed to be *selfish*, i.e. concerned only with optimizing their own outcomes. The players are also *rational*. The choice of the player's strategy depends only on the optimization of the player's outcome. Furthermore, the players *reason strategically*. While making decision, a player takes into account the decisions he/she expects other players will take.

In this thesis we study *strategic games* (also called games in normal form) [Osborne and Rubinstein, 1994]. In such games, all players chose their strategies at the same time. Thus, while deciding, a player cannot observe the strategies chosen by others.

The following definition formally introduces a strategic game in a form used in this thesis (comparing to [Osborne and Rubinstein, 1994] we use outcome functions and not preference relations).

Definition 1.12. *A strategic game is a tuple* $(\mathcal{P}, \boldsymbol{\Sigma}, \mathbf{u})$ *that consists of:*

Table 1.1: The structure of payoffs in Prisoner's Dilemma [Osborne and Rubinstein, 1994]. The payoff specify the number of years a player will spend in prison. The payoff of the row player is in the top left corner, the payoff of the column player is in the bottom right corner.

	to confess	not to confess
to confess	3 3	0 4
not to confess	4 0	1 1

- *a finite set* $\mathcal{P} = \{P_1, \ldots, P_N\}$ *of players;*
- *for each player* $P_k \in \mathcal{P}$*, set of available strategies* $\Sigma_k = \{\sigma_k\}$*,* $\boldsymbol{\Sigma} = \{\Sigma_k\}$*;*
- *for each player* $P_k \in \mathcal{P}$*, an outcome function* $u_k : \boldsymbol{\Sigma} \to \mathbb{R}$

Table 1.1 presents the structure of payoffs in a strategic game called the Prisoner's Dilemma (PD). In PD, each of two players has two possible strategies, $\Sigma_k = \{\text{to confess}, \text{not to confess}\}$. The payoffs u_k are symmetric and specify the number of years a player will spend in prison. For instance, if both players chose to to confess, each of them will spend 3 years in prison. If one player confesses while the other stays silent, the former will be freed and the latter will spend 4 years in prison.

Definition 1.13. Nash Equilibrium (NE) *is a profile of players' strategies* $\boldsymbol{\sigma}^{NE} = [\sigma_1^{NE}, \ldots, \sigma_N^{NE}]$ *such that no player has an incentive to unilaterally change his/her strategy. Given the NE strategies of other players, each player optimizes his/her outcome by playing a NE strategy.*

$$\forall k \; \forall \sigma_k \neq \sigma_k^{NE} : u_k(\sigma_1^{NE}, \ldots, \sigma_k, \ldots, \sigma_N^{NE}) \geq u_k(\sigma_1^{NE}, \ldots, \sigma_k^{NE}, \ldots, \sigma_N^{NE})$$

A game is not guaranteed to always end in a NE. However, NE tries to capture players' behavior in a steady state of the game. It is thus usual to analyze a game by analyzing the payoffs of players in NE.

For instance, in PD the unique Nash equilibrium is (to confess, to confess).

Nash Equilibrium can be not optimal, i.e. there might exist a profile of strategies resulting in a more preferred outcome for each player (equivalently, the outcome vector of a NE is not necessary Pareto-optimal). For instance, in PD $\mathbf{u}\big((\text{not to confess}, \text{not to confess})\big) \prec_P \mathbf{u}\big((\text{to confess}, \text{to confess})\big)$.

In order to compare the performance of the whole system between two profiles of strategies $\boldsymbol{\sigma}^1$ and $\boldsymbol{\sigma}^2$, classic (utilitarian) approaches compare sums of outcomes of individual players. $\boldsymbol{\sigma}^1$ is preferred to $\boldsymbol{\sigma}^2$ iff $\sum_k u_k(\boldsymbol{\sigma}^1) < \sum_k u_k(\boldsymbol{\sigma}^2)$. Such an aggregation is meaningful only if outcome functions are *cardinal* [Samuelson and Nordhaus, 1998], and their values are directly comparable which each other. As this assumption is considered too strong in general, a so-called *system goal* can be defined, as a function of players' strategies $u_g(\boldsymbol{\sigma})$. Using $u_g(\boldsymbol{\sigma})$, $\boldsymbol{\sigma}^1$ is preferred to $\boldsymbol{\sigma}^2$ iff $u_g(\boldsymbol{\sigma}^1) < u_g(\boldsymbol{\sigma}^2)$. Note that this notion includes utilitarian performance, that can be defined as $u_g^{UTIL}(\boldsymbol{\sigma}) = \sum_k u_k(\boldsymbol{\sigma})$.

Definition 1.14. *The* social optimum *is the profile of strategies $\boldsymbol{\sigma}^{SOC}$ that optimizes the system goal u_g, i.e.*

$$\forall \boldsymbol{\sigma} \; u_g(\boldsymbol{\sigma}^{SOC}) \leq u_g(\boldsymbol{\sigma}).$$

The *Price of Anarchy (PoA)* [Koutsoupias and Papadimitriou, 1999] is a quantitative measure of inefficiency of the worst NE. PoA can be interpreted as the game-theoretical equivalent of worst-case performance ratio of an algorithm (see the next section).

Definition 1.15. *The* Price of Anarchy (PoA) *is the maximum ratio between the system goal in a NE and the socially-optimal result:*

$$PoA = \frac{\max_{\boldsymbol{\sigma}^{NE}} u_g(\boldsymbol{\sigma}^{NE})}{u_g(\boldsymbol{\sigma}^{SOC})}.$$

The max function in the numerator corresponds to the fact that there might be more than one Nash equilibrium.

1.3.4 Approximation Algorithms

Many optimization problems are NP-hard [Garey and Johnson, 1979], thus it is not possible to solve them optimally by a polynomial algorithm, unless $P = NP$. However, many problems become considerably easier if the requirement of optimality is relaxed. Instead of searching for *the* optimal solution, the relaxed problem consists of returning *any* solution that is at most ρ times worse than the optimal one.

Approximation algorithms [Hochbaum, 1997] are a class of polynomial optimization algorithms for NP-hard problems. An approximation algorithm has a guarantee on its worst-case performance, called the approximation ratio ρ.

Definition 1.16. *The approximation ratio ρ of an algorithm $\mathcal{A}$ for a minimization problem $\mathcal{P}$ is the minimum r such that for every problem instance $(X, f) \in \mathcal{P}$ the cost $f(x^{\mathcal{A}})$ of the solution $x^{\mathcal{A}}$ returned by $\mathcal{A}$ is at most r times larger than the cost $f(x^*)$ of the optimal solution x^*, i.e.*

$$\rho = \inf \left\{ r \geq 1 : \forall (X, f) \in \mathcal{P} : \frac{f(x^{\mathcal{A}})}{f(x^*)} \leq r \right\}.$$

1.4 Related Work

Grid systems, being one of the most complex computer systems ever built, span a number of disciplines of computer science. Consequently, the related work is fairly broad. We start the survey with grid systems in general in Section 1.4.1, then we review "classic" grid scheduling (Section 1.4.2) and two specific approaches that take into account distributed nature of the grid: free market (Section 1.4.3) and game theory (Section 1.4.4).

1.4.1 Grid Systems

The grid [Foster and Kesselman, 2004] is a computer system that allows coordinated usage of resources that are under various administrative domains.

In this Section we discuss the main differences between the grid and conventional supercomputing systems. We focus in this thesis on the administrative and social challenges caused by the novel features of the grid. Nevertheless, we briefly present the most popular software (middleware) that provides basic functionality of the grid. Finally, we present two projects providing grid infrastructure that form the context of our work: EGEE [EGEE, 2007] and Grid'5000 [Bolze et al., 2006].

According to the most commonly used definition [Foster, 2002], a grid is a system that fulfills the following assumptions:

- Resources are not subject to centralized, administrative control.
- Standard, open protocols are used.
- Non-trivial Quality-of-Service are delivered.

The first point defines that the system is distributed not only in computer science meaning (e.g. non-zero latencies), but also administratively. The second point requires that the grid, rather than being a single application, creates an architecture on which various applications can be run. Finally, the third point guarantees that the resources are shared in coordinated way, and thus the utility of the whole system is greater than the sum of its parts.

Existing grid projects are based on middleware, software that provides the basic functions of the system. Currently, a number of grid middleware projects are competing. Basically, there are two main architectural paradigms for the middleware: Service-oriented grids and lightweight grids.

Service-oriented paradigm [Foster and Kesselman, 2004] describes the grid as a Service-Oriented Architecture (SOA). A service encapsulates not only traditional, physical resources (such as computer or a storage system), but also databases, software licenses, data transfers, etc. SOA defines uniform interfaces for various services. The access to a service is granted through Service-Level Agreements (SLAs), a trade contract between the client and the resource provider that specifies in detail the parameters of the resource. The Globus Toolkit from version 4 [Foster, 2005] follows this approach.

Service-oriented paradigm has been widely criticized for its complexity: large overhead to access the resources and difficulties for service providers and developers. Lightweight grid middleware projects seek to provide tools for a certain community. These projects focus usually on high-performance computing, i.e. executing computationally- or data-intensive jobs over large set of heterogeneous resources. Lightweight grid middlewares include gLite [gLite, 2007] (used in [EGEE, 2007]), CiGri [CiGri, 2007, Calas et al., 2005] and OurGrid [Andrade et al., 2003].

In this thesis we focus, however, on the economical, or even sociological challenges caused by the fact that the grid crosses administrative boundaries. In this meaning, a grid is also a community of users or providers and sets of formal rules and informal "good manners". Consequently, we describe two important grid projects, EGEE and Grid'5000, from an administrative point of view.

EGEE [EGEE, 2007] provides stable, production-level computing platform for computationally- and data-intensive applications. The system has very large scale, as it unites resources from over 200 sites providing, in total, over 30 000 processors. Users of the grid are federated in Virtual Organizations (VOs). Each VO groups users with common interests (e.g. computational chemistry). VOs are independent from existing (physical) institutions such as universities or laboratories. EGEE enables VOs to use resources granted by providers. Each provider decides which VOs can use its resources and it is obliged to manage its resources daily by e.g. fixing defect nodes, or installing new versions of middleware. However, providers do not have any specific rewards for granting access to their resources.

Grid'5000 [Bolze et al., 2006] is a French national grid initiative that provides a new kind of scientific instrument for computer scientists. Grid'5000 makes it possible to test distributed systems in settings very close to the reality, at the same time providing rigorous scientific environment, allowing, for instance, to repeat an experiment in exactly the same settings. Each of 17 laboratories participating in the project grants the whole community access

to the local resources. At the same time local users can run their experiments on the whole platform. Neither users, nor their host laboratories have to pay to access the non-local resources. However, there exists a number of more or less informal rules of *savoir-vivre*, such as running large-scale experiments preferably during nights and weekends, or not submitting more than 2 advance reservations.

1.4.2 Scheduling Theory

The mainstream of the current research on scheduling and resource management [Feitelson et al., 2005] concerns systems in which the performance of all jobs is optimized. Usually, a common metric (e.g. the maximum completion time) is optimized and thus all the jobs are treated in approximately equal manner (sometimes weights are used to express their relative importance). This approach is justified by the fact that the system has one owner (an organization) which is able to enforce policies on the local users. Load-balancing can be achieved in such systems by *Work Stealing* (WS). Each time a processor becomes free, it connects to its (loaded) neighbors in order to take off some of their load. It can be proved that WS delivers good performance (under some hypotheses) when the makespan is optimized in an on-line system [Acar et al., 2002].

Nevertheless, the main difference between a grid and a large-scale supercomputer is that a grid is, by its definition [Foster, 2002], formed by resources controlled by different administrative entities. In the context of grid computing, *multi-criteria* approaches may be used. Different criteria either express performance of different jobs [Marchal et al., 2005] (or sets of jobs as in [Agnetis et al., 2004]), (*centralized scheduling*), or different factors, such as completion time and cost of one job [Kurowski et al., 2004b] (*broker-based scheduling* [Berman et al., 2003]). A scheduling algorithm is expected to deliver Pareto-optimal solutions. The main disadvantage of centralized scheduling is that it requires a centralized control over all the resources and all the jobs. Given the action enforced by the scheduler on the others, a

user does not have guarantee that his/her solution is optimal. The main drawback of broker-based scheduling is that optimizing the execution of one job independently of the others might not necessarily lead to good results in grids with many users trying to execute their jobs in parallel. Actions of each user influence the state of the environment and consequently the parameters of functions optimized by other users' brokers. Therefore, users might be tempered to influence the environment in order to create most favorable conditions for their job. Optimization alone is not able to model such feedback loops.

1.4.3 Economic Approaches to Grid Resource Management

Grid economic approaches [Buyya et al., 2005] [Wolski et al., 2003] analyze grid resource management by means of market economy. Each resource has a monetary cost for its usage; each user has a budget to spend on the execution of his/her jobs.

The simplest approach to match users and resources is to organize auctions [Buyya et al., 2005] each time a user is looking for resources for the execution of his/her job. In such an auction resources of a requested type compete for the job. However, most studies proposing auctions as a way of trading resources leave many important questions unanswered. Especially, it is not clear how should the bids be formulated and how should they react to previous auctions. Additionally, the concept of the auction itself has some specific drawbacks, concerned with the unpredictability of prices. For an user, it is hard to balance his/her budget between a number of different jobs (although there exists a mechanism of combinatorial auction, in its general form it is computationally expensive to implement [Walsh et al., 2001]). For an owner, who considers investing in the costly infrastructure, it is hard to calculate the expected return of investment.

Commodity markets for trading the access to the resources [Kenyon and Cheliotis, 2004], similar to markets for e.g. soya bean, seem to

be more stable [Wolski et al., 2001], as a large number of buyers and sellers of homogeneous good is centrally matched. However, computational power is not a trivial commodity, because both resources and applications are heterogeneous. Usually, an application needs a certain operating system, libraries, system architecture, size of RAM locally available. Moreover, its performance depends heavily on various parameters, like e.g. speed of interconnection network [Gruber et al., 2006]. Heterogeneity divides the computational market into many sub-markets, in which one player is able to manipulate the price. Consequently, the resulting market is far from the *perfect competition* principle [Samuelson and Nordhaus, 1998], the fundamental assumption of free-market economy, and its efficiency cannot be guaranteed. In addition, commodity markets (with derivative tools, such as futures or options, players' strategies etc.) are complex instruments. Considering that grids are already one of the most complicated computer systems ever created, the introduction of another complex system to manage their resources seems to be both risky and, as we show in this work, not always necessary.

Moreover, there are some disadvantages of the concept of "money" itself. Firstly, "money" requires a centralized entity, a form of a grid bank, to keep track of it. Secondly, a recent psychological study [Vohs et al., 2006] has shown that money has a negative impact on cooperation. In a number of experiments, participants performed an irrelevant activity during which they were exposed to the concept of money (represented by e.g. a screen-saver with floating banknotes). Then, the participants were asked to perform a collaborative task (like helping to pick up pens that "accidentally" fell or preparing a place for a discussion). Comparing to the control group, the participants who were exposed to "money" were less willing to help others and worked longer alone before asking for help.

Some semi-market approaches were also proposed. The *Network of Favors*, implemented in the OurGrid project [Andrade et al., 2003], implements a kind of barter trade. A grid job is executed if there are no local jobs, but it is canceled if a local job appears. Through this approach a site can thus chose

a grid job to be executed, but it cannot solve the general problem whether to execute a grid job. There are also no guarantees on the finish time of a grid job, as a grid job is suspended when a new local job appears.

1.4.4 Game Theory

Approaches presented in this section analyze the problem of grid resource management by means of game theory.

A number of papers focus on creating rules to avoid free-riders, users who gain from the system without contributing [Ranganathan et al., 2004] [Golle et al., 2001, Ghosal et al., 2005, Buragohain et al., 2003]. There were some theoretical works on this subject, in which game theory models were formed to model users' behavior. Unfortunately, the models proposed captured mainly steady-state behavior of the participants, as early peer-to-peer systems were focused on file-sharing. In file-sharing (and similarly, in selfish caching problem [Chun et al., 2004], [Laoutaris et al., 2005]), it is possible to formulate a static (not depending on time) utility function, which can express the gain each user gets from the system in a closed formula. In the context of distributed resource management, such a solution is proposed in [Volper et al., 2004] and focuses on maintaining good relationships of a node with its neighbors by accepting neighbors' jobs to be executed on the node and therefore increasing the probability that the node's jobs will be accepted by its neighbors in the future. However, the need for computational power is usually highly time-dependent with peaks of activity followed by periods of considerably lower needs. When the local demand is high, the ability to use foreign computational power is valuable and, at the same time, the execution of foreign jobs becomes costly. Yet, foreign resources are almost useless when the local demand drops below some threshold.

[Kwok et al., 2005] proposes a model in which individual clusters (for example belonging to different departments of a university) are visible as a one site in the grid. The model assumes that a job has been already accepted for the execution by the site. [Kwok et al., 2005] studies which cluster from

that site should eventually execute the job. Our model concerns the assumption which [Kwok et al., 2005] made a priori, as we are studying the problem whether a site should accept a job coming from another site.

There exist also some previous work where the infrastructure was considered a common property. [Angel et al., 2006, Liu et al., 2005] consider jobs as players in the game. Each job minimizes its completion time through choosing one of available processors. [Grosu and Chronopoulos, 2005] considers steady state scheduling in which users, players in the game, send parts of their load to one of the available processors. Similarly, each user minimizes the expected response time of his/her jobs.

Another related field of research is mechanisms design for the load balancing problem [Grosu and Chronopoulos, 2004]. Generally, a mechanism can balance the agent's cost of invoking an action by a payoff paid to the agent [Fudenberg and Tirole, 1991]. This enables the mechanism to control the global behavior of the system. For instance, [Grosu and Chronopoulos, 2004] introduces a mechanism which assigns a fraction of a global queue to each participating site and which encourages the sites to report their true processing power, therefore enabling the scheduler to take globally-optimal decisions. The major disadvantage is that a common "currency" for both the cost of execution and the payoff of players is required.

1.5 Summary of Contributions

In this work, we propose a new model of the grid that facilitates the analysis of the performance of the whole system with approaches based on game theory and on equitable multicriteria optimization. We study three concrete models based on this general framework. For the resulting problems, we compute price of anarchy, propose scheduling algorithms and compute worst-case performance ratios. The rest of this section details our contribution.

The Multi-Organizational Grid Model (Chapter 2) describes the grid as an agreement to share resources between independent organizations. The chap-

ter is an extended version of [Rzadca, 2007]. Each organization groups users willing to use the grid but also provides a resource for users belonging to other organizations. By mixing resource providers with consumers, we are able to avoid payments, required by economic approaches to grid resource management. Regarding existing game-theoretic approaches to resource management, in which the infrastructure is considered a common property, we claim that our approach models better the heterogeneity present in academic grids, in which a job is viewed through the organization that has submitted it.

To our best knowledge, this work presents also the first application of the axiomatic theory of equity (Section 1.3.2) to the problem of scheduling. The framework of this approach is defined in Section 2.4 and then applied in specific models in Sections 3.3 and 4.3. By considering axiomatic fairness, we are able to produce a number of solutions that balance the equity of individual participants with the performance of the whole system.

In Chapter 3 we extend to multiple dedicated processors the problem of scheduling sets of sequential jobs on a single processor [Agnetis et al., 2004]. The chapter is an extended version of [Rzadca et al., 2007]. In Proposition 3.2 we prove that, even when a particular class of schedules is considered, the resulting search space is exponential. In Section 3.3 we propose two scheduling algorithms for this problem: exact algorithm based on Dynamic Programming and a heuristics based on taboo search. In Section 3.5 we show that, if each organization controls the local schedule, the Price of Anarchy is linear on the number of jobs. We also show that the resulting game is similar to the Prisoner's Dilemma.

In Chapter 4 we extend the Divisible Load Scheduling theory [Robertazzi, 2003] to on-line systems with multiple loads scheduled on resources having different owners. The chapter is an extended version of [Rzadca and Trystram, 2007]. In Section 4.3 we show that averaging load balancing produces equitable results. Yet, in Section 4.4 we prove that load balancing will never be performed in systems when owners have complete control over their resources. In Section 4.5 we show that, if resources are

similarly loaded, it is sufficient to force organizations to commit to their decisions during "longer" time periods in order to make load balancing the dominating strategy. In Section 4.6, we prove that, with resources controlled by independent entities, it is not possible to construct a load balancing algorithm based on currently observed queue lengths. In Section 4.7 we propose a centralized load balancing algorithm that delivers equitable results even when loads of resources differ significantly.

In Chapter 5 we extend the problem of scheduling rigid, parallel jobs to sets of jobs on multiple resources. This chapter is an extended version of [Pascual et al., 2007]. In Section 5.3 we prove that the worst-case performance ratio of List Scheduling algorithm degrades on multiple resources. We show the lower bound of 2.25 and proof for approximation ratio of 3. In Section 5.4 we compute the Price of Anarchy for the resulting game. In Section 5.5 we propose a centralized scheduling algorithm that has a fixed worst-case performance on the system-wide makespan and at the same time does not worsen the local makespans of individual organizations.

Chapter 2

Multi-Organizational Grid Model

The central notion of our model is that of an *organization*: an entity that groups a resource donated to the grid and local users willing to employ the whole system. Thus, a computational grid logically interconnects several resources such as clusters, supercomputers, but also pieces of specialized equipment like sophisticated displays, microscopes, or DNA sequencers (Figure 2.1). The users of the grid are grouped in organizations, such as laboratories or faculties. Each organization owns one[1] of the resources, which is the organization's contribution to the grid. By contributing, an organization expects that its users will be treated fairly when accessing other resources. We assume that organizations are *independent* from each other. Consequently, each organization is concerned only with the performance of the jobs produced by its members. Moreover, resources may have their local schedulers, which order jobs to be computed according to some criteria. A centralized grid scheduler coordinates these local schedulers. However, as resources are local to organizations, a local scheduler is not obliged to follow grid scheduler's advice. For instance, an organization may locally alter the

[1]Even if an organization has many physical resources, it has complete centralized control over all of them. Thus, in the context of the general model, such resources can be modeled as one 'virtual' resource with many processors.

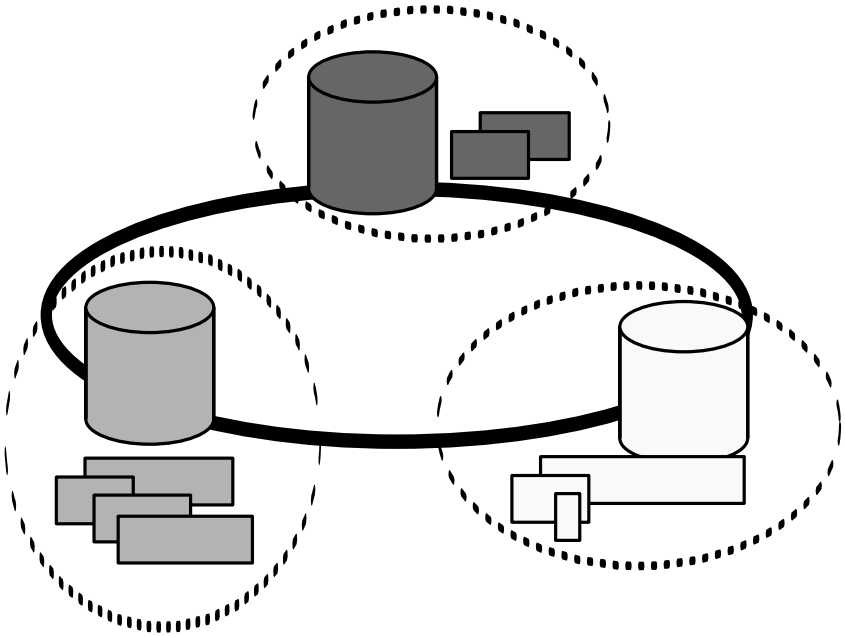


Figure 2.1: A Grid (represented by a thick, black ellipse) interconnects resources (depicted by cylinders) belonging to different organizations, thus crossing organizations' boundaries (represented by dotted lines). Jobs (depicted by rectangles) are owned by organizations.

proposed solution or even quit the grid completely, if it finds its performance unacceptable.

We also assume that there are no external means of compensations for accessing resources. An organization cannot explicitly "pay" other organization neither in some kind of money, nor in barter trade.

2.1 Definitions and Notation

In this section we formally define basic elements of the model and provide corresponding notation. Then, we define possible characteristics of the model that change the available information and the type of jobs to be scheduled.

2.1.1 Elements of the Model

The model of the grid is composed of three basic entities: resources, organizations and jobs.

Resource (denoted as M_k) is an abstraction of an entity that process jobs (e.g. a supercomputer, a cluster, but also a telescope or a sophisticated display). A resource is composed of one, or more *processors* that do the actual work.

Organization (denoted as O_k) is an administrative entity (such as a faculty, a laboratory or a company) that groups users and provides access to a resource. Organizations are independent. $\mathcal{O} = \{O_1, \ldots, O_N\}$ denotes the set of organizations in the model.

Job (denoted as J_k^i) is an entity that, in order to be completed, requires an access to a resource during certain time p_k^i . J_k^i is ith job produced (and owned) by organization O_k. Index i is used only to distinguish between jobs on a processor and does not imply the arrival or execution order. $\mathcal{J}_k$ stands for the set of all jobs produced by O_k. $n_k = |\mathcal{J}_k|$ is the number of such jobs. In a *dedicated* grid model, each job must be executed on a certain resource M_l. $J_{k,l}^i$ denotes such a job.

For resource M_k, jobs produced by the resource's organization O_k are called *local jobs*. All other jobs assigned for execution on this resource are called *foreign jobs*. For organization O_k, *remote jobs* are the jobs produced by this organization which are to be executed on non-local processors.

2.1.2 Additional Characteristics of the Model

Informally, the goal of the scheduler is to find the allocation (*where*) and the time of execution (*when*) for each job. However, the scheduler's decision depends greatly on the information available (e.g. whether all the jobs known in advance or not) and the type of jobs (e.g. whether a job can be stopped and

resumed later or not). Such properties translate into additional assumptions in the modeling process. A number of standard properties is commonly used in the literature [Błażewicz, 1996, Brucker, 2004]. Here, we define the ones used in this work.

Parallel / Sequential Job J_k^i that must be computed in parallel on more than one processor (on a fixed number of q_k^i processors) is called a (rigid) *parallel* job. A *sequential* job requires only one processor.

Preemption If *preemption* is allowed, a job can be interrupted during execution. Otherwise, each job must be executed completely and cannot be interrupted after a processor has started to execute it.

Divisible Load This model allows to divide jobs into a large, but finite, number of fragments that are independent and can be processed in any order.

Release dates Job J_k^i that has a non-zero release date r_k^i cannot be started before time r_k^i (it is not *ready*).

Off-line / on-line A model is *off-line* if release dates r_k^i of all the jobs are known before the scheduling starts. Otherwise, it is called *on-line* [Sgal, 1998].

Clairvoyance In a *clairvoyant* model, the exact size p_k^i of every job submitted to the system is known. For a *non-clairvoyant* model, there are several possibilities. Job's size p_k^i may be completely unknown until the job finishes, it may be estimated with a certain error, or the owner may provide the maximum running time.

Time / Space Sharing If a resource has many processors, they can be shared by the jobs in many ways. In *time sharing*, at any moment, a resource executes only one job (note that time sharing does not require preemption). In *space sharing*, a resource is split between jobs that run on the resource's processors in parallel.

2.1.3 Additional Notation

We use the standard game-theoretic notation of $-k$ (dash) to denote the set containing everything but the element k, so $\mathcal{J}_{-k}$ denotes a set of jobs produced by all but kth organization, $\mathcal{J}_{-k} = \left(\bigcup_l \mathcal{J}_l\right) - \mathcal{J}_k$. Similarly, $\cdot$ (dot) denotes the set containing all possibilities, e.g. in dedicated grid model, $\mathcal{J}_{\cdot,l}$ is the set of jobs produced by all organizations that must be executed on resource M_l, $\mathcal{J}_{\cdot,l} = \bigcup_k \mathcal{J}_{k,l}$.

2.2 Performance Measures

In order to assess the performance of the system, we measure the completion time of jobs [Błażewicz, 1996, Brucker, 2004]. However, there are various levels of aggregation, on which the performance can be measured: the level of individual jobs, the level of organizations or the level of the complete system. There are also various ways in which such an aggregation can be made.

Note that by focusing on the classic notion of completion time, we implicitly assume that the goal of the system is the high-performance computing, i.e. to finish the computation of the given set of jobs as fast as possible. Recently, a number of other goals have been introduced in literature, such as maintaining reliability when resources are falling [Dongarra et al., 2007] or considering power consumption [Graybill and Melhem, 2002] [Kim et al., 2007, Albers et al., 2007]. However, by using the completion time, we are able to study the impact of the notion of organization, i.e. the novelty of the grid, *ceteris paribus*, under the same conditions as the classic parallel systems. Therefore, we are able to compare directly our results.

According to usual notation, C_k^i denotes the completion (finish) time of an individual job J_k^i. To measure the performance experienced by organization O_k, we may compute two aggregated measures. The *sum of completion times* is the sum of completion times of jobs $\mathcal{J}_k$ owned by the organization: $C_k =$

$\sum_i C_k^i$. The *makespan* (maximum completion time) is the time when the last organization's job finishes $C_{\max}(O_k) = \max_i C_k^i$.

The sum of completion times C_k is a measure more equitable than makespan $C_{\max}(O_k)$, assuming that the jobs are owned by different users. Both measures satisfy the condition of symmetry, but the makespan is not monotonic, as it only reflects the improvements of the last finished job. However, both measures do not fulfill the principle of transfers. The level of equity of the measure being optimized by the scheduler has an direct impact on produced schedules. In the classic multiprocessor scheduling problem, the optimization of the sum of completion times results in schedules better for users, whereas the optimization of makespan $C_{\max}$ produces schedules with higher utilization of the resource [Dutot et al., 2005].

We define similar measures on the system level. The *global sum of completion times* ΣC is defined as the sum of completion times of all the jobs in the system $\Sigma C = \sum_k \sum_i C_k^i$. The *global makespan* $C_{\max}$ is the time when last job in the system finishes $C_{\max}(O_k) = \max_{i,k} C_k^i$.

In a system with non-zero release dates, the completion time of the job is not appropriate to evaluate the actual performance. A job released early and completed late has similar completion time as a job released late and completed immediately. Yet, the performance experienced by the first job is low comparing to the latter. Consequently, it is usual to measure instead the flow time F_k^i, the time job J_k^i spends in the system. Flow time is defined as the difference between the completion time and the release date, $F_k^i = C_k^i - r_k^i$. The aggregated measures are defined similarly. The *sum of flow times* of organization O_k's jobs $\mathcal{J}_k$ is defined as $F_k = \sum_i F_k^i$.

2.3 Assumptions and Scope of the Model

From the perspective of resource management algorithms, a *grid* is an agreement between selfish, independent organizations to share their resources. In

this section, we present the assumptions of our model. Note that some of these assumptions are implicitly present in the chosen notation.

We assume that there are no external means of recompense for accessing resources. An organization cannot explicitly "pay" other organization neither in some kind of money, nor in barter trade.

Each user is local to some organization. There are no external, nor "grid-level" users. Consequently, we may assume that each job that has been produced by a user is owned by the user's organization.

For the most part of this work, we assume that all the available resources are owned by organizations. If an organization is outside the grid, it has complete control over the local resource. There are no external, nor "community-owned" resources. For comparison, we also study systems in which there are no local resources. In such systems, all the resources are owned directly by the administrative entity that organizes the grid. The organizations compete for the access to the resources. The goal of the grid organizer is to share the resources in a fair manner.

We assume that the organizations are administratively *independent*. There is no central, administrative entity that forces them to cooperate. A "grid" is rather a possibility than an obligation. If an organization is not satisfied with the performance of the system perceived by its local users, it can eventually reject the agreement and quit the grid.

The organizations are *selfish* in the game-theoretic meaning. Each organization is concerned only with the performance of the locally-produced jobs. Consequently, organizations are not jealous of results achieved by others, if they do not have a direct, negative impact on the locally perceived performance.

Organizations behave *rationally*. They take all the possible actions to optimize the performance of the local jobs. Thus, they not try to "punish" others, if it does not lead to better local performance. Similarly, an organization will not be altruistic towards other organizations. The only motivation the organizations have to take part in the grid is to optimize the performance

of the locally produced jobs.

We assume that resources are connected by high-performance network links offering negligible latency and high bandwidth. We also assume that the grid offers some middleware that allows to send a job to a resource and then to execute remotely the job. Thus, the actual connections, both on the hardware and on the software layer, are out of the scope of this work.

Finally, we assume that the system (resources, communication links and software) is perfectly reliable. Resources do not break down, are always available and do not require maintenance.

2.4 Approaches for Optimization

In this work we consider the minimization of the completion time of jobs in the grid defined in the previous sections. We introduce a centralized, grid-level scheduler which proposes a schedule to each resource. However, the power of the centralized scheduler and, consequently, the kind of solutions it can impose on individual processors, depends heavily on the level of control the individual organizations have on their resources.

In this thesis, we study the problems from three perspectives, leading to three different approaches for optimization: multi-criteria optimization, game theory and constrained multi-criteria optimization.

Firstly, in the most restricted case, we can assume that the organization is not able to impose any schedule on its local resource, neither it is able to quit the grid. This assumption reflects the models in which all the resources are owned and controlled directly by the grid. The problem transforms then into *multi-criteria optimization* of performance measures of respective organizations. We assume that all the organizations use the same type of criteria. Consequently, the grid scheduler optimizes either the vector of completion times $[C_1, \ldots, C_N]$, makespans $[C_{\max}(O_1), \ldots, C_{\max}(O_N)]$ or flow times $[F_1, \ldots, F_N]$.

Secondly, each organization may have complete control over the schedule of the local resource. In such a situation, the grid scheduler acts only as an advisor. The proposed solution must be profitable for each organization. However, each organization is tempted to modify it locally, if the organization's gain is increased. Consequently, such a problem must be analyzed with a *game-theoretic* approach. The framework of such a game may be defined as follows:

- the set of players is equal to the set of organizations $\mathcal{O}$;
- strategy σ_k of player O_k is the scheduling of jobs on player's local resource M_k;
- payoff function u_k for player O_k resulting from a profile of strategies $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_N]$ is the performance of player's local jobs $\mathcal{J}_k$. To measure the performance, a player may use any measure defined in Section 2.2.

The third situation is a combination of the first two. We assume that each organization independently decides whether to join or to leave the grid. However, once inside, the organization grants complete control over its resources to the grid scheduler. Yet, an organization will leave the system, if the gain experienced is lower than the performance the organization could achieve being outside. Therefore, in order to sustain the grid, the resource management system must achieve an acceptable performance not only at the level of the community of users (as in classic, monocriterion scheduling), but also on the between-organizations level. This problem may be defined as *constrained multi-criteria* optimization of the vector of performance measures of respective organizations. For each organization, its performance measure of the resulting schedule must be higher than the performance if the organization remained outside the grid and maintained complete control over the local schedule.

The three approaches defined above are far from being exhaustive to the problem of grid scheduling. However, we claim that they present the problem in three distinct perspectives that are (or will be) mostly widespread

in future grid systems. The multi-criteria optimization approach is suitable for classic systems, in which a number of resources must be shared between organizations in a fair manner. A classic application scenario is a supercomputer bought by a state agency, that is later shared between a number of public laboratories and universities. The game theoretic perspective concerns systems with almost no central control, in which independent parties try to maximize their own gain, with no motivation to optimize the performance of the system perceived as a whole. We expect that some highly distributed, peer-to-peer systems will behave in that manner. Finally, the third perspective, constrained multi-criteria optimization, concerns systems in which individual goals are noticed, but not necessary selfishly maximized. Some level of trust and social control can be maintained in the form of e.g. personal respect of administrators of individual resources. This perspective models grids in which there are a few participating organizations and the participation is somehow limited (like in e.g. academic grids).

The perspective presented in this chapter gives a spectrum of views on grid systems and results in different optimization problems and requires different approaches to solve them. Such a spectrum allows us to compare the performance and thus to measure the cost of the decreased control. This spectrum ranges from systems that are almost like classic supercomputers (optimization), through partly distributed ones (constrained optimization), to systems that are highly distributed (game theory).

2.5 The Model and Real-World Grids

The presented framework emphasizes the significance of individual organizations that form the grid. We argue that it closely models a number of academic grids, such as Grid'5000 [Bolze et al., 2006], which are in fact a cooperative of participating laboratories. In this section, we briefly compare our model with notions present in other types of grids.

Our notion of an organization differs slightly from Virtual Organization (VO) [Foster and Kesselman, 2004]. VO is an entity that groups real-world organizations to collaborate temporary on a specific project. Thus, members of a particular VO have the same goal, that is to advance the common project. However, a VO is not required to provide its own computational resources. In a provider-consumer scheme, VO can either pay resource providers and rent the resources, or convince them in another way. Thus, the organization defined in this thesis can be perceived as a VO that also contributes resources to a common pool.

Recent attempts to commercialize grid computing focus on so-called Service Oriented Architecture (SOA), in which a number of providers sell access to services to consumers. A "service" is defined very widely as it may encapsulate entities as diverse as computational resources, network links but also software licenses or even access to particular data. An architecture in which providers are independent of consumers requires some form of compensation for providers who need to cover their expenses (e.g. electricity, costs of hardware, software or staff). The compensation can be realized in a form either external to the domain of the problem (such as money [Buyya et al., 2005, Kenyon and Cheliotis, 2004]), or a kind of barter trade of CPU power [Andrade et al., 2003]

Note however, that formal payments are not always required in provider-consumer grids. EGEE grid [EGEE, 2007] uses provider-consumer scheme, yet providers are either payed indirectly (through e.g. partial funding of the hardware), or grant their resources for free. All of the participating institutions are, however, research laboratories or educational institutes, in which economic profit is less important than the participation in a cutting-edge research project. Similarly, thousands of providers of computers in BOINC [Anderson, 2004] grant access to their computers in exchange for the sense of doing something "for humanity" (or, more down-to-earth, to appear in one of the various "best providers" rankings).

We assume that there are no "payments" for accessing resources. This

assumption allows us to study the problem of resource management directly, thus without requiring the strong assumptions needed by e.g. the free market theory. In addition, the resulting resource management systems do not require book-keeping, complex accounting nor a centralized, trusted bank that issue money. Finally, we consider that a grid system should promote cooperation between organizations and individual users. However, the notion of money reduces the willingness of people to cooperate (Section 1.4.3, [Vohs et al., 2006]).

2.6 Summary

In this section we presented a new framework model of a grid (Section 2.1). We assume that the grid is a result of an agreement to share resources between independent organizations. An organization groups local users but also provides a resource for users coming from different organizations.

We assume that there are no "payments" for accessing resources. This assumption allows us to study directly, i.e. without studying the market, the problem of resource management, to avoid the problems of book-keeping and banking, and finally, to avoid the negative impact the concept of money has on cooperation.

In Section 2.4, we presented three perspectives from which we will study the concrete models in next chapters: fair multicriteria optimization, game theory, and constrained fair multicriteria optimization. In the first approach we assume that the grid resource management system has a complete control over resources. The goal is to share them equitably among the organizations. In the second approach, we analyze the system from the game-theoretic perspective. Here, the grid scheduler can suggest a schedule to individual resources. Still, the owner of a resource can alter the schedule, if he/she finds the resulting modification profitable. Finally, in constrained optimization, organizations cannot modify the proposed schedule. However, they may reject it (and thus quit the grid), if the organization's performance is reduced,

compared to a situation with no grid. These three diverse approaches allow us to study the impact of different levels of decentralization on the performance of the system.

Chapter 3

Resource Sharing

In this chapter, we consider the grid as a tool for accessing specialized resources (see Figure 3.1). We assume that resources are dedicated. Each job in the system must be computed on a specific resource, not necessarily the one belonging to the owner of the job. The main problem is when to execute such foreign jobs. The solution proposed by classic approaches is to execute jobs in order of their increasing computation times on each processor [Błażewicz, 1996]. However, this solution may be inefficient for organizations owning highly demanded equipment. Another possible solution is to procrastinate the execution of "foreign" jobs on each processor, though it is usually globally inefficient.

We study three models which differ by the level of control the grid scheduler has over the resources' schedules. The three models correspond to three approaches for optimization defined in Section 2.4.

Firstly, we analyze the problem of fair scheduling in systems in which resources are owned by the community and the grid scheduler has complete control over their schedules. As it was stated in Section 2.4, we formulate the problem as the *multi-criteria optimization*. In Section 3.2, we study the complexity of the problem, and in Section 3.3 we propose three algorithms for multicriteria optimization: an exhaustive search, an exact dynamic programming, and a greedy heuristics.

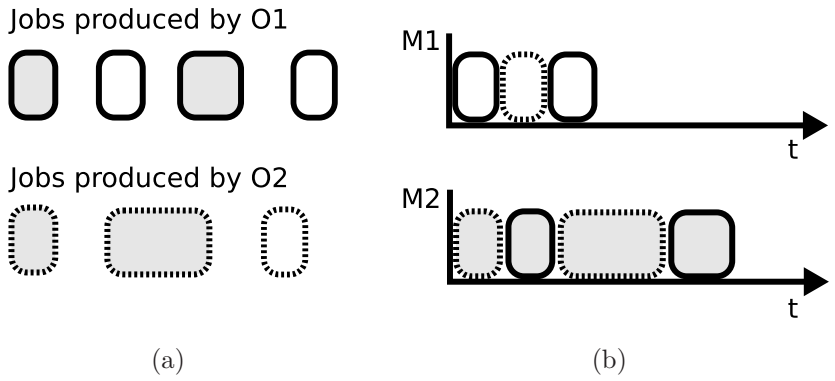


Figure 3.1: A dedicated grid composed of two organizations and two dedicated resources. Each organization produces jobs for both resources (a). O_1's jobs are plotted in continuous lines, O_2's jobs – in dotted. Jobs are dedicated, therefore there are scheduled on a specific processor (b). M_1 processes white jobs, M_2 processes gray jobs.

Secondly, in Sections 3.4 and 3.5 we study the problem of scheduling in a decentralized case, in which the grid scheduler only suggests a schedule, which can be later modified by a resource's owner. This problem will be analyzed with *game-theoretic* approach. We show that the game corresponding to decentralized scheduling is analogous to the well known Prisoner's Dilemma game. The Nash equilibrium results in significant performance drop. Therefore, a strong community control is required to achieve acceptable performance.

Thirdly, in Section 3.6, we analyze the problem of feasible scheduling in semi-decentralized case. Here, organizations cannot locally modify the schedule, but they may leave the grid, if the experienced performance is unsatisfactory. This problem corresponds to the *constrained multi-criteria* approach as it was stated in Section 2.4. To find a feasible solution, we use algorithms similar to the ones used in *multi-criteria optimization*.

3.1 Grid Model

In this chapter, we make the following assumptions (in addition to the ones assumed in Section 2.3) about the system (see Section 2.1.2 for formal definitions):

- the model is *off-line*;
- there are no release dates;
- the scheduler is *clairvoyant*;
- *preemption* is not allowed;
- jobs are *sequential*;
- each resource has only one processor;
- resources are *time-shared* between jobs;
- in order to measure the performance, each organization computes the *sum of completion times* of locally produced jobs.

These assumptions lead to the following definitions. For resource M_l, a *(list) schedule* is a permutation $\pi_l : \mathcal{J}_{\cdot,l} \rightarrow (J_{\cdot,l}^{\pi_l(1)}, J_{\cdot,l}^{\pi_l(2)}, \ldots, J_{\cdot,l}^{\pi_l(n_{\cdot,l})})$ of jobs $\mathcal{J}_{\cdot,l}$ that must be executed on that resource. A non-preempting, time-sharing resource executing schedule π_l, firstly executes the first job $J_{\cdot,l}^{\pi_l(1)}$. Then, when $J_{\cdot,l}^{\pi_l(i-1)}$ is finished, $J_{\cdot,l}^{\pi_l(i)}$ is started. For simplicity of notation, in the description of algorithms we treat π_l as a vector containing $\mathcal{J}_{\cdot,l}$ in the order of execution.

A *Shortest Processing Time (SPT)* schedule π_l^{SPT} is a schedule which orders the jobs of each organization $\mathcal{J}_{k,l}$ according to non-decreasing sizes of jobs. If there is one resource and one organization, SPT schedule is optimal with respect to the sum of completion times of jobs [Błażewicz, 1996]. By **SPT**, we denote a SPT schedule that orders jobs on all resources according to non-decreasing execution times, regardless of jobs' owners.

A *My Jobs First (MJF)* schedule is a SPT schedule that orders all the local jobs $\mathcal{J}_{\cdot,l}$ before the rest of the jobs $\mathcal{J}_{-l,l}$.

A *grid schedule* is a tuple $(\pi_1, \ldots, \pi_N)$ that defines a schedule π_l for each machine M_l.

Without loss of generality, let us assume that on each resource and for each organization the jobs are numbered according to non-decreasing execution times (i.e. $p^i_{k,l} \leq p^{i+1}_{k,l}$).

By $C_{k,l}$ we denote the sum of completion times of jobs $\mathcal{J}_{k,l}$ produced by O_k and executed on M_l: $C_{k,l} = \sum_i C^i_{k,l}$.

Each organization O_k is concerned with the sum of completion times C_k of the jobs $\mathcal{J}_{k,\cdot}$ which have been produced by its members. Organization O_k does not care about the performance of other organizations. However, as the processors are dedicated, organizations must submit jobs also to non-local processors. As a result, C_k depends heavily on the performance of jobs executed on processors other than the organization's local processor.

3.2 Optimization Approach

The scheduling problem considered in this section is the *fair* ordering of jobs $\mathcal{J}_{\cdot,l}$ on each resource M_l which minimizes the sums of completion times $[C_k]$ of all the organizations. This section considers the complexity of the problem. The optimization algorithms are presented in the next section, as we use related algorithms in section concerning constrained optimization approach.

We analyze two related problems. Firstly, in Section 3.2.1, we count how many possible Pareto-optimal solutions exist for a given instance. An optimal algorithm should return all the optimal solutions, amongst which a choice can be made by an entity managing the grid. However, we show that the number of Pareto-optimal schedules can be exponential on the number of jobs. Consequently, there is no efficient algorithm enumerating all the optimal schedules.

As producing all the solutions is not feasible, in Section 3.2.2 we study the problem of finding just one schedule with given performance for each organization. We consider a restricted case with one resource and two organizations. We prove that even in such a restricted case, the decision version of the problem is NP-Complete. Thus, the complete grid scheduling problem defined in this section is also NP-hard.

3.2.1 All Optimal Solutions

In equitable multicriteria optimization, the set of equitable solutions is the subset of the set of all Pareto-optimal solutions. Therefore, the number of equitable solutions is not higher than the size of the latter set. We firstly prove that only SPT schedules can be Pareto-optimal, a result that reduces the solution space considerably. However, we show that, in the worst case, there is still an exponential number of Pareto-optimal solutions. Nevertheless, determining the worst-case number of equitable solutions remains an open problem.

Firstly, we prove that all Pareto-optimal schedules are SPT schedules. Then, we consider a restricted case with one resource and two organizations. We show that for this restricted case, the number of SPT schedules is exponential. Moreover, it is possible to construct an instance for which the number of Pareto-optimal SPT schedules is exponential. Finally, we formally prove that the order of Pareto-optimal solutions is not reduced when considering more than one resource.

We start by proving the following proposition that reduces the number of reasonable schedules for each resource.

Proposition 3.1. *For resource M_l, a schedule which orders jobs $\mathcal{J}_{k,l}$ originating from a organization O_k not in shortest processing time (SPT) order is Pareto-dominated by a SPT schedule for these jobs.*

Proof: The proof of this proposition is based on an exchange argument and it is analogous to the well known proof of the optimality of the Shortest Processing Time algorithm for a single resource and a single organization.

Let us assume that there are two jobs $J^i_{k,l}$ and $J^j_{k,l}$ executed in non-SPT order. Job i is longer ($p^i_{k,l} > p^j_{k,l}$), but it is executed before job j ($C^i_{k,l} < C^j_{k,l}$). If these two jobs are swapped (resulting in completion times $\widetilde{C}^j_{k,l} < \widetilde{C}^i_{k,l}$), the sum of completion times of these two jobs will be decreased. $J^j_{k,l}$ completes earlier (i.e. $\widetilde{C}^j_{k,l} < C^i_{k,l}$, since the first job starts at the same moment, but it is shorter). $J^i_{k,l}$ completes at the same moment as $J^j_{k,l}$ did before the swap ($\widetilde{C}^i_{k,l} = C^j_{k,l}$). All the jobs scheduled between these two jobs will complete earlier, as they will start earlier. The rest of the jobs is not affected by the swap. Consequently, this swap reduces at least the sum of completion times C_k for the owner O_k of the jobs being swapped. □

Note that this proposition does not hold for jobs belonging to different organizations. Given two jobs with different owners $J^i_{k,l}$ and $J^j_{m,l}$, $J^i_{k,l}$ executed before $J^j_{m,l}$, swapping them would increase C_k, at the same time decreasing C_m. Consequently, neither solution would Pareto-dominate the other one.

The consequence of the proposition presented above is that the number of reasonable (Pareto-efficient) schedules is reduced considerably. However, their number is still very large even when there are only two organizations O_1 and O_2.

Proposition 3.2. *The number of SPT schedules on one resource is exponential on the number of jobs on that resource.*

Proof: Consider resource M_1 with $n_{1,1}$ local jobs and $n_{2,1}$ foreign jobs. A multiset (called also a "bag") is a set which may contain multiple copies of an element. The schedule can be fully described by a multiset of size $n_{2,1}$, with elements $\{1, \ldots, n_{1,1}+1\}$. The idea is that each member of the multiset specifies a placement for one foreign job. In order to construct a schedule from the multiset, the local jobs are ordered according to SPT. Then, the shortest foreign job is assigned the smallest element from the multiset, which

specifies its placement (1 – the job is placed before the shortest local job, 2 – before the second shortest local job, ..., $n_{1,1}+1$ – after the longest local job). This element (actually, one of the copies of the element) is removed from the multiset. The algorithm proceeds with the rest of foreign jobs (considering them in SPT order). Consequently, the number of SPT schedules for one resource is equal to the number of multisets of size $n_{2,1}$ with elements $\{1,\ldots,n_{1,1}+1\}$, which in turn is equal to the number of combinations with repetition of $(n_{1,1}+1)$ objects from which $n_{2,1}$ objects are chosen:

$$\binom{n_{1,1}+n_{2,1}}{n_{2,1}} = \binom{n}{n_{2,1}} \leq \binom{n}{\frac{n}{2}},$$

where $n = n_{1,1}+n_{2,1}$ is the total number of jobs. We will compute the order of this expression by estimating the logarithm with Stirling's formula:

$$\begin{aligned}
\log\left(\binom{n}{\frac{n}{2}}\right) &= \log\left(\frac{n!}{(\frac{n}{2})!\,(\frac{n}{2})!}\right) = \log(n!) - 2\log\left(\left(\frac{n}{2}\right)!\right) \\
&= (n+\frac{1}{2})\ln 2 - n + \ln\sqrt{2\pi} - 2\left(\frac{n+1}{2}\ln 2 - \frac{n}{2} + \ln\sqrt{2\pi}\right) \\
&= n\ln 2 - \frac{1}{2}\ln m + \ln 2 - \ln\sqrt{2\pi} \\
&= \ln\left(\frac{2^n}{\sqrt{n}}\frac{2}{\sqrt{2\pi}}\right).
\end{aligned} \tag{3.1}$$

Consequently,

$$\log\left(\binom{n}{\frac{n}{2}}\right) \leq \ln\left(2^n\frac{2}{\sqrt{2\pi}}\right).$$

Neglecting the constants, we get:

$$\binom{n}{\frac{n}{2}} \in O(2^n).$$

□

Note, that one SPT schedule may Pareto-dominate other SPT schedules. Let us consider an instance in which O_1 has three jobs of sizes $p^1_{1,1} = 1$,

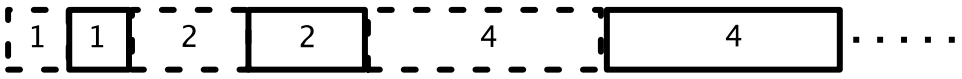


Figure 3.2: An instance for which the number of Pareto-optimal SPT schedules on one processor is exponential. There are two organizations O_1 and O_2. Their jobs are denoted by continuous (O_1) and dashed (O_2) lines. Each organization has n jobs of lengths $1, 2, 4, \ldots, 2^{n-1}$.

$p^2_{1,1} = 3$, $p^3_{1,1} = 5$ and O_2 has two jobs of sizes $p^1_{2,1} = 2$, $p^2_{2,1} = 4$. SPT schedule $(J^1_{2,1}, J^1_{1,1}, J^2_{1,1}, J^3_{1,1}, J^2_{2,1})$ results in total completion times $[20, 17]$, which is Pareto-dominated by schedule $(J^1_{1,1}, J^2_{1,1}, J^1_{2,1}, J^2_{2,1}, J^3_{1,1})$, resulting in total completion times $[20, 16]$.

In the worst case, however, the number of non-dominated SPT schedules remains exponential, as it is formally stated by the following proposition.

Proposition 3.3. *The number of Pareto-optimal SPT schedules can be exponential on the number of jobs.*

Proof: We construct an instance for which the proposition holds. Let us consider the following instance (proposed by [Agnetis et al., 2004]). Two organizations O_1 and O_2 produced identical workload of $n = n_{1,1} = n_{2,1}$ jobs with increasing lengths $p^1_{k,1} = 1$, $p^2_{k,1} = 2$, $\ldots$, $p^n_{k,1} = 2^{n-1}$ (see Figure 3.2). Let us further consider a subset of all possible SPT schedules that execute jobs in SPT order regardless of the owner of the job (i.e. $\forall_{k,k'} J^i_{k,1}$ is executed before $J^{i+1}_{k',1}$).

The lower bound (LB) on C_k caused by the SPT order between jobs is at least:

$$\begin{aligned} C_k \geq LB &= 1 + \\ &+ 2 \cdot 1 \cdot (n-1) + \\ &+ 2 + \\ &+ 2 \cdot 2 \cdot (n-2) + \\ &+ \ldots + \\ &+ 2^{n-2} + \\ &+ 2 \cdot 2^{n-2} \cdot 1 + \\ &+ 2^{n-1}. \end{aligned}$$

In the above formula, the first line corresponds to $J^1_{k,1}$ length, the second to the fact that $J^1_{1,1}$ and $J^1_{2,1}$ precede other jobs, the third is the $J^2_{k,1}$ length, the forth is the length of $J^2_{1,1}$ and $J^2_{2,1}$ that precede longer jobs and so on.

Let us now consider C_1. In each pair of jobs of the same length $(J^i_{1,1}, J^i_{2,1})$, executing $J^i_{2,1}$ before $J^i_{1,1}$ delays $J^i_{1,1}$ by $p^i_{2,1} = 2^{i-1}$ and therefore increases C_1 by 2^{i-1}. We introduce a control variable $o_i = 1$ if $J^i_{2,1}$ is executed before $J^i_{1,1}$, $o_i = 0$ otherwise. $[C_1; C_2]$ can be expressed as:

$$[C_1; C_2] = [LB + \sum_i o_i \cdot 2^{i-1}; LB + \sum_i (1 - o_i) \cdot 2^{i-1}].$$

This encodes all pairs of integer numbers in the form: $[LB+2^n-1, LB], [LB+2^n-2, LB+1], \ldots, [LB+1, LB+2^n-2], [LB, LB+2^n-1]$. All such pairs are Pareto-optimal. Moreover, there are 2^n such pairs, as each of n control variables o_i can be either 0, or 1. □

A complete schedule on the grid level specifies schedules for each processor. In a grid composed of two resources, each Pareto-optimal SPT schedule of the first resource is matched with each SPT schedule of the second resource. In the worst case, the number of possible combinations is of the order of $O(2^n \cdot 2^n)$. Most of the resulting combinations are Pareto-dominated. Nevertheless, the following proposition can be proved.

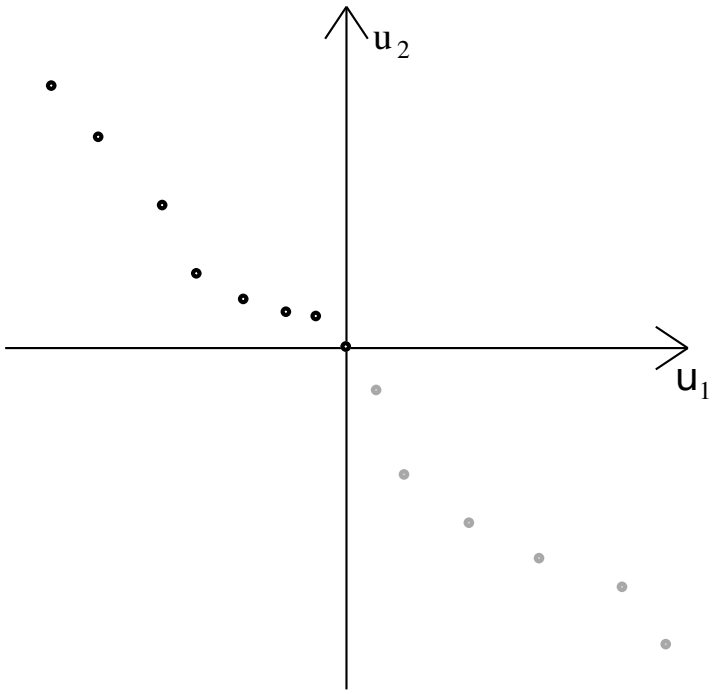


Figure 3.3: Total completion times resulting from base grid schedules adjusted by a constant $u_i = C_i(\mathbf{MJF}) - C_i(\boldsymbol{\pi})$. Gray points present Π_2, or results of advancing foreign jobs $\mathcal{J}_{1,2}$ on M_2 (assuming that M_1 executes all local jobs before any foreign job), black points present Π_1, or the results of advancing $\mathcal{J}_{2,1}$ on M_1 with the same assumption.

Proposition 3.4. *The number of Pareto-optimal grid schedules is at least of the order of the number of Pareto-optimal schedules on one of the resources.*

Proof: Let us denote as Π_1 the set of Pareto-optimal schedules on the first resource, and as Π_2 – on the second. Π_1 and Π_2 can be plotted (Figure 3.3) in two-dimensional space (u_1, u_2), where $u_i = C_i(\mathbf{MJF}) - C_i(\boldsymbol{\pi})$, $C_i(\boldsymbol{\pi})$ is the sum of completion times resulting from a particular SPT schedule $\boldsymbol{\pi} = [\pi_1, \pi_2]$ and $C_i(\mathbf{MJF})$ is a constant enabling us to start both plots at $0, 0$. Note that this plot can also be obtained by plotting (C_1, C_2), reversing the axes and adjusting each C_i solution by adding constant $C_i(\mathbf{MJF})$ (the set of Pareto-optimal solutions is the same).

The proof is by induction on set of schedules of one resource. Let us

assume that we enumerate Π_1 in the order of decreasing u_1. The first solution is $\pi_1^1 = [u_1^1, u_2^1] = [0, 0]$, the second solution is $\pi_1^2 = [u_1^2, u_2^2]$ and so on. In order to construct a set of grid schedules, each $\pi_1 \in \Pi_1$ must be matched with each $\pi_2 \in \Pi_2$. This can be visualized by duplicating Π_2 $|\Pi_1|$ times and translating consecutive duplicates by vectors $[u_1^1, u_2^1]$, $[u_1^2, u_2^2]$,.... Let us denote Π^i as Π_2 translated by $[u_1^i, u_2^i]$. Consider iteratively adding set Π^i to set $\bigcup_{k=1}^{i-1} \Pi^k$. Let us denote π^{*i} as the element of Π^i with the highest value of u_1 that Pareto-dominates an element from $\bigcup_{k=1}^{i-1} \Pi^k$. Firstly, all the elements in Π^i with lower value of u_1 are also Pareto-optimal. Secondly, the value of u_1 of π^{*i} before the translation must be higher than u_1 of the first Pareto-dominated point in $\bigcup_{k=1}^{i-1} \Pi^k$ (as the translation reduces u_1). It follows that if Π^i dominates n Pareto-optimal elements of $\bigcup_{k=1}^{i-1} \Pi^k$, at least $n + 1$ new Pareto-optimal elements are added. Consequently, the resulting number of Pareto-optimal points is at least of the order of the number of elements of Π_2. □

Proposition 3.5. *In a grid composed of two organizations ($N = 2$), the number of Pareto-optimal grid schedules can be in $O(2^{2n})$.*

Proof: We show an instance with $N = 2$ for which no combination of Pareto-optimal SPT schedules is dominated and the proposition holds. Let us consider an instance similar to one in the proof of Proposition 3.3. On M_1, two organizations O_1 and O_2 produced identical workload of $n = n_{1,1} = n_{2,1} = n$ jobs with increasing lengths $p_{k,1}^1 = 1$, $p_{k,1}^2 = 2$, ..., $p_{k,1}^n = 2^{n-1}$. On M_2, the length of jobs is multiplied by $K \geq 2^n$, i.e. $n = n_{1,2} = n_{2,2} = n$, and $p_{k,2}^1 = K$, $p_{k,2}^2 = K2$, ..., $p_{k,2}^n = K2^{n-1}$. Similarly to Proposition 3.3, **SPT** schedules on M_1 result in pairs $[C_1, C_2]$ equal to $[LB+2^n-1, LB]$, $[LB+2^n-2, LB+1]$, ..., $[LB+1, LB+2^n-2]$, $[LB, LB+2^n-1]$, whereas on M_2 we get $[LB'+K(2^n-1), LB']$, $[LB'+K(2^n-2), LB'+K]$, ..., $[LB'+K, LB'+K(2^n-2)]$, $[LB', LB'+K(2^n-1)]$ (where LB' is LB computed as in Proposition 3.3, but with increased job sizes). After matching those schedules in all-to-all manner, we add $[C_1, C_2]$ on M_1 to $[C_1, C_2]$ on M_2. Denoting $LB'' = LB+LB'$,

we get pairs of $[LB'' + K(2^n - 1) + 2^n - 1, LB'']$, $[LB'' + K(2^n - 1) + 2^n - 2, LB'' + 1]$, $[LB'' + K(2^n - 1) + 2^n - 3, LB'' + 2]$, $\ldots$, $[LB'' + K(2^n - 2) + 2^n - 1, LB'' + K]$, $[LB'' + K(2^n - 2) + 2^n - 2, LB'' + K + 1]$, $\ldots$, $[LB'', LB'' + + K(2^n - 1) + 2^n - 1]$. Note that all such pairs are Pareto-optimal. Considering grid schedules in the aforementioned order, C_2 is strictly increasing, whereas C_1 is strictly decreasing. □

Corollary 3.1. *When N is pair, the number of Pareto-optimal grid schedules can be in $O(2^{Nn})$.*

Proof: It is sufficient to consider instances in which, $\forall 0 \leq k \leq \frac{N}{2}$, $\mathcal{J}_{2k,2k}$, $\mathcal{J}_{2k,2k+1}$, $\mathcal{J}_{2k+1,2k}$, $\mathcal{J}_{2k+1,2k+1}$ are constructed as in the proof of Proposition 3.3, and the rest of the sets $\mathcal{J}_{l,l'}$ is empty (i.e. each organization produces jobs only for its own processor and the processor of its predecessor or successor). All combinations of schedules between pairs M_{2k}, M_{2k+1} are Pareto-optimal. □

Consequently, the number of Pareto-optimal grid schedules is exponential in the worst case. This defines the lower bound on the complexity of the optimization algorithms that enumerate all Pareto-optimal solutions.

3.2.2 One Solution

In the last section we showed that finding *all* Pareto-optimal solutions is not feasible as the size of the resulting set can be exponential. However, even when we relax the goal to find just one of the Pareto-optimal solutions on one resource, the decision version of the problem is NP-Complete.

Here, we consider the grid scheduling problem restricted to one resource and two organizations (denoted as RGS). An instance of this problem is given by two sets of jobs $\mathcal{J}_{1,1} = \{J^1_{1,1}, J^2_{1,1}, \ldots, J^{n_{1,1}}_{1,1}\}$ and $\mathcal{J}_{2,1} = \{J^1_{2,1}, J^2_{2,1}, \ldots, J^{n_{2,1}}_{2,1}\}$ with known sizes $p^i_{k,l}$. The decision version of the problem consists in finding a schedule (a permutation) of $\mathcal{J}_{1,1} \cup \mathcal{J}_{2,1}$, such that the total completion time

Algorithm 3.1: Algorithm that checks if a given schedule π fulfills the constraints Q_1, Q_2 on the sum of completion times of organizations.

Input: $\mathcal{J}_{1,1}$, $\mathcal{J}_{2,1}$, π, Q_1, Q_2
Output: True if $C_1 \leq Q_1$ and $C_2 \leq Q_2$, False otherwise
$C_1 = 0; C_2 = 0$;
$t = 0$;
for $i = 1, \ldots, |\mathcal{J}_{1,1} \cup \mathcal{J}_{2,1}|$ **do**
    $t = t + p_{.,1}^{\pi(i)}$;
    if $J_{.,1}^{\pi(i)} \in \mathcal{J}_{1,1}$ **then**
        $C_1 = C_1 + t$;
    else
        $C_2 = C_2 + t$;
if $C_1 \leq Q_1$ **and** $C_2 \leq Q_2$ **then return** *True* **else return** *False*

C_1 of O_1 is not worse than a given constraint Q_1; at the same time C_2 is not worse than Q_2. Following the usual notation of scheduling problems, the restricted problem can be denoted as $1\|\Sigma C_{1,1}^i : \Sigma C_{2,1}^i$.

Claim 3.1. *The decision version of the grid scheduling problem restricted to one resource and two organizations (RGS) belongs to NP.*

Proof: Given a schedule π, Algorithm 3.1 can be used to check if the solution fulfills the constraints. □

We will prove the NP-Completeness of $1\|\Sigma C_{1,1}^i : \Sigma C_{2,1}^i$ by a reduction from Partition, following the proof given by [Agnetis et al., 2004].

An instance of Partition is given by a set $A = \{s_1, s_2, \ldots, s_k\}$ of k positive integers (without loss of generality numbered such that $s_{i-1} \leq s_i$). We denote as S the sum of all elements of A, $S = \sum_i s_i$. The decision version of Partition consists of answering the following question: is there a subset $A' \subseteq A$ such that $\sum_{i:s_i \in A'} s_i = \sum_{i:s_i \in A-A'} s_i = \frac{S}{2}$?.

Let us now construct an instance of RGS from an instance of Partition by the following reduction:

- Organizations have identical sets of $n = n_{k,1} = |A|$ jobs of sizes equal to the elements of the set A: $p^i_{1,1} = p^i_{2,1} = s_i$
- $Q_1 = Q_2 = \frac{3}{2}S + 2(\sum_i^n (n-i)p_i)$.

Claim 3.2. *The reduction is polynomial.*

Proof: Each $s_i \in A$ is converted to exactly two jobs $J^i_{1,1}$ and $J^i_{2,1}$. Q_1 and Q_2 can be computed in $O(n)$ time. □

In the remainder of the proof, we restrict our attention to a subset **SPT** of all possible schedules. In such schedules, jobs are executed in non-decreasing order of jobs' sizes, i.e. $\forall \pi \in \mathbf{SPT} : p^{\pi(i)}_{\cdot,1} \leq p^{\pi(i+1)}_{\cdot,1}$. Note that there are 2^n such schedules in the considered instance. We firstly compute the sum of completion times for such schedules and then show that a feasible schedule for RGS is one of such schedules.

Lemma 3.1. *All the* **SPT** *schedules have the same total completion time* $\sum_i C^i_{\cdot,1} = T = 3S + 4\sum_{i=1}^n (n-i)s_i$.

Proof: Firstly, all the jobs must be executed, so the total completion time is at least $2\sum_{i=1}^n s_i = 2S$. Then, as the schedule is **SPT**, each of the two shortest jobs (of size s_1) delays the following $2n-2$ jobs by s_1. Moreover, as they are executed sequentially, one of the jobs delays the other by s_1. Extending this reasoning to other jobs, we obtain the total completion time increased by $\sum_{i=1}^n 2(2n-2i)s_i + s_i = S + \sum_{i=1}^n 4(n-i)s_i$. By adding the two elements, we get $\sum_i C^i_{\cdot,1} = T$. □

Note that the maximum total completion time goal of both organizations is equal to the half of the total completion time $Q_1 = Q_2 = T/2$. This will finally enable us to prove that a feasible schedule is **SPT**.

Lemma 3.2. *If a feasible schedule* π *exists (such that* $\Sigma C^i_{1,1} \leq Q_1$ *and* $\Sigma C^i_{2,1} \leq Q_2$*), it is* **SPT**.

Proof: The proof is by contradiction. Let us assume that π is feasible, yet it is not **SPT**. If π is feasible, the total completion time $\Sigma C^i_{\cdot,1} \leq Q_1 + Q_2 \leq T$. If it is not **SPT**, there is at least one longer job executed before shorter one. Switching the order of execution of these two jobs reduces the total completion time. Let us make all possible switches until the resulting schedule is **SPT**. All switches were reducing the global sum of completion times, so the resulting total completion time is strictly lower than T. However, Lemma 3.1 shows that all **SPT** schedules have the same total completion time of T. This leads to a contradiction. □

A direct conclusion is that, in a feasible schedule, the two organizations have their total completion times exactly equal to their bounds Q_1 and Q_2.

Corollary 3.2. *In a feasible schedule,* $\Sigma C^i_{1,1} = \Sigma C^i_{2,1} = T/2$.

Proof: From the last lemma we know that if a feasible schedule is **SPT**, thus $\Sigma C^i_{\cdot,1} = T = \Sigma C^i_{1,1} + \Sigma C^i_{2,1}$. Hence, $\Sigma C^i_{2,1} = T - \Sigma C^i_{1,1}$. A feasible schedule satisfies $\Sigma C^i_{1,1} \leq T/2$ and $\Sigma C^i_{2,1} \leq T/2$. In the second inequality, after substituting $\Sigma C^i_{2,1}$, we get $\Sigma C^i_{1,1} \geq T/2$. The only value of $\Sigma C^i_{1,1}$ satisfying two inequalities is $\Sigma C^i_{1,1} = T/2$, thus also $\Sigma C^i_{2,1} = T/2$. □

We now compute the total completion time of an organization. Firstly, in **SPT** schedule π, consider a pair of jobs $J^i_{1,1}$ and $J^i_{2,1}$ of equal size s_i. Let us define a function $x(i)$ such that $x(i) = 0$, if in π $J^i_{1,1}$ is executed before $J^i_{2,1}$, $x(i) = s_i$ otherwise. x denotes the value of the sum, $x = \sum_{i=1}^{n} x(i)$.

Lemma 3.3. *In* **SPT** *schedule* π*, total completion times of organizations are* $\Sigma C^i_{1,1} = x + S + 2\sum_i (n-i)s_i$ *and* $\Sigma C^i_{2,1} = (S - x) + P + 2\sum_i (n-i)s_i$.

Proof: For an organization, each of n jobs must be executed, which results in a completion time equal to $\sum_i s_i = S$. In addition, executing the i-th pair $(J^i_{1,1},\ J^i_{2,1})$ delays the $n - i$ remaining tasks by $2s_i$. For all tasks, this leads to $2\sum_i (n-i)s_i$. Finally, if the pair is executed in order $(J^i_{1,1},\ J^i_{2,1})$, $J^i_{1,1}$ increases $\Sigma C^i_{2,1}$ by $s_i = s_i - x(i)$. Otherwise, $\Sigma C^i_{1,1}$ increases by $x(i)$. For all

pairs, this results in x for the first organization, and $S - x$ for the second. □

Now, we can show that the problem is NP-Complete.

Theorem 3.1. *The grid scheduling problem restricted to one resource and two organizations (RGS) is NP-Complete.*

Proof: We showed that the problem is in NP. We also proved a polynomial reduction from Partition. We now show that, given a solution of RGS, it is possible to solve Partition, and, given a solution of Partition, it is possible to solve RGS.

From Corollary 3.2, we have $\Sigma C^i_{1,1} = \Sigma C^i_{2,1}$. From Lemma 3.3, we have $\Sigma C^i_{1,1} = x + S + 2\sum_i (n-i)s_i$ and $\Sigma C^i_{2,1} = (S-x) + P + 2\sum_i (n-i)s_i$. Thus, $x + S + 2\sum_i (n-i)s_i = (S-x) + P + 2\sum_i (n-i)s_i$, which reduces into $x = S/2$. Thus, x must be equal to the half of the sum of the elements defined by an instance of Partition.

If solution A' to Partition is known, we construct a **SPT** schedule in which $J^i_{2,1}$ precedes $J^i_{1,1}$ iff $s_i \in A'$. For all such pairs, $x(i) = s_i$, which gives us the required $\sum_i x(i) = S/2$.

If a schedule π satisfying $\Sigma C^i_{1,1} \leq Q_1$, $\Sigma C^i_{2,1} \leq Q_2$ is known, we construct set A' by adding element s_i iff $J^i_{2,1}$ precedes $J^i_{1,1}$ in π. As $\sum_i x(i) = S/2$, and $x(i) = s_i \Leftrightarrow J^i_{2,1}\text{precedes}J^i_{1,1}$, $x(i) = 0$ otherwise, the sum of elements in A' will be equal to $S/2$. □

3.3 Algorithms

In this section, we present different approaches to produce the set of equitably-optimal solutions. These algorithms solve the problem of multi-organizational scheduling using the optimization approach (defined in the previous section). However, after some slight modifications, we use them in the constrained optimization problem (Section 3.6).

Procedure sptRecursive(π_l, j_l, $n_{.,l}$, *jobsLeft*, Π_l^{SPT})

Input:

- π_l: currently constructed schedule;
- j_l: number of jobs scheduled in π_l;
- $n_{.,l}$ the total number of jobs on M_l;
- jobsLeft: a dictionary that maps O_k to the list containing unscheduled jobs of $\mathcal{J}_{k,l}$ ordered in SPT order;
- Π_l^{SPT} the set of all SPT schedules constructed so far.

```
if j_l = n_{.,l} then
    Π_l^SPT ∪= {π_l} ;
    return
foreach O_k ∈ jobsLeft.keys() do
    if len(jobsLeft[O_k]) > 0 then
        job = jobsLeft[O_k].pop() ;
        π_l[j_l] =job ;
        sptRecursive(π_l, j_l + 1, jobsLeft,Π_l^SPT) ;
        jobsLeft[O_k].push(job) ;
```

Subsequent parts describe three algorithms: naive exhaustive search that tests all the possible schedules and removes the equitably-dominated ones (Section 3.3.1); an algorithm based on dynamic programming that iteratively expands grid schedules removing the Pareto-dominated ones (Section 3.3.2); and a greedy one, based on a taboo-search [Michalewicz and Fogel, 2004] heuristics, called Equitable Walk (Section 3.3.3).

3.3.1 Scheduling with Exhaustive Search

Exhaustive search (ES) is a naive optimization algorithm that enumerates all the possible solutions and removes the ones that are equitably dominated.

Algorithm 3.3: Exhaustive Search (ES) that produces all equitably-optimal grid schedules.

Input: $\mathcal{J}_{k,l}$, sets of jobs on each processor

Output: Π^{eq}, the set of all equitably-optimal grid schedules

foreach M_l **do**

$\Pi_l^{SPT} = \emptyset$;

`sptRecursive`$(()$, $\emptyset$, $n_{\cdot,l}$, $\{O_k \rightarrow \mathcal{J}_{k,l}\}$, $\Pi_l^{SPT})$;

$\Pi^{eq} = \emptyset$;

$\Pi^{SPT} = \Pi_1^{SPT} \times \ldots \times \Pi_N^{SPT}$;

$\Pi^{eq =}$ `equitablyOptimalSubset`(Π^{SPT});

return Π^{eq};

Algorithm

The algorithm is formally described as Algorithm 3.3 and uses Procedure `sptRecursive` to produce all SPT schedules. The algorithm starts with producing all SPT schedules on all the resources. Then, these schedules are matched in an all-to-all manner to form the set of grid schedules. Finally, the algorithm returns the grid schedules that result in equitably-optimal vectors of completion times.

Complexity

The complexity of the algorithm is determined by the number of grid schedules. In the worst case, there are $O(2^n)$ SPT schedules on each resource, which results in $O(2^{Nn})$ combinations. Given a set of k points, an algorithm returning a subset of Pareto-optimal points has a complexity of at least $k \log k$ (two criteria). Thus, the worst case complexity of the whole algorithm is in $O(Nn2^{Nn})$.

3.3.2 Scheduling with Dynamic Programming

Our dynamic programming algorithm (DYN) finds the set of Pareto-optimal grid schedules by constructing partial schedules (specifying the owners of the first i jobs on a resource) and iteratively expanding them to eventually contain all the jobs on all the resources. Finally, to find the set of equitably optimal grid schedules, the Pareto-optimal set is pruned.

Principle

By discarding partial grid schedules that are Pareto-dominated, the dynamic algorithm uses the property of the optimal substructure of Pareto-optimality. A Pareto-dominated partial grid schedule will remain Pareto-dominated by the subsequent `paretoSchedules` operations that extend the schedule by adding jobs at the end (see the next section for the definition). The following proposition formally states this result.

Proposition 3.6. *All possible grid schedules extended from a partial grid schedule that is Pareto-dominated will be Pareto-dominated.*

Proof: Let us assume that a partial grid schedule Π' resulting in completion times $[C'_1, C'_2, \ldots, C'_N]$ is Pareto-dominated by a partial grid schedule Π'' with $[C''_1, C''_2, \ldots, C''_N]$ (thus, $\forall_k, C'_k < C''_k$). Moreover, the number of scheduled jobs $j_{k,l}$ are the same for the two schedules. A series of `paretoSchedules` operations of Π' that build the grid schedule will result in adding a vector $[\Delta C_1, \Delta C_2, \ldots \Delta C_N]$. However, the same `paretoSchedules` operations can be performed on Π''. As $\forall_k, C'_k + \Delta C_k < C''_k + \Delta C_k$, the grid schedule resulting from extending Π' is Pareto-dominated by the grid schedule extended in the same way from Π''. □

Algorithm

The complete algorithm is presented as Algorithm 3.5, and the function that recursively computes all Pareto-optimal schedules is presented as Function

Function `paretoSchedules(`j, ref`)`

Input:

- j a matrix; $j[k][l]$ is the number of scheduled jobs from $\mathcal{J}_{k,l}$;
- ref a dictionary indexed by matrices defined as j, containing already computed Pareto-optimal partial schedules for j

Output: paretoOpt: Pareto-optimal partial schedules in which, for each organization O_k and each machine M_l, $j[k][l]$ jobs are scheduled

if $j ==$ *zeros(k,l)* **then** $\pi = ()$; $\pi.C = [0, \ldots, 0]$; **return** $\{\pi\}$;
if *ref.hasKey(j)* **then return** $ref[j]$;
schedules $= \emptyset$;
foreach $k \in \{1, \ldots, N\}$ **do**
  foreach $l \in \{1, \ldots, N\}$ **do**
    if $j[k][l] > 0$ **then**
      $j[k][l] -= 1$;
      currSched $=$ `paretoSchedules(`j, ref`)` ;
      $j[k][l] += 1$;
      foreach $\pi \in$ *currSched* **do**
        π.add($J_{k,l}^{j[k][l]}$) ;
        $p = p_{k,l}^{j[k][l]}$;
A $\pi.C[k] += p$;
B   **foreach** $k' \in 1, \ldots, N$ **do** $\pi.C[k'] += p(n_{k',l} - j[k][l])$;
      schedules $\cup =$ currSched ;
C paretoOpt $=$ `paretoOptimalSubset(`*schedules*`)` ;
  $ref[j] =$paretoOpt ;
  return paretoOpt ;

Algorithm 3.5: Dynamic Programming (DYN) that produces all equitably-optimal grid schedules.

Input: $\mathcal{J}_{k,l}$, sets of jobs on each processor
Output: Π^{eq}, the set of all equitably-optimal grid schedules
$j == zeros(k,l)$;
foreach $k \in \{1,\ldots,N\}$ **do**
    foreach $l \in \{1,\ldots,N\}$ **do**
        $j[k][l] = n_{k,l}$;
$ref = \emptyset$;
$\Pi^{Pareto} = pareto(j, ref)$;
Π^{eq} = `equitablyOptimalSubset(`Π^{Pareto}`)`;

`paretoSchedules`. The relation of Pareto-dominance (line C)) is computed based on the vector of total completion times for each schedule ($\pi.C$) computed in lines A-B. After adding next job $J_{k,l}^{j[k][l]}$ to schedule π, the cost caused by execution of $J_{k,l}^{j[k][l]}$ is added to $\pi.C$. Firstly (line A), $J_{k,l}^{i}$'s owner C_k is increased by $p_{k,l}^{i}$. Then (line B), for each organization $O_{k'}$, all the remaining $n_{k',l} - j[k][l]$ jobs will be delayed by $p_{k,l}^{i}$, as they will be executed after $J_{k,l}^{j[k][l]}$.

Complexity

As the algorithm produces the set of all Pareto-optimal solutions, its worst-case complexity is determined by the worst-case number of Pareto-optimal grid schedules, which is in $O(2^{Nn})$. Following the reasoning similar to ES, the worst case complexity of DYN is in $O(Nn2^{Nn})$.

Finding Equitably-Optimal Schedules by Dynamic Programming

Note that it is not possible to find equitably optimal schedules directly, as an equitably-dominated partial schedule may become dominating after the next extension. The following proposition formally states this observation:

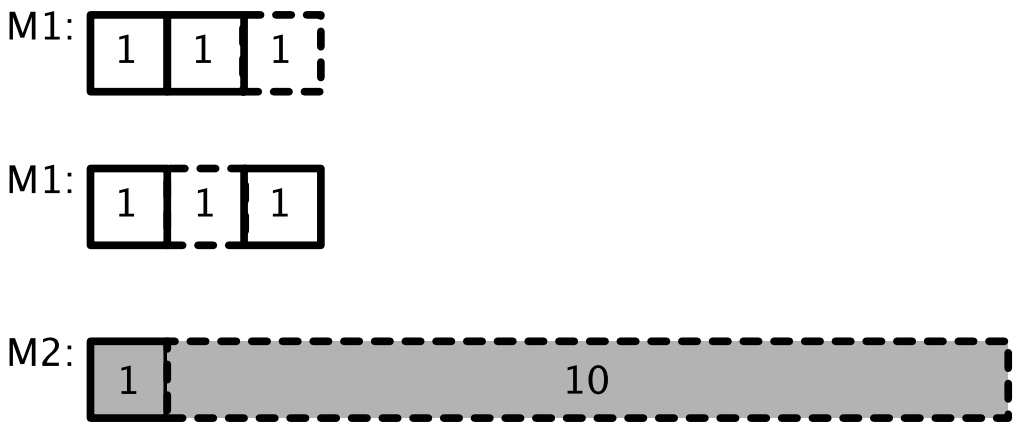


Figure 3.4: An instance in which O_1 has $n_{1,1} = 2$ unit-size ($p^i_{1,1} = 1$) jobs on M_1; O_2 has $n_{2,1} = 1$ unit-size job on M_1. On the second resource, O_1 has $n_{2,1} = 1$ unit-size job; O_2 has one job of size $p^1_{2,2} = 10$. When analyzing schedules for resource M_1, the top schedule Π' equitably dominates the middle one Π'' , as it leads to the same sum of completion times ($[C_1, C_2] = [3, 3]$). However, if the schedules are extended to contain also the bottom schedule for resource M_2, the combination of Π' and the bottom schedule (resulting in $[C_1, C_2] = [4, 14]$) is equitably-dominated by the combination of Π'' and the bottom schedule (for which $[C_1, C_2] = [5, 13]$).

Proposition 3.7. *The structure of the problem of finding the set of equitable grid schedules by extending partial schedules is not optimal.*

Proof: We show an instance and an equitably-dominated partial schedule that becomes dominating after a subsequent extension.

Consider a grid formed by two organizations O_1, O_2 (see Figure 3.4). Two of Pareto-optimal schedules after scheduling all the jobs on the first processor are $\pi' = (J^1_{1,1}, J^2_{1,1}, J^1_{2,1})()$ with $[C'_1, C'_2] = [3, 3]$ and $\pi'' = (J^1_{1,1}, J^1_{2,1}, J^2_{1,1})()$ with $[C''_1, C''_2] = [4, 2]$. Note that Π'' result $[4, 2]$ is equitably dominated by Π' result $[3, 3]$ (by the principle of transfers). Yet extending both schedules by scheduling the shorter job before the longer one on the second processor (so that M_2 schedule is $(J^1_{1,2}, J^1_{2,2})$ results in completion times of $[4, 14]$ for Π' and $[5, 13]$ for Π''. Now, it is Π'' which is equitably-dominating Π' (again by the principle of transfers). □

3.3.3 Equitable Walk (EW)

Equitable Walk (EW) is a heuristics which produces a number of grid schedules by iterative modifications of the initial **SPT** schedule in order to improve the outcome of the organization disfavored by the **SPT** schedule. The algorithm is based on the taboo-search paradigm, as it stores a list of already considered schedules in order not to revisit them and thus possibly stop in a local optimum. The resulting schedules can be equitable: the algorithm may produce non-equitable schedules and not all possible equitable schedules may be produced.

Principle

The algorithm modifies the schedules by iteratively *switching* the order of two jobs executed one after another on the same resource. Each iteration starts with the selection of the organization with the worst total completion time in the current schedule. Then, the algorithm temporary switches each job of this organization and computes the resulting vectors of total completion times $[C_k]$. Finally, the algorithm advances the job which resulted in equitably-optimal $[C_k]$ (regarding all the temporary moves performed).

The algorithm starts with **SPT** schedule. The following result formally states that there is a **SPT** schedule that is equitable. Note that there is more than one **SPT** schedule iff at least two organizations have at least two of equal sizes on the same resource.

Corollary 3.3. *There is a* **SPT** *schedule that is equitably-optimal.*

Proof: This is a direct consequence of Theorem 1.1. Each **SPT** schedule is optimal with regard to the sum of completion times of all jobs on all resources $(\sum_k \sum_l \sum_i C^i_{k,l})$. Therefore, each such a schedule minimize the sum of sum of completion times of respective organizations $(\sum_k C_k)$. Hence, each **SPT** gives minimal value of $\bar{\boldsymbol{\theta}}_N$, the last criterion of IEMOP. Consequently, in IEMOP, only a **SPT** schedule can Pareto-dominate another **SPT** schedule.

Thus, there is at least one **SPT** schedule that is not Pareto-dominated in IEMOP. □

Algorithm

The complete algorithm is presented as Algorithm 3.6. EW starts with ordering jobs $\mathcal{J}_{\cdot,q}$ on every resource M_l according to SPT (line 1). Then, the organization O_k with the largest C_k is selected (line 6). Then, for each O_k's job $J^i_{k,l}$ (line 12), the job is temporary advanced (line 13) by switching $J^i_{k,l}$ with the job executed immediately before. If the resulting schedule has not been considered yet (line 14), the algorithm computes the resulting vector of completion times (lines 16-18) to chose the job resulting in equitably-optimal total completion times (line 19). Finally, the optimal job is advanced (lines 23-25). The algorithm repeats such modifications until no further improvement is possible (line 3).

The output of the algorithm is a list of grid schedules. The algorithm starts with a **SPT** schedule. If there is only one **SPT** schedule, it is equitably-optimal. If there are more **SPT** schedules, it is not guaranteed that the algorithm finds the one which is equitably-optimal. Moreover, the equitableness of the rest of the schedules is not guaranteed. Therefore, before returning, the algorithm removes the equitably-dominated schedules (line 26) from the resulting list.

The algorithm may not produce all possible equitable solutions (the equitable part of the Pareto-front), because it may be trapped in a local optimum. Let us consider an instance in which two organizations O_1 and O_2 produced jobs only for one machine (i.e. $\mathcal{J}_{1,2} = \mathcal{J}_{2,2} = \emptyset$). Two organizations have identical sets of two jobs of sizes $p^1_{1,1} = p^1_{2,1} = 1$ and $p^2_{1,1} = p^2_{2,1} = 2$. There are exactly two equitably-optimal schedules of this instance, i.e. $(J^1_{1,1}, J^1_{2,1}, J^2_{2,1}, J^2_{1,1})$ resulting in completion times $[7, 6]$ and $(J^1_{2,1}, J^1_{1,1}, J^2_{1,1}, J^2_{2,1})$ resulting in completion times $[6, 7]$. Let EW start with a SPT schedule $(J^1_{2,1}, J^1_{1,1}, J^2_{2,1}, J^2_{1,1})$ with completion times $[8, 5]$. In the next step, EW improves O_1 result by

Algorithm 3.6: Equitable Walk

Input: $\mathcal{J}_{k,l}$, sets of jobs on each processor

Output: Π^{EWF}, the set of possibly equitably-optimal grid schedules

$\pi = \mathbf{SPT}$;
$taboo = \emptyset$; $\Pi^{EW} = \{\Pi\}$;
repeat
  $C = \pi.C$;
  $job^* = null; C^* = [\infty]; i^* = 0$;
  $k = argmax_{k'}(C[k'])$;
  foreach $l \in \{1, \ldots, N\}$ **do**
    foreach $i \in \{2, \ldots, n_{\cdot,l}\}$ **do**
      $\pi_l = \pi[l]$;
      $job = \pi_l[i]$;
      $prevJob = \pi_l[i-1]$;
      if $job \in \mathcal{J}_{k,l} \wedge prevJob \notin \mathcal{J}_{k,l}$ **then**
        $\pi_l[i] = prevJob; \pi_l[i-1] = job$;
        if $\pi \notin taboo$ **then**
          $k' = prevJob.owner$;
          $C' = C$;
          $C'[k'] += job.size$;
          $C'[k] -= prevJob.size$;
          if $C' \prec_e C^*$ **then**
            $job^* = job; degr^* = degr; C^* = C'; i^* = i$;
        $\pi_l[i] = job; \pi_l[i-1] = prevJob$;
  if $job^* \neq null$ **then**
    $l = job^*.machine$;
    $\pi[l][i^*] = \pi[l][i^*-1]; \pi[l][i^*-1] = job$;
    $taboo \cup = \{\pi\}; \Pi^{EW} \cup = \{\pi\}$;
until $job == null$;
$\Pi^{EWF} =$ `equitablyOptimalSubset(`Π^{EW}`)`;
return Π^{EWF};

advancing $J^2_{1,1}$, which results in schedule $(J^1_{2,1}, J^1_{1,1}, J^2_{1,1}, J^2_{2,1})$ and completion times $[6, 7]$. EW tries now to improve O_2 result, however the only possible move is to delay $J^2_{1,1}$, a schedule which has been already visited. Consequently, EW stops.

Complexity

The worst-case complexity of EW is exponential as the algorithm is able to produce the set of all Pareto-optimal solutions. Contrary to DYN, however, EW can be stopped at any moment (e.g. when the time available for decision expires). The partial solution is the incomplete list of possibly equitable grid schedules.

3.4 Game-Theoretic Model

In this section we assume that each organization O_k controls its local resource M_k and therefore is able to impose the scheduling of jobs $\mathcal{J}_{\cdot,k}$. Each organization wants to minimize the sum of completion times C_k of its jobs $\mathcal{J}_{k,\cdot}$. However, C_k depends also on the completion time of O_k's remote jobs $\mathcal{J}_{-k,\cdot}$, which in turn depends on the schedules imposed by other organizations. Game theory studies the problems in which players maximize their returns which partially depend on actions of other players. Consequently, it seems to be an adequate tool to study this model.

Definition 3.1. *A* grid scheduling game *is a game in the normal form where:*

- *the set of players is equal to the set of organizations $\mathcal{O}$;*
- *a strategy σ_k of a player O_k is an ordering of $\mathcal{J}_{\cdot,k}$ (following the local SPT rule from Proposition 3.1, as all the other strategies are Pareto-dominated);*
- *a payoff function $u_k(\boldsymbol{\sigma})$ for a player O_k resulting from a profile of strategies $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_N]$ is the reduction of the player's sum of completion*

times with comparison to the selfish outcome of the game:

$$u_k(\boldsymbol{\sigma}) = C_k(\mathbf{MJF}) - C_k(\boldsymbol{\sigma}).$$

Note that the aforementioned definition specifies a game that is not repeated (it is a so-called one-shot game). This is caused by the fact that the model analyzed in this chapter is off-line.

A selfish player O_k can use a greedy *My Jobs First* (MJF) strategy, which schedules all the local jobs $\mathcal{J}_{k,k}$ before any foreign job. Given any strategies of the rest of organizations, MJF strategy will reduce the total finish time C_k, by advancing local jobs $\mathcal{J}_{k,k}$. Therefore, MJF is the *prudential strategy* for each player. However, a solution in which every organization schedules jobs according to MJF (denoted as **MJF**) very often is not optimal and it is Pareto-dominated by other solutions. The goal of the grid scheduler is to propose a schedule at least Pareto-dominating **MJF**. That is why we define the payoff $u_k(\boldsymbol{\sigma})$ for each player O_k as the *gain over* **MJF** for that player. The payoffs defined in such a way must be maximized. Any profile of strategies $\boldsymbol{\sigma}$ resulting, for a player O_k, in $u_k(\boldsymbol{\sigma}) < 0$ is not feasible. The proposed total execution time for O_k's jobs is greater than the longest possible total execution time when O_k decides to order jobs on its resource according to MJF. Therefore, O_k would play MJF, which would also reduce other payoffs.

We analyze this game from two different perspectives. Firstly, in Section 3.5, we assume that there is no cooperation between the players. The grid scheduler proposes a schedule (a strategy) for each player. However, players are not obliged to follow the strategies proposed. We show that such a game is analogous to the well-known Prisoner's Dilemma (PD) game. Consequently, shortsighted players will tend to chose the greedy strategy, which results in suboptimal performance of the grid. The price the grid users pay for such lack of control is high. The *price of anarchy*, or the ratio between total completion times achieved in Nash solution and in the optimal one, may be as high as the number of foreign jobs. Secondly, in Section 3.6, we increase the power of the grid scheduler. If the scheduler proposes a schedule resulting

in positive payoff for each player, the players must follow the schedule. We show how to chose a fair schedule in such setting. This corresponds to the constrained optimization approach.

3.5 Non-cooperative Game

In this section we assume that each organization O_k controls its local resource M_k and therefore is able to impose a schedule for $\mathcal{J}_{\cdot,k}$.

Proposition 3.8. MJF *is the only Nash equilibrium of the one round, non-cooperative grid scheduling game.*

Proof: Let us assume that, for a particular job configuration $\mathcal{J} = \{J_{k,l}\}$, a grid scheduler is able to produce schedule $\boldsymbol{\sigma^*} = [\sigma_1^*, \ldots, \sigma_n^*]$, which results in positive payoff $u_k(\boldsymbol{\sigma^*}) > 0$ for each player. Consequently, there must be at least one player O_k, for whom the proposed strategy σ_k^* is different than MJF. Thus, in M_k's schedule, there is at least one foreign job $J_{l,k}^i$ scheduled before a local job $J_{k,k}^j$. If O_k decides to switch the order of execution of these two jobs, the local job $J_{k,k}^j$ will be finished faster and, consequently, the player's payoff u_k will increase. At the same time the payoff of the owner of the delayed job u_l will decrease. It follows that the strategy maximizing u_k is MJF, given that the others play any profile of strategies $\boldsymbol{\sigma}_{-k}$. Additionally, if all the other players play MJF, the only strategy which guarantees non-negative u_k for O_k is to play MJF as well. □

Consequently, the game is analogous to the multi-player Prisoner's Dilemma (PD). The equivalent of the mutual collaboration in PD is when organizations follow scheduler's suggestion $\boldsymbol{\sigma^*}$. When one of the organizations O_k play $\sigma_k = MJF$, whereas the others play σ^* (a single betrayal in PD), O_k's payoff increases $u_k(MJF, \sigma_{-k}^*) > u_k(\sigma^*)$, and the others lose. If everyone plays MJF (multiple betrayal in PD), resulting payoff for each organization is 0.

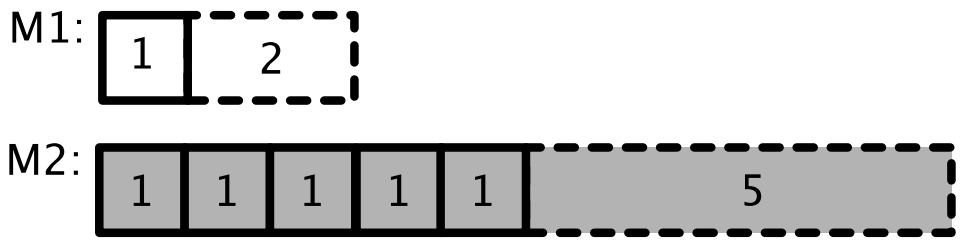


Figure 3.5: A globally optimal schedule may be unfair to one of the organizations (O_2), which experience its performance degraded comparing to locally-optimal schedule.)

In order to measure the loss of performance experienced by players in the Nash equilibrium, we firstly characterize the globally-optimal outcome of the game.

Corollary 3.4. SPT *maximizes the sum of outcomes,* $\sum u_k$. *Therefore,* **SPT** *is the social optimum of the scheduling game.*

Proof: The result is a direct consequence of the optimality of SPT schedule on one resource. On each resource M_l, such schedule minimizes $\sum_i C^i_{\cdot,l}$. Therefore, **SPT** minimizes $\sum_l \sum_i C^i_{\cdot,l} = \sum_k \sum_l \sum_i C^i_{k,l} = \sum_k C_k$. Recall that $\sum_k u_k = \sum_k C_k(\mathbf{MJF}) - \sum_k C_k$. $\sum_k C_k(\mathbf{MJF})$ is constant and $\sum_k C_k$ is minimized by **SPT**. Therefore **SPT** maximizes the sum of the outcomes of players. □

Note that **SPT** may not be fair to all organizations, nor, in general, feasible (see Figure 3.5). In this instance, **SPT** leads to $(C_1(\mathbf{SPT}), C_2(\mathbf{SPT})) = (16, 13)$, that gives the minimum sum of completion times measured on the system level $\Sigma C = 29$. However, the optimality on the system level is not reflected by O_2's result that is worse than in **MJF** strategy: $(C_1(\mathbf{MJF}), C_2(\mathbf{MJF})) = (41, 8)$.

When each player plays MJF, the resulting payoffs are 0, therefore everyone feels a performance drop. The following result shows that the price the grid pays for the lack of control is considerable.

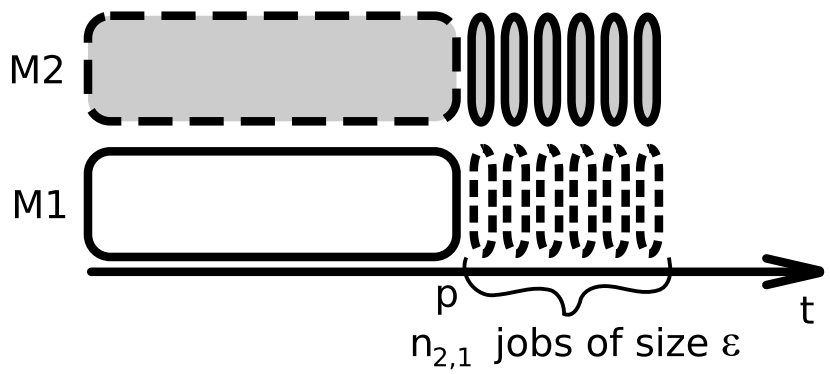


Figure 3.6: An instance with one large local job and n small foreign jobs on each resource which leads to a price of anarchy in $O(n)$.

Proposition 3.9. *The Price of Anarchy (PoA) is at least linear with the number of jobs.*

Proof: Consider an instance (Figure 3.6), in which two resources M_1 and M_2 have identical loads: one local job of size $p^1_{1,1} = p^1_{2,2} = p$ and $n_{2,1} = n_{1,2}$ foreign jobs, each of size $p^i_{2,1} = p^i_{1,2} = \epsilon$. A **MJF** schedule results in sum of completion times of:

$$\begin{aligned} C_1(\mathbf{MJF}) &= p + (p+\epsilon) + (p+2\epsilon) + \cdots + (p+n_{1,2}\epsilon) = \\ &= (n_{1,2}+1)p + \frac{1}{2}\epsilon n_{1,2}(n_{1,2}+1), \end{aligned}$$

whereas an optimal schedule, in which the small foreign jobs are executed before the local job on both resources, results in total completion time of:

$$\begin{aligned} C^*_1 &= (\epsilon + 2\epsilon + \ldots + n_{1,2}\epsilon) + (n_{1,2}\epsilon + p) = \\ &= \frac{1}{2}\epsilon n_{1,2}(n_{1,2}+1) + (n_{1,2}\epsilon + p). \end{aligned}$$

The *price of anarchy PoA* is the ratio between the result in the worst Nash equilibrium and the socially-optimal result.The price of anarchy is equal to:

$$PoA = \frac{2C_1(\mathbf{MJF})}{2C^*_1} \underset{\epsilon\to 0}{\longrightarrow} n+1.$$

□

Note that in real-world grids scheduling will be repeated, in function of new jobs arrivals. This would lead to multiple iterations of the game presented above. In Infinitely Repeated PD, cooperation (following $\boldsymbol{\sigma}^*$) becomes profitable, as a single betrayal (playing MJF) can be punished during the next rounds by other players (by refusing to cooperate with the free-rider). Consequently, finding a good $\boldsymbol{\sigma}^*$ becomes an important problem, analyzed in the next section. However, the scheduling game will be not repeated infinitely, as collaboration in the Grid (e.g. in one Virtual Organization) is rather short-term, therefore the "shadow of the future" is reduced. In PD repeated N times (N known to players), defection is again the only Nash equilibrium.

Note also that each player can verify if the other player is executing the collaborative version of the schedule before completing the whole schedule (by examining completion times of his/her jobs executed on the other processor). This enables on-line verification of the other player and, consequently, almost immediate punishment of the non-cooperative behavior by changing the schedule. However, the player that executes a foreign job first (according to the cooperative schedule) does not know if the other player will cooperate or not. Such an uncertainty transforms the on-line game into a series of finitely-repeated off-line games, that can be analyzed similarly to the game analyzed in this section.

3.6 Constrained Optimization Approach

In this section, we assume that the community, represented by the centralized grid scheduler, is able to impose a scheduling on the players, if the resulting payoffs $\boldsymbol{u}^* = [u_1, \ldots, u_n]$ are *feasible*, i.e. $u_k \geq 0$ for each organization O_k. This guarantees that C_k does not rise in comparison with $C_k(\mathbf{MJF})$. Moreover, $\boldsymbol{u}^*$ should be equitably-optimal, as no organization should be preferred *a priori*.

To produce such solutions, we may use EW, the same method as in the optimization approach. EW must be adjusted to maximize u_k, instead of

minimizing C_k. We call the adjusted algorithm Adjusted Equitable Walk (AEW). AEW starts with a **SPT** schedule. Note that one of such schedules is still equitable (Corollary 3.3 holds), as **SPT** maximizes the sum of all the criteria (see Corollary 3.4). However, **SPT** may not be feasible, and, generally, AEW may be unable to produce a feasible result.

3.7 Experiments

In this section we present experimental validation of the proposed optimization algorithms. Firstly, we compare the performance of two exact algorithms: naive exhaustive search (ES) and dynamic programming (DYN). Then, we compare the solutions produced by heuristic algorithms (EW and AEW) to the set of equitably optimal solutions extracted from the results of the exact algorithms. In these experiments, the instances considered are relatively small, as exhaustive search has an exponential complexity. Finally, we show the results of EW and AEW applied to larger instances.

3.7.1 Methods

In this section we briefly describe the experimental setup used to evaluate the proposed algorithms.

All the algorithms were implemented in Python and used common code base (e.g. the same function that returned Pareto-optimal solutions). To optimize the run-time, the algorithms were profiled to remove basic performance bottlenecks and then precompiled with Psyco [Psyco, 2007]. Tests that compared the exact algorithms with greedy heuristics were executed on a single node running Linux 2.6.15 kernel with one Pentium 4 3.0Ghz processor and 1GB of RAM. The tests of greedy algorithms on larger data sets were executed on 20 nodes of a cluster composed of Intel Bi-Xeon 2.4 GHz dual-core processors, with 1.5 GB of RAM on each node. In all tests, the background load was negligible.

Workloads

The algorithms were evaluated on randomly generated workloads and on some specific workload. The random workload was generated as follows. Each resource M_k has the same number of jobs $n_{\cdot,k}$ (i.e. $n_{\cdot,k}$ is the parameter of experiments). The number of jobs of each organization on each processor is distributed uniformly: e.g. in a two-organizational grid, $n_{1,k}$ is uniformly distributed over $\{0, 1, \ldots n_{\cdot,k}\}$, and $n_{2,k} = n_{\cdot,k} - n_{1,k}$. Jobs' sizes $p^i_{k,l}$ are uniformly distributed over $\{1, \ldots, \max p^i_{k,l}\}$ ($\max p^i_{k,l}$ is a parameter experiments). For each combination of $n_{\cdot,k}$ and $\max p^i_{k,l}$, 50 instances were generated. Using this method, we generated the following set of workloads:

- *small* data set, used for the evaluation of exact algorithms, with $N = 2$ organizations, $n_{\cdot,k} \in \{5, 8, 10, 11, 12\}$ and $\max p^i_{k,l} \in \{5, 10, 20, 50\}$
- *small-3* data set, used for the evaluation of exact algorithms, with $N = 3$ organizations, $n_{\cdot,k} \in \{4, 5, 6\}$) and $\max p^i_{k,l} = 10$
- *large* data set, used for large-scale evaluation of greedy heuristics with $N \in \{2, 3, \ldots, 8\}$ organizations, $n_{\cdot,k} \in \{10, 15, 20\}$ and $\max p^i_{k,l} = 10$

We also used an artificial data set composed of instances depicted in Fig. 3.2. Two organizations have identical load of $n_{1,1} = n_{2,1} = n$ jobs of increasing sizes $(1, 2, 4, 8, \ldots, 2^{n-1})$. There are no jobs for the second resource ($n_{1,2} = n_{2,2} = 0$). Such a series of instances of increasing size enables us to directly measure the impact the increased number of Pareto-optimal solutions on the performance of the algorithm.

Quality indicators

All the algorithms produce a number of equitably-optimal solutions, yet a grid scheduling system would use only one of them. This motivates the need to measure the quality of the whole set by evaluating one of the returned solutions.

To measure the quality of the proposed solutions, we choose two solutions from the set of equitably-optimal schedules:

- a *min-sum* solution with the minimum sum of completion times of organizations, when optimizing $[C_k]$, or the maximum Σu_k, when constrained optimization is used. Such a solution can be interpreted as the best solution from the community point of view.

- A *min-max* solution that minimizes the maximum completion time of an organization; or maximizes the the minimum organization's improvement u_k in constrained optimization. Such a solution is the "fair" solution in the classic sense.

Note that the *min-sum* solution for the unconstrained optimization (i.e. optimization of $[C_k]$) is a **SPT** solution. However, in the case of constrained optimization, we consider only feasible (i.e. $\forall_k u_k \geq 0$) solutions, thus **SPT** is the *min-sum* solution only if it is feasible.

Then, we compare both solutions with the **MJF** solution. For each solution type (*min-sum* and *min-max*), we calculate the improvement over **MJF** regarding two measures: the sum of completion times of organizations (representing the social goal) and the worst completion time of an organization (representing the fair goal). We can define the *min-sum sum score* as:

$$1 - \frac{\sum_k C_k(\textit{min-sum})}{\sum_k C_k(\mathbf{MJF})},$$

and the *min-sum max score* as:

$$1 - \frac{\max_k C_k(\textit{min-sum})}{\max_k C_k(\mathbf{MJF})}.$$

Two measures for the *min-max* solution (*min-max sum score* and *min-max max score*) are defined similarly.

Table 3.1: Average run-time of DYN and ES algorithm over randomized instances in function of $n_{\cdot,k}$. *small* data set. Each row is an average from 200 random instances.

	run time [s]	
$n_{\cdot,k}$	DYN	ES
5	0.01	0
8	0.16	0.05
10	0.87	1.18
11	1.83	5.3
12	3.91	36.78
average	1.36	8.66

3.7.2 Exact Algorithms

Run-time

Both the ES and the dynamic programming algorithm have the same exponential worst-case complexity. The complexity of the ES is determined by the number of possible SPT schedules. DYN is able to discard partial schedules that are Pareto-dominated, and thus substantially reduce the number of solutions considered. Thus, even though Section 3.2.1 showed that the number of Pareto-optimal schedules can be exponential, dynamic programming should be faster. The results are presented in Tables 3.1, 3.2 and Figure 3.7 for *small* data set, Table 3.3 and Figure 3.8 for exponential instances and Table 3.4 for *small-3* data set.

In the random data set, on the average, the DYN algorithm performed significantly faster than the ES in larger instances ($n_{\cdot,k} \geq 10$). Moreover, the average runtime of the ES algorithm was more sensitive to the increase in $\max p^i_{k,l}$. However, the actual linear correlation coefficients (Pearson's product-moment coefficient) between $\max p^i_{k,l}$ and run-times were small (0.041 for the exhaustive search, 0.030 for DYN). This was caused by large standard deviations of the run-times.

Table 3.2: Average run-time of DYN and ES algorithm over randomized instances in function of $\max p^i_{k,l}$. *small* data set. Each row is an average from 200 random instances.

	run time [s]	
$\max p^i_{k,l}$	DYN	ES
5	1.39	6.54
10	1.23	8.03
20	1.31	9.96
50	1.5	10.11

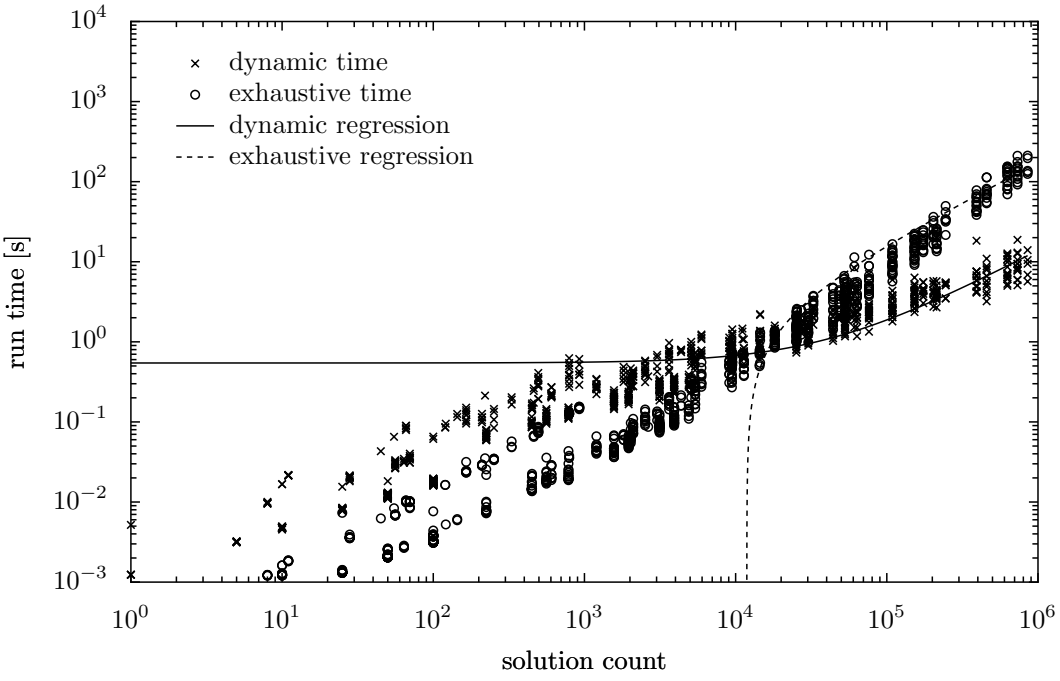


Figure 3.7: Comparison of the run-time of DYN and ES plotted against the total number of possible schedules. *small* data set. Each dot represents one run of the algorithm over one instance.

All the observed differences are statistically significant (using paired, two-tailed t-test, we obtained $p < 0.0001$).

Detailed analysis revealed that the run-time of both algorithms was almost perfectly linearly correlated with the total number of possible SPT

Table 3.3: Comparison of the run-time of DYN and ES over exponential instances.

	run time [s]		solution count	
n	dynamic	exhaustive	all	pareto
3	0.01	0.01	20	16
4	0	0.01	70	44
5	0.01	0.03	252	116
6	0.04	0.15	924	278
7	0.15	0.69	3432	620
8	0.65	3.8	12870	1324
9	2.8	33.38	48620	2752
10	12.56	275.09	184756	5628
11	70.61	2126.75	705432	11400

schedules. For the ES, the resulting correlation coefficient is 0.958, whereas for the DYN the coefficient is not as high (0.882), but the dependence is clearly visible. Figure 3.7 presents the run-time of both algorithms plotted against the number of possible SPT schedules. We can see that the linear regressions become close to the results observed for run-times greater than a few seconds. DYN algorithm is almost an order of magnitude slower on small instances, but becomes more than an order of magnitude faster on large instances.

Similar phenomena can be observed in exponential instances (Table 3.3 and Figure 3.8). We see that DYN algorithm is faster for all but the smallest instance. Both run times are almost perfectly linearly correlated with the number of solutions ($\rho = 0.991$ for the ES, $\rho = 0.996$ for the DYN) and the number of Pareto-optimal solutions ($\rho = 0.927$ for the ES, $\rho = 0.944$ for the DYN). However, the run time of the DYN algorithm rises much slower with the total number of solutions compared to the ES. The linear regression coefficient is equal to $9.78E-5$ for the DYN and $2.91E-3$ for the ES.

Note also that the number of all solutions rises much faster than the number of Pareto-optimal ones. Therefore, an algorithm that does not have

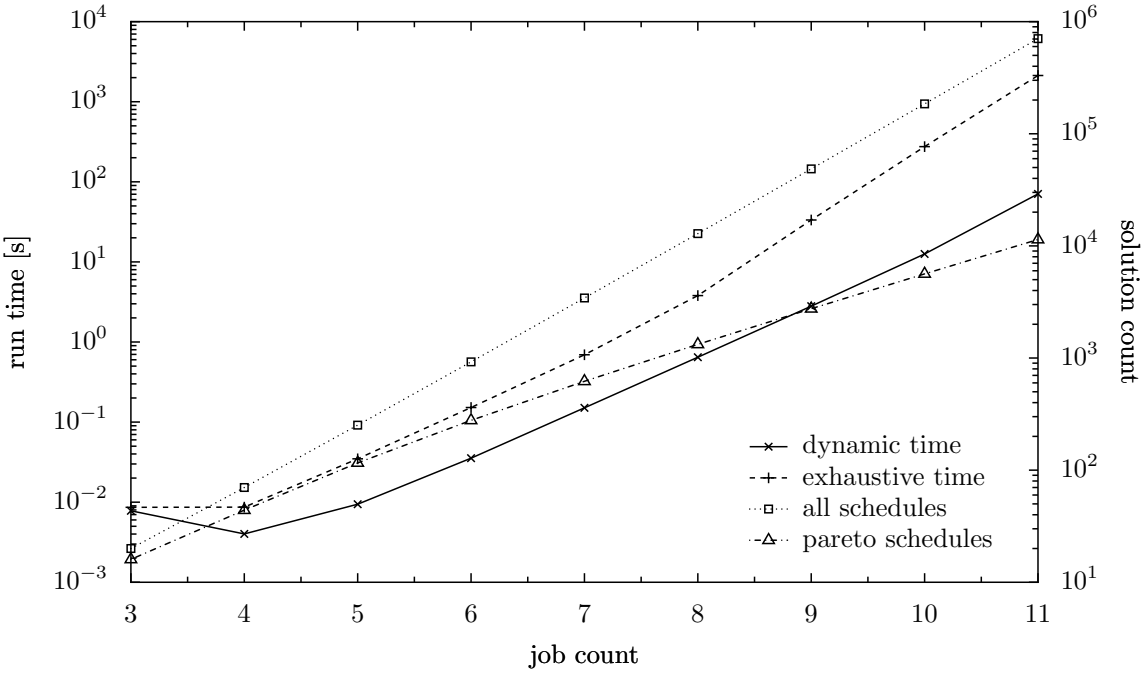


Figure 3.8: Comparison of the run-time of DYN and ES over exponential instances (left axis); the total number of schedules and the number of Pareto-optimal schedules (right axis).

to go over all the solutions will be faster for larger instances.

When the number of organizations is increased, the number of criteria, the number of possible solutions, and thus the number of Pareto-optimal solutions raises. Thus, the exact algorithms start to require large amount of memory and, eventually, start swapping. Consequently, the run-time of the exact algorithms becomes unacceptable even for small instances. During our initial tests, a simple instance with $N = 4$ organizations and $n_{\cdot,l} = 4$ jobs on each processor (with sizes $\mathcal{J}_{1,1} = \{3, 7\}$,$\mathcal{J}_{1,2} = \{5\}$,$\mathcal{J}_{1,3} = \{1\}$, $\mathcal{J}_{1,4} = \{2\}$, $\mathcal{J}_{2,1} = \{6, 6\}$, $\mathcal{J}_{2,2} = \{1, 2\}$, $\mathcal{J}_{2,3} = \{1, 8\}$, $\mathcal{J}_{2,4} = \emptyset$, $\mathcal{J}_{3,1} = \{8\}$, $\mathcal{J}_{3,2} = \{5\}$, $\mathcal{J}_{3,3} = \{1\}$, $\mathcal{J}_{3,4} = \{2\}$, $\mathcal{J}_{4,1} = \emptyset$, $\mathcal{J}_{4,2} = \{6\}$, $\mathcal{J}_{4,3} = \emptyset$, $\mathcal{J}_{4,4} = \{1, 5\}$) was not computed by DYN in more than 15 hours. Therefore, we experimented on smaller instances with $N = 3$ organizations. However, we were able to experiment only on relatively small instances ($n_{\cdot,l} < 6$), as ES was not able to finish in more than a week of constant computation in a number of instances

for $n_{\cdot,l} = 6$ (e.g. with sizes $\mathcal{J}_{1,1} = \{3, 3, 9\}$ $\mathcal{J}_{1,2} = \{1, 6\}$ $\mathcal{J}_{1,3} = \{3, 4, 9\}$, $\mathcal{J}_{2,1} = \{1, 4\}$, $\mathcal{J}_{2,2} = \{2, 2\}$, $\mathcal{J}_{2,3} = \{2\}$, $\mathcal{J}_{3,1} = \{5, 9\}$, $\mathcal{J}_{3,2} = \{2, 8\}$, $\mathcal{J}_{3,3} = \{4, 6, 8\}$).

Table 3.4: Comparison of the run-time of DYN and ES over random instances. *small-3* data set.

	run time [s]	
n	dynamic	exhaustive
4	1.68	0.27
5	19.69	4.05

Table 3.4 presents aggregated results. We see that ES is faster, which is caused by the fact that instances considered are relatively small (we observed similar phenomena for data sets with two organizations). Note also that the runtime (for $n = 5$) is about three orders of magnitude greater than the runtime in case of 2 organizations.

Number of Solutions

In the worst case, the number of Pareto-optimal solutions can be exponential on the size of the instance. In this section we investigate if this phenomena can be observed frequently in random data set. Also, as equitable solutions are a subset of Pareto-optimal solutions, we measure if the number of resulting solutions is substantially reduced. Aggregated results are presented in Table 3.5, Table 3.6 and in Figure 3.9.

A number of interesting relationships can be observed. Firstly, the number of optimal solutions of all the types grows with the size of the instance $n_{\cdot,k}$. The growth of the number of all solutions is almost perfectly exponential, as the Pearson's correlation coefficient between the instance size and the logarithm of the number of all solutions is equal to $\rho = 0.780$ (compared to $\rho = 0.410$ for the number itself). Moreover, the number of Pareto-solutions also grows exponentially. Correlation coefficients in this case are $\rho = 0.808$

for the logarithm and $\rho = 0.369$ for the number itself. However, the relationship between the number of equitably optimal solutions and the instance size is not as clear. In unconstrained optimization, the correlation coefficients are $\rho = 0.347$ for raw number and $\rho = 0.490$ for the logarithm. In constrained optimization, the correlation coefficients are $\rho = 0.349$ for raw number and $\rho = 0.407$ for the logarithm. Consequently, the difference is small. That result leaves the question of the complexity of finding all the equitable solutions open.

When the sizes of jobs are similar (as in the first row of Table 3.6), there are significantly more Pareto-optimal solutions (two tailed type-3 t-test, $p < 0.001$), even though the number of all solutions is the same. However, the number of equitable solutions is the same (two tailed type-3 t-test, $p > 0.1$). Also, all the other differences are insignificant.

With the increased number of organizations ($N = 3$, Table 3.7), the number of all and Pareto-optimal solutions rises considerably in comparison with two organizations. Note that we were not able to run ES algorithm on some instances with $n_{\cdot,k} = 6$ due to large runtime (see previous section), thus we do not have the results on the number of exact solutions. The increase in the number of equitably-optimal solutions is also visible.

Table 3.5: Average number of solutions and standard deviations σ in function of $n_{\cdot,k}$. *small* data set.

	all solutions		pareto-optimal		equitable			
					unconstrained		constrained	
$n_{\cdot,k}$	mean	σ	mean	σ	mean	σ	mean	σ
5	57.3	32.4	19.4	7.8	4.8	3.8	3.5	2.5
8	1630.7	1501.4	82.0	44.3	12.1	10.9	7.3	6.8
10	18998.6	19852.4	210.0	184.1	22.7	20.6	11.3	11.1
11	59359.9	66803.4	341.0	442.1	26.7	27.5	16.3	21.8
12	225004.9	261132.4	531.9	803.1	42.5	63.3	19.3	21.4
average	61010.3	147442.1	236.9	456.7	21.8	35.0	11.5	15.9

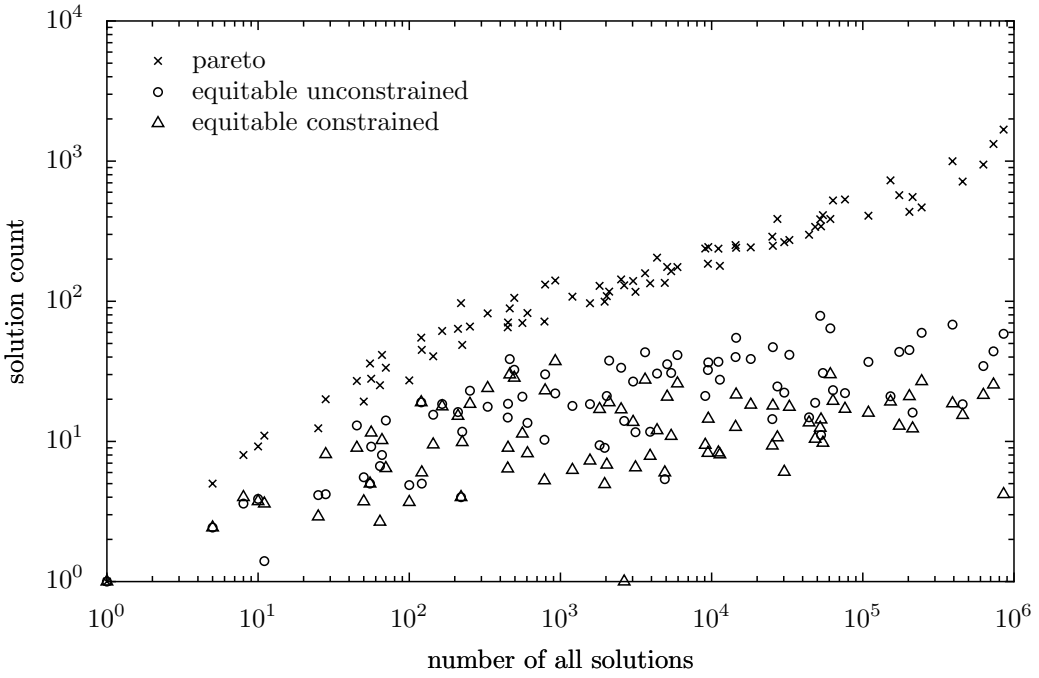


Figure 3.9: Number of Pareto-optimal solutions and equitably optimal solutions (for constrained and unconstrained optimization) plotted against the total number of solutions. *small* data set.

Table 3.6: Average number of solutions and standard deviations σ in function of $\max p^i_{k,l}$. *small* data set.

	all solutions		pareto-optimal		equitable			
					unconstrained		constrained	
$\max p^i_{k,l}$	mean	σ	mean	σ	mean	σ	mean	σ
5.0	63070.1	153116.6	416.6	845.2	24.1	50.7	12.4	20.4
10.0	63544.8	145851.6	183.6	183.6	19.4	34.1	8.8	13.5
20.0	64030.6	148650.7	165.1	142.0	20.6	23.4	11.2	13.3
50.0	53395.5	142569.5	182.2	158.6	22.9	25.2	13.8	15.1
average	61010.3	147442.1	236.9	456.7	21.8	35.0	11.5	15.9

Table 3.7: Average number of solutions and standard deviations σ in function of $n_{\cdot,k}$. *small-3* data set.

	all solutions		pareto-optimal		equitable			
					unconstrained		constrained	
$n_{\cdot,k}$	mean	σ	mean	σ	mean	σ	mean	σ
4	1264.8	1174.5	198.2	114.8	7.2	7.4	4.4	3.3
5	13251.2	21163.1	581.4	377.0	18.3	25.4	7.2	6.8
6	n.a.	n.a.	1766.3	1036.4	29.5	38.6	12.9	17.3
average	7258.0	16082.3	848.6	923.5	18.4	28.3	8.2	11.4

Profile of Solutions

In this section we measure how *min-sum* and *min-max* solutions improved the non-cooperative **MJF** solution. Table 3.8 presents the aggregated results for unconstrained optimization of $[C_k]$, whereas Table 3.9 – for constrained optimization of $[u_k]$ where unfeasible solutions are discarded. Tables 3.10 and 3.11 present results for instances with $N = 3$ organizations.

The impact of the maximum size of the job $\max p^i_{k,l}$ is almost insignificant, therefore we do not present these results here. The worst scores are achieved for $\max p^i_{k,l} = 5$. However, the difference between averages scores for $\max p^i_{k,l} \in 10, 20, 50$ are not significant (e.g. p value for two-tailed t-test of type 3 comparing scores for $\max p^i_{k,l} = 10$ and $\max p^i_{k,l} = 20$ is 0.74).

Firstly, we see that, on the average, **MJF** is a good strategy. In unconstrained optimization, *min-sum* solution is **SPT** solution that is optimal for $\sum C_k$. Yet, on the average, **SPT** is only 12% better than **MJF**.

Secondly, constraint optimization does not affect much $\sum C_k$, as the average $\sum C_k$ for *min-sum* solutions is 10% better than **MJF**. Somewhat surprisingly, the worst completion time in *min-sum* solutions (*max min-sum*) is improved more in unconstrained optimization (9.3% compared to constrained optimization's 8.5%). Yet, a detailed analysis of the results reveals that in 232 instances the maximum total completion time was worse than in **MJF**

Table 3.8: Mean scores (in percents) and standard deviations σ (in percentage points) of solutions returned by exact methods in unconstrained optimization (multicriteria minimization of $[C_k]$). *small* data set. Each row is an average over 200 instances.

	min-max				*min-sum*			
	sum		max		sum		max	
$n_{\cdot,k}$	mean	σ	mean	σ	mean	σ	mean	σ
5	11.1	7.3	7.8	14.5	5.1	9.7	18.6	10.4
8	11.7	6.7	9.0	14.1	6.4	8.9	20.4	10.7
10	12.5	6.3	10.7	12.2	6.9	9.0	22.2	9.9
11	11.9	5.7	10.1	13.0	7.0	8.4	21.1	9.9
12	13.1	6.1	11.5	13.5	8.2	9.2	22.6	10.3
average	12.1	6.5	9.8	13.5	6.7	9.1	21.0	10.3

Table 3.9: Mean scores (in percents) and standard deviations σ (in percentage points) of solutions returned by exact methods in constrained optimization (multicriteria maximization of $[u_k]$). *small* data set. Each row is an average over 200 instances.

	min-max				*min-sum*			
	sum		max		sum		max	
$n_{\cdot,k}$	mean	σ	mean	σ	mean	σ	mean	σ
5	9.1	8.1	7.4	9.1	8.1	7.4	6.8	6.5
8	10.0	7.6	7.6	8.1	9.1	7.2	7.6	6.4
10	11.7	7.0	9.5	8.2	10.9	6.7	8.9	5.8
11	10.8	6.3	8.4	7.6	9.9	6.0	8.3	5.4
12	11.8	7.4	9.6	8.7	11.0	7.1	9.3	6.5
average	10.7	7.4	8.5	8.4	9.8	7.0	8.2	6.2

Table 3.10: Mean scores (in percents) and standard deviations σ (in percentage points) of solutions returned by exact methods in unconstrained optimization (multicriteria minimization of $[C_k]$). *small-3* data set. Each row is an average over 50 instances.

	min-max				*min-sum*			
	sum		max		sum		max	
$n_{\cdot,k}$	mean	σ	mean	σ	mean	σ	mean	σ
4	3.1	9.7	22.4	11.1	9.5	6.5	9.0	14.6
5	2.2	9.1	25.5	9.1	9.6	6.2	8.8	14.1
6	3.0	7.5	24.4	9.2	9.8	6.4	8.6	13.6
average	2.8	8.8	24.1	9.9	9.6	6.3	8.8	14.0

strategy (resulting in negative score). Consequently, the standard deviation of *max min-sum* in unconstrained optimization is much higher (13.5%) than in constrained optimization (8.4%).

Thirdly, in unconstrained optimization, choosing the *min-max* solution considerably worsens the social criterion, as the gain in $\sum C_k$ is reduced by the half. At the same time, however, the worst C_k is considerably improved (by more than 20%). In constrained optimization, *min-max* solution is only slightly better than *min-sum* regarding the maximum C_k. However, *min-max* is the solution that maximizes the minimum gain over **MJF**, which not necessary corresponds to the minimization of the maximum C_k.

Finally, the impact of the size of the instance $n_{\cdot,k}$ is small, although there is a slight improvement of the results with the instance size, e.g. for $n_{\cdot,k} = 5$, *sum min-sum*= 11%, whereas for $n_{\cdot,k} = 12$, *sum min-sum*= 13%.

With the increase in the number of organizations, scores get better, especially in unconstrained optimization (Table 3.10). This is probably caused by the increase in the number of solutions, out of which it is easier to chose the better one. Additionally, we can see similar phenomena as observed in two organization case.

Table 3.11: Mean scores (in percents) and standard deviations σ (in percentage points) of solutions returned by exact methods in constrained optimization (multicriteria maximization of $[u_k]$). *small-3* data set. Each row is an average over 50 instances.

	min-max				*min-sum*			
	sum		max		sum		max	
$n_{\cdot,k}$	mean	σ	mean	σ	mean	σ	mean	σ
4	6.4	5.8	4.6	4.8	7.5	7.1	5.4	7.9
5	6.6	5.7	5.1	5.1	7.7	6.6	6.0	7.3
6	7.0	5.6	4.9	4.1	8.2	6.3	5.5	5.3
average	6.6	5.7	4.9	4.6	7.8	6.7	5.6	6.9

3.7.3 Greedy Heuristics

To measure the performance of EW and AEW, we experimented on the same random data set as in the previous section. In the first place, we measure the run-time of the algorithms. Then, we measure the quality of the results returned by both methods: the number of equitable solutions returned; and the quality of two "best" solutions from each result set.

Run-time analysis

We started with comparing the runtime of the DYN and that of greedy heuristics. Table 3.12 presents the aggregated results. Note that the increase in the DYN runtime (comparing with Table 3.1) is caused by an additional operation (i.e. removing equitably-dominated solutions) on DYN results. We do not present the average run-time in function of $\max p^i_{k,l}$, because the results of EW and AEW did not differ from the average.

We can see that, firstly, both greedy algorithms produce results much faster than DYN (the differences between DYN and EW; and DYN and AEW are statistically significant). Secondly, although on average AEW is slower than EW, the difference is small. Using paired, two-tailed t-test, we obtained $p = 2.75E - 5$ for the average, but only $p = 0.025$ for $n_{\cdot,k} = 12$.

Table 3.12: Comparison of the average run-time of DYN and equitable walk algorithms (EW, AEW) over randomized instances. Each row is an average from 200 random instances. *small* data set.

	run time [s]		
$n_{\cdot,k}$	DYN	EW	AEW
5	0.01	0.00	0.00
8	0.16	0.01	0.01
10	0.88	0.03	0.03
11	1.84	0.04	0.06
12	4.04	0.06	0.08
average	1.39	0.03	0.04

In the second series of tests, we measured the runtime of both greedy heuristics on larger instances (Table 3.13).

We see that the run-time of heuristics rises sharply with the increased number of organizations up to $N = 6$. Afterward, although the average run-time is increasing, the differences are no longer statistically significant. With two-tailed, type 3 T-Test on EW average run-time, we obtained $p = 0.14$ for $N = 6$ and $N = 7$, and $p = 0.11$ for $N = 7$ and $N = 8$. The increase of the run-time is sharp, but not necessary exponential. The number of organization is similarly correlated with the run-time (EW: $\sigma = 0.73$, AEW: $\sigma = 0.70$) as with the logarithm of the run-time (EW: $\sigma = 0.68$, AEW: $\sigma = 0.63$).

When increasing the number of jobs on each processor $n_{\cdot,k}$, we also note a steady increase in the average run-time of both algorithms. Here, however, all the differences are statistically significant. The increase is rather linear than exponential. $n_{\cdot,k}$ correlates with run-times better (EW: $\sigma = 0.52$, AEW: $\sigma = 0.54$) than with the logarithms of run-times (EW: $\sigma = 0.29$, AEW: $\sigma = 0.39$).

Table 3.13: Comparison of the average run-time (in seconds) of equitable walk algorithms over randomized instances from *large* data set. Each cell is an average from 50 random instances

$n_{\cdot,k} \rightarrow$	average		10		15		20		25	
N $\downarrow$	EW	AEW	EW	AEW	EW	AEW	EW	AEW	EW	AEW
2	11.69	19.99	0.06	0.05	1.23	1.24	20.43	26.29	25.05	52.40
3	89.43	80.12	8.20	2.79	58.29	58.62	124.41	107.24	166.84	151.86
4	149.24	125.44	52.04	30.14	133.41	103.09	184.80	168.67	226.72	199.87
5	189.80	159.56	93.89	56.02	161.00	142.49	216.32	199.68	288.02	240.04
6	285.72	238.35	142.45	104.74	237.98	204.19	332.12	282.27	430.32	362.21
7	303.00	261.42	163.87	139.96	249.78	223.58	344.04	296.76	454.31	385.39
8	324.42	275.97	174.12	145.44	269.91	243.39	374.69	331.06	478.96	383.99
average	193.33	165.84	90.66	68.45	158.80	139.51	228.12	201.71	295.75	253.68

Table 3.14: Quality of the set of solutions returned by EW and AEW compared to the exact solutions. *small* data set. First two columns present the percentage of solutions returned by EW and AEW that were equitably-dominated by one of the DYN solutions. Last two columns show the percentage of equitably-optimal solutions found by EW and AEW. Each row is an average from 200 random instances.

	% of sols eq dominated		% of sols found	
$n_{.,k}$	EW	AEW	EW	AEW
5	2.27	1.50	89.38	94.49
8	9.99	5.46	74.51	84.72
10	17.78	8.95	61.59	77.52
11	18.52	10.72	60.47	75.83
12	22.20	13.61	54.57	70.24
average	14.15	8.05	68.10	80.56

Reconstruction of the Equitably-Optimal Set

In the second series of tests, we measured the quality of the results returned by both methods. Firstly, we compared the number of equitable solutions returned by DYN and both greedy methods. The more solutions returned, the broader the choice of the grid community.

Table 3.14 compares the solutions produced by EW and AEW to that produced by the DYN algorithm. Firstly, we can see that most of the solutions returned by both algorithms are indeed equitable, although the percentage of non-equitable solutions grows with the size of the instance.

However, the algorithms are not able to reconstruct the whole set of equitably-optimal solutions. The number of solutions omitted by greedy heuristics increased with the size of instance. Figure 3.10 (for AEW) and Figure 3.11 (for EW) support this remarks. These figures aggregate the data on the level of the number of equitably-optimal solutions. If a few instances have a particular number of equitably-optimal solutions, the scores of algorithms are averaged over these instances. We can see that the both algorithms work fairly well if there are a few equitably-optimal solutions.

Table 3.15: Quality of the set of solutions returned by EW and AEW compared to the exact solutions. *small-3* data set. First two columns present the percentage of solutions returned by EW and AEW that were equitably-dominated by one of the DYN solutions. Last two columns show the percentage of equitably-optimal solutions found by EW and AEW. Each row is an average from 50 random instances.

	% of sols eq dominated		% of sols found	
$n_{.,k}$	EW	AEW	EW	AEW
4	4.57	5.87	70.92	77.45
5	12.57	14.16	53.13	68.66
6	15.64	9.08	39.41	61.11
average	10.92	9.70	54.49	69.07

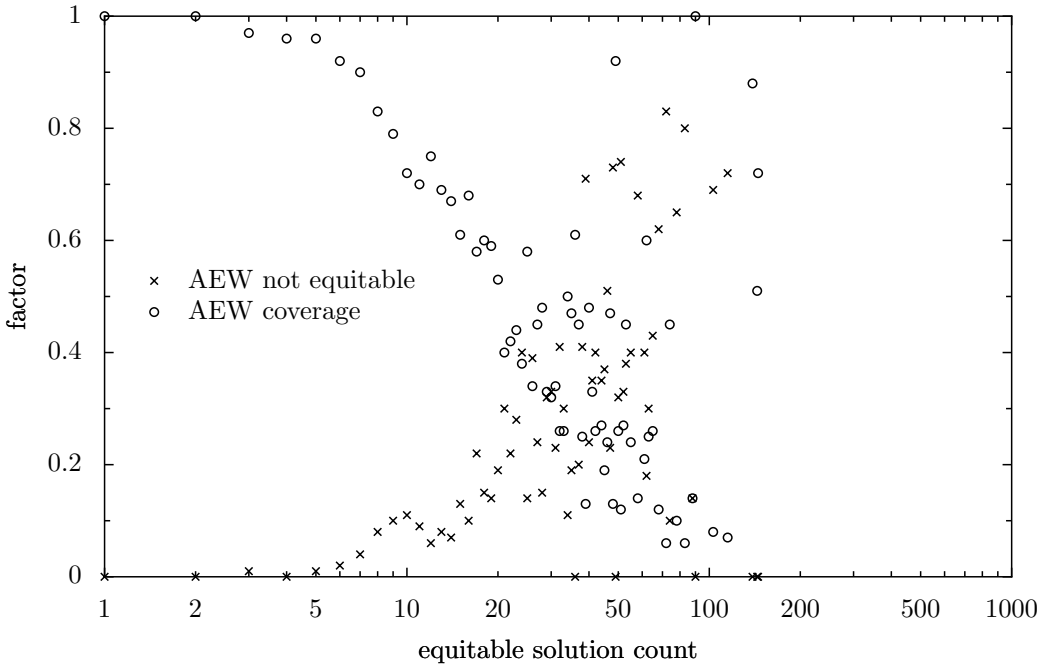


Figure 3.10: The percentage of solutions returned by AEW that were equitably-dominated by one of the exact solutions and the percentage of all equitably-optimal solutions plotted against the total number of equitable solutions (data is averaged over all instances with the same number of equitably-optimal solutions). *small* data set.

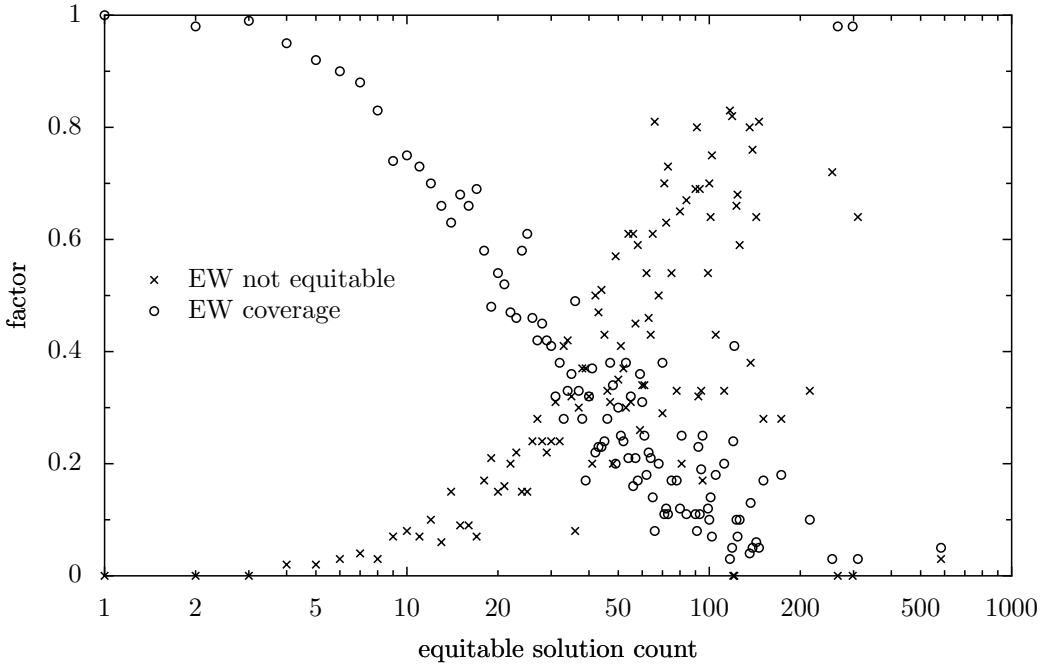


Figure 3.11: The percentage of solutions returned by EW that were equitably-dominated by one of the exact solutions and the percentage of all equitably-optimal solutions plotted against the total number of equitable solutions (data is averaged over all instances with the same number of equitably-optimal solutions). *small* data set.

When the number of equitably-optimal solutions grows, scores of the algorithms quickly worsen. This trend is no longer clearly visible for larger sets of equitably-optimal solutions, as there are only few such instances.

The same trends can be seen in three organizational grids (Table 3.15). The difference between the percentage of equitably optimal solutions returned for $n_{\cdot,k} = 5$, $N = 2$ (89%) and $N = 3$ (53%) is probably caused by the increase in the number of solutions: for $N = 2$ there are, on average, 4.8 equitably optimal solutions (Table 3.5), whereas for $N = 3$ there are 18.3 solutions (Table 3.7). For $n_{\cdot,k} = 10$, $N = 2$, the number of equitably-optimal solutions is similar, but the number of equitably optimal solutions found is slightly higher.

Table 3.16: Mean Δ-scores and standard deviations σ (in percentage points) showing the difference between scores achieved by exact method and EW algorithm in unconstrained optimization (multicriteria minimization of $[C_k]$). *small* data set. Each row is an average over 200 instances.

	Δ *min-sum*				Δ *min-max*			
	sum		max		sum		max	
$n_{\cdot,k}$	mean	σ	mean	σ	mean	σ	mean	σ
5	0.00	0.00	0.00	0.00	-0.08	0.70	0.11	0.49
8	0.00	0.00	0.00	0.00	0.01	0.31	0.07	0.24
10	0.00	0.00	0.00	0.00	0.04	0.18	0.04	0.13
11	0.00	0.00	0.00	0.00	0.04	0.28	0.06	0.21
12	0.00	0.00	0.00	0.00	0.04	0.19	0.04	0.11
average	0.00	0.00	0.00	0.00	0.01	0.39	0.06	0.27

Quality of individual solutions

The last result indicates that both greedy algorithms produce fewer solutions. Yet, in the end, only one solution will be used to schedule the jobs. In this experiment we measured how this decrease in the number of produced solutions translates into the loss of the efficiency of the whole system.

From the set of equitably-optimal solutions produced by each greedy algorithm, we chose two solutions, similarly to the analysis in 3.7.2: the *min-sum* solution and the *min-max* solution. Then we compared scores of these solutions with that of corresponding solutions taken from the whole set of equitably-optimal solutions produced by DYN algorithm. For each measure, we subtracted the score of the greedy method from that of the exact method. Consequently, for a measure, a Δ-*score* of e.g. 10 percentage points means that the score of a greedy heuristics was worse by 10 percentage points from the score of the exact method. Aggregated results for $N = 2$ are presented in Table 3.16 for EW and in Table 3.17 for AEW; for $N = 3$ in Table 3.18 for EW and in Table 3.19 for AEW. Note that EW returns the same *min-sum* solution as DYN (i.e. **SPT**), so the corresponding Δ-*score* is always 0.

Table 3.17: Mean Δ-scores and standard deviations σ (in percentage points) showing the difference between scores achieved by exact method and AEW algorithm in constrained optimization (multicriteria maximization of $[u_k]$). *small* data set. Each row is an average over 200 instances.

	Δ *min-sum*				Δ *min-max*			
	sum		max		sum		max	
$n_{.,k}$	mean	σ	mean	σ	mean	σ	mean	σ
5	0.06	0.40	0.01	1.31	-0.03	0.28	0.00	0.65
8	0.09	0.58	0.07	0.98	0.01	0.21	0.03	0.36
10	0.03	0.12	0.03	0.34	0.02	0.09	0.02	0.20
11	0.06	0.26	0.01	0.32	0.02	0.11	0.01	0.20
12	0.04	0.16	0.01	0.25	0.03	0.12	0.04	0.14
average	0.06	0.35	0.03	0.77	0.01	0.18	0.02	0.36

Table 3.18: Mean Δ-scores and standard deviations σ (in percentage points) showing the difference between scores achieved by exact method and EW algorithm in unconstrained optimization (multicriteria minimization of $[C_k]$). *small-3* data set. Each row is an average over 50 instances.

	Δ *min-sum*				Δ *min-max*			
	sum		max		sum		max	
$n_{.,k}$	mean	σ	mean	σ	mean	σ	mean	σ
4	0.00	0.00	0.00	0.00	-1.48	2.67	0.87	1.70
5	0.00	0.00	0.00	0.00	-1.13	2.46	0.69	1.38
6	0.00	0.00	0.00	0.00	-0.99	2.52	0.80	1.90
average	0.00	0.00	0.00	0.00	-1.20	2.54	0.78	1.66

Table 3.19: Mean Δ-scores and standard deviations σ (in percentage points) showing the difference between scores achieved by exact method and AEW algorithm in constrained optimization (multicriteria maximization of $[u_k]$). *small-3* data set. Each row is an average over 50 instances.

	Δ *min-sum*				Δ *min-max*			
	sum		max		sum		max	
$n_{\cdot,k}$	mean	σ	mean	σ	mean	σ	mean	σ
4	0.22	0.80	0.43	1.57	-0.05	1.37	0.28	1.56
5	0.29	0.67	-0.08	2.44	-0.15	0.95	0.05	1.02
6	0.28	1.00	0.29	1.79	0.17	0.91	0.19	1.34
average	0.26	0.83	0.21	1.97	-0.01	1.10	0.17	1.32

The greedy methods provide solutions that are not far from the optimal ones. For $N = 2$, all average scores differ less than 0.1 percentage points from the optimal ones. However, the difference is statistically significant (two-tailed type one t-test, $p < 0.001$).

The standard deviations are significant compared to the average values. This is caused by large dispersion of results. For instance, for $N = 2$, $n_{\cdot,k} = 5$, and Δ *min-max max* only 13 instances out of 200 had score different than 0, but the highest difference observed was 4.5 percentage points.

We also saw a number of instances in which Δ *min-max sum* or Δ *min-sum max* were negative, i.e. greedy heuristics returned "better" results than the exact methods. In this instances, however, greedy heuristics were not able to return all equitably-optimal solutions. Consequently, an exact solution that has the lowest *min-max max* can have higher *min-max sum* than the solution that has the lowest *min-max max* among the ones found by a greedy heuristics.

Further investigating the difference in the quality of solutions, we saw that, generally, the less solutions discovered by a greedy algorithm, the larger is the difference. For instance, the correlation coefficient between the percentage of equitably-optimal solutions found by AEW and the Δ *min-sum sum*

Table 3.20: Mean scores and standard deviations σ (in percents) showing the quality of solutions produced by EW algorithm in unconstrained optimization (multicriteria minimization of $[C_k]$). *large* data set. Each row is an average over 200 instances.

	min-sum				*min-max*			
	sum		max		sum		max	
N	mean	σ	mean	σ	mean	σ	mean	σ
2	12.51	5.05	11.71	11.22	8.47	7.26	21.68	9.54
3	12.18	4.36	15.74	10.82	8.89	6.44	27.84	8.63
4	10.96	3.64	15.65	9.84	8.74	4.61	28.23	8.07
5	10.16	3.28	14.42	9.58	8.37	3.79	28.05	7.89
6	8.79	2.72	14.69	8.54	7.16	3.07	29.06	7.66
7	8.14	2.57	14.76	8.05	6.97	2.92	27.54	7.26
8	7.33	2.24	14.19	7.02	6.33	2.53	26.75	6.93
average	10.01	3.99	14.45	9.46	7.85	4.77	27.02	8.34

score is -0.31. However, the correlation between the number of equitably-optimal solutions and the score is much smaller (0.05), indicating that there is no clear relationship.

With the increased number of organizations ($N = 3$, Tables 3.18 and 3.19), both the observed differences and the standard deviations are increased. However, the differences in the score of the main measure do not exceed 0.3 percentage points for AEW and 0.8 percentage points for EW.

We conclude that, even that the number of solutions returned is smaller, the quality of the solutions returned is similar.

In the second series of tests, we run both greedy algorithms over *large* data set. As the instances were significantly larger, it was not feasible to run exact algorithms. Thus, Table 3.20 and Table 3.21 present only the results achieved by EW and AEW. We do not present quality of both algorithms in function of the number of jobs on each processor $n_{\cdot,k}$. Similarly to last tests, we found the average values similar and all but a few differences statistically insignificant.

Table 3.21: Mean scores and standard deviations σ (in percents) showing the quality of solutions produced by AEW algorithm in constrained optimization (multicriteria maximization of $[u_k]$). *large* data set. Each row is an average over 200 instances.

	min-sum				*min-max*			
	sum		max		sum		max	
N	mean	σ	mean	σ	mean	σ	mean	σ
2	11.71	5.94	9.72	6.85	10.94	5.83	9.24	5.40
3	11.81	4.63	10.41	8.15	10.91	4.40	8.37	4.30
4	10.67	3.75	8.63	6.54	10.03	3.53	7.28	3.38
5	10.00	3.33	7.60	6.07	9.51	3.20	7.17	3.93
6	8.62	2.83	6.81	5.24	8.24	2.75	6.04	3.13
7	8.07	2.60	6.52	5.26	7.75	2.49	6.02	3.36
8	7.17	2.36	5.34	4.44	6.89	2.28	5.06	2.52
average	9.72	4.17	7.86	6.40	9.18	3.96	7.03	4.04

We see that the quality of proposed solutions falls with the number of organizations. However, it is not because of the algorithms that are not scalable with the size of the instance. *min-sum sum* solution returned by EW is **SPT** solution that is optimal for this criteria. Yet, we see that even the optimal solution is better than **MJF** by, on the average, 7% when $N = 8$ (compared to almost 12% when $N = 2$). Similar trends can be seen in all the other scores.

Note also that in some instances AEW was not able to find at least one acceptable result, i.e. $\forall_k u_k \geq 0$ (see Table 3.22). Here, as we do not have the results of exact algorithm, we do not know whether such a solution does not exist or, simply, AEW is unable to find it. The worst result, however, is achieved for the smallest instance considered (with $N = 2$ organizations and $n_{\cdot,l} = 10$). In the last series of tests, for the same N and $n_{\cdot,l}$, AEW always found an acceptable solution, or such a solution did not exist.

Table 3.22: Number of instances in which AEW was able to find at least one acceptable result (i.e. $\forall k u_k \geq 0$).*large* data set.

	$n_{\cdot,l}$				
orgCount	10	15	20	25	sum
2	44	45	48	47	184
3	48	49	50	50	197
4	50	49	50	50	199
5	50	50	50	50	200
6	50	50	50	49	199
7	50	50	50	50	200
8	49	50	50	49	198
sum	341	343	348	345	1377

3.8 Equitable Criteria

Using the above optimization scheme, organizations can manipulate the results of the process only by dividing long jobs into pieces. Let's assume that job $J^i_{k,l}$'s size is $p^i_{k,l}$ and the job is executed at the beginning of the queue, the job's contribution to the C_k is $p^i_{k,l}$. If job's owner O_k, instead of submitting $J^i_{k,l}$, divides the job into $p^i_{k,l}$ jobs of size 1 and then submits them as independent jobs, the contribution will increase to $\frac{1}{2}p^i_{k,l}(p^i_{k,l}-1)$. For the grid scheduler, this would be a signal that the performance of O_k decreases. Consequently, a equitable scheduler, which compares C_k directly with completion times of different organization, would favor O_k jobs.

A simple solution is to compare instead the *weighted* sum of completion times $\sum_{i,l} w_k C^i_{k,l}$. Let us assume that, for each organization O_k, the weight of each job is equal to the inverse of the total number of jobs produced by the organization $w_k = \frac{1}{n_{k,\cdot}}$. As a result, in the above example, when $j^i_{k,l}$ is not divided, it's contribution to the weighted completion time is the same. When the job is divided, it's contribution becomes $\frac{1}{2p^i_{k,l}}p^i_{k,l}(p^i_{k,l}-1) = \frac{1}{2}(p^i_{k,l}-1)$, which is of the same order as the contribution in the first case.

The shift of the optimization goal from C_k to $w_k C_k$ does not change much the EW. The first schedule considered must order jobs according to weighted SPT rule, i.e. according to ascending $\frac{p^i_{k,l}}{w_k}$. Then the algorithm proceeds as described above.

3.9 Summary

In this chapter we addressed the problem of resource management in highly heterogeneous computational grids. Our model emphasizes the miscellaneous nature of both the resources (by considering dedicated resources) and the users (by introducing the notion of an organization). We analyzed the model using three approaches: equitable optimization (Section 3.2), when the grid scheduler has a complete control over all the local schedulers; game theory (Section 3.5), when local schedulers can alter their schedule; and constrained equitable optimization (Section 3.6), when local schedulers cannot alter their schedule, but the owner of the resource can leave the grid. The optimization problems considered are difficult: the problem of finding one Pareto-optimal solution is NP-hard (Section 3.2.2) and the number of all Pareto-optimal solutions can be exponential (Section 3.2.1). Using game theoretic approach, we demonstrated (Proposition 3.9) that the complete decentralization may lead to a significant performance loss of the system. We proposed simple, greedy heuristics to find solutions for optimization problems in the unconstrained and in the constrained case (Section 3.3.3). Through an extensive experimental validation (Section 3.7) we showed these heuristics deliver results that are not far from the optimal ones and work a few orders of magnitude faster at the same time. Our experiments also show that the constraints on optimization do not worsen the results significantly. Therefore, strong control performed by the grid community is indispensable to achieve good performance.

Chapter 4

Divisible Load Balancing

One of the first applications of large-scale distributed computing was Seti@home, that eventually evolved into a platform called BOINC [Anderson, 2004]. Seti@home gathered 95 TFLOPS of sustained computing power through computers of hundreds of thousands of volunteers connected over the Internet [Anderson and Fedak, 2006]. Such a large scale parallelism is possible because BOINC applications are *embarrassingly parallel*. Each BOINC application would take thousands of CPU-years on a single machine. However, embarrassingly parallel applications can be divided into a large number of relatively *small* and *independent* fragments, that are then computed in parallel on a large number of volunteers' computers. Fortunately, there are many problems which are embarrassingly parallel, mostly through data parallelism or parameter sweeps. In data parallelism, a large volume of data to analyze can be divided into independent pieces, as it was the case of Seti@home. In parameter-sweep problem, a relatively simple computer model must be run with many parameter settings. In scheduling theory, embarrassingly parallel problems are modeled as *divisible load*.

In this chapter, we consider the grid as a tool to compute divisible load jobs. Following our general model of multi-organizational grid, we assume that each job was produced by an organization. Yet, it can be computed in parallel on other organizations' resources, if these organizations agree to

accept the job.

We study what decision organizations should make in the presence of uncertainty. We consider an *on-line* model, in which jobs are produced during the lifetime of the system and are unknown before having been produced.

As long as there are no local jobs, computing a foreign job does not cause any additional cost. This is one of the reasons of BOINC popularity. However, if a new local job is produced, the foreign job is blocking the needed resources and thus delaying the newly-produced local job. In such situation, BOINC, running in the *best-effort* mode, suspends the computation of foreign job. Consequently, the owner of the foreign job cannot have any fixed guarantee on its completion time.

In contrast to this approach, we assume that the system must guarantee the latest finish time of every submitted job, which is announced to the user while the job is submitted. We claim that such guarantee gives better quality of service than best-effort execution commonly used in distributed divisible load computing.

This chapter is organized as follows. After the formal definition of the model in Section 4.1, we compute the expected delay in the computation of next local job in Section 4.2. This function allows us to analyze a simplified case with optimization approach (Section 4.2). We study a classic algorithm, averaging load balancing, that distributes the load so that every resource finishes computation at the same moment. We show that the solution proposed is not only globally optimal, but also the most fair (Section 4.3), but it always deteriorates the performance of the least loaded resource. In Section 4.4 we use a game-theoretic approach to demonstrate that the fair solution will never be realizable. Then, in Section 4.5 we show that, in the case of similarly loaded resources, if we oblige the organizations to collaborate during longer time periods, averaging load balancing becomes the dominant strategy for each organization. Yet, when the load is uneven among the resources, using simple, averaging load balancing algorithm becomes unfeasible for the less-loaded resources. Moreover, in Section 4.6 we show that if organizations

make decisions depending on their current queue length, load balancing will never be performed. In Section 4.7 we solve this problem by introducing another load distribution algorithm, that employs the general ideas of fair, constrained optimization.

4.1 Grid Model

In this chapter, we make the following assumptions about the model:

- the system works *on-line*;
- the scheduler is *clairvoyant*;
- each resource has only one processor;
- processors are identical;
- jobs follow the *divisible load* model;
- resources are *time shared* between jobs;
- *preemption* is not allowed;
- as a performance measure, each organization O_k calculates the *sum of flow times* F_k of locally produced jobs $\mathcal{J}_k$.

At a given moment, let the *local load* L_k stand for the time moment when, on resource M_k, the computation of the last currently known local job ends.

For the sake of the theoretical analysis, unless otherwise stated, we assume that the jobs $\mathcal{J}_k$ are produced by a Poisson process with a known mean time between arrivals λ_k. The sizes p_k^i of the jobs follow exponential distribution with known μ_k. Such assumptions are commonly used in queuing theory due to the fact that the probability of arrival at each time moment is constant. Similarly, job, during its execution, has constant probability to be finished in each time moment. Note that these parameters are not needed by the algorithms presented in the chapter.

Local resource management systems queue incoming jobs on First Come First Served (FCFS) basis.

Although we discuss several load balancing (LB) algorithms, they all share a common property of being able to guarantee the latest finish time of every job submitted to the system, which is announced to the user in the moment of job submission. We consider that such guarantee, as opposed to best-effort execution, provides better usage experience for the users (better Quality of Service) and prevents job starvation.

We can formulate the problem of selfish LB both from the grid's (infrastructure's) and from the organization's point of view. The grid should provide a LB algorithm which satisfies as many organizations as possible. Each organization must decide whether to participate in the LB process, or not, given the algorithm and the observed behavior of other organizations.

4.2 Cost of Computation

Most of the papers dealing with Grid Economy assume that the cost of a resource is given in advance. But what is the real cost of running a job during a certain amount of time? One possible approach might consider that a job should participate in resource's running cost, such as the cost of the electricity consumed, or the cost of the staff. However, if no jobs are executed, these costs remain constant. Supercomputers are not normally switched off, nor the staff is dismissed. On the other hand, in space-sharing systems, a small (and therefore cheap) job can block resources required by a large job, which would pay more.

We can alternatively suppose that the cost of accepting foreign load Φ_k is determined by the delay in the computation of the "next" local job J_k^n, unknown at the moment of decision. Such a measure expresses the "irritation" of a local user who experiences his/her job being delayed because of a foreign load being executed (see Figure 4.1). Note that if Φ_k is accepted, the arrival of J_k^n cannot interrupt Φ_k's scheduled or ongoing execution, because such

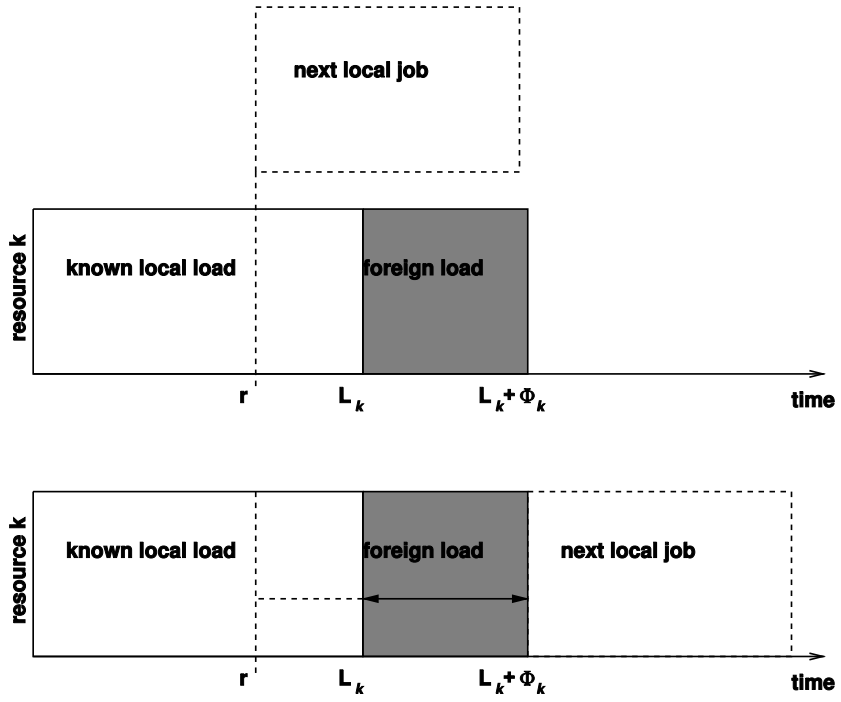


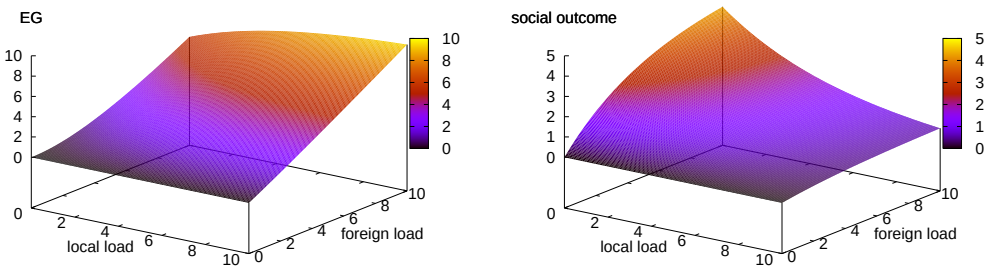
Figure 4.1: A dilemma (top figure) faced by organization O_k accepting a foreign load. Accepting it (bottom figure) can delay the execution of the next local job by Φ_k, if the local job arrives before $L_k + \Phi_k$.

interruption could break the announced finish times for jobs which fragments belong to Φ_k. It would essentially turn our system into best-effort one.

More formally, we can formulate a function $g(r, L_k, \Phi_k)$ which expresses the delay in the computation of J_k^n in function of known local load L_k, size of the foreign load Φ_k, and J_k^n's arrival time $r = r_k^n > 0$. We assume that the decision whether to accept a foreign load Φ_k is made at time $t = 0$. We also do not consider the impact of the following local jobs nor LB which might occur in the future. Using this notation,

$$g(r, L_k, \Phi_k) = \begin{cases} \Phi_k & \text{if } r \in (0, L_k), \\ -r + L_k + \Phi_k & \text{if } r \in (L_k, L_k + \Phi_k), \\ 0 & \text{if } r \in (L_k + \Phi_k, +\infty). \end{cases}$$

If $J_{i,n}$ arrives before the foreign load is started, it is delayed by Φ_k, i.e. the



a b

Figure 4.2: The expected delay in the execution of the next local job $EG(L_k, \Phi_k)$ plotted against local load L_k and the accepted foreign load Φ_k (a) and the social outcome $\Phi_k - EG(L_k, \Phi_k)$, i.e. the difference between the sender's gain Φ_k and the expected delay (b). All variables are measured in time units. $\lambda = 0.2$.

size of the foreign load (first row). If $J_{i,n}$ arrives after the foreign load is finished, it is not delayed (last row). Finally, during computation of Φ_k, the delay linearly decreases with r (second row).

Knowing that r is a random variable with distribution $f(r)$, we can compute the expected value of $g(r, L_k, \Phi_k)$ as

$$\begin{aligned} EG(L_k, \Phi_k) &= \int_{-\infty}^{+\infty} g(r, L_k, \Phi_k) f(r) dr = \\ &= \int_{0}^{L_k} \Phi_k f(r) dr + \int_{L_k}^{L_k+\Phi_k} (-r + L_k + \Phi_k) f(r) dr. \end{aligned} \quad (4.1)$$

If the local jobs arrive according to a Poisson process with known λ_k, then the delays between the jobs follow exponential distribution, thus $f(r) = \lambda_k \cdot \exp(-\lambda_k r)$ for $r > 0$. After computing the integrals, we get the following formula:

$$EG(L_k, \Phi_k) = \Phi_k + \frac{e^{-\lambda_k L_k}}{\lambda_k}\Big(e^{-\lambda_k \Phi_k} - 1\Big).$$

$EG(L_k, \Phi_k)$ is plotted on Fig. 4.2.a. It should be noted that, firstly, the expected delay $EG(L_k, \Phi_k)$ is not linear with foreign load Φ_k. This

suggests that the approach used in [SunGrid, 2005], that calculates the cost of execution of a job on a resource as $cost = coefficient \cdot jobsize$, may miss some important phenomena. Secondly, $\Phi_k - EG\,(L_k, \Phi_k) > 0$ for all possible (positive) values of Φ_k and L_k.

4.3 Averaging Load Balancing: A Fair Optimization Perspective

The cost function EG shows that the gain the sender gets from sending a part of its load is always greater than the expected lateness of future local jobs of the receiver. In this section, we will use this fact to characterize the fairness of the load resulting from a simple, averaging load balancing algorithm.

In two-organization grid, setting the load to be sent to the half of the difference in loads results in equitably-optimal solution. Note that an averaging load balancing algorithm works in that way. Note also that we do not consider here more sophisticated algorithms that e.g. balance each job separately. Therefore, we may treat 'the load' on each resource as composed of a single job. To simplify our analysis, we also consider that both resources optimize their makespan (which, with a single job on each resource, is identical with the sum of completion time C_k).

Proposition 4.1. *Let us assume that $L_2 > L_1$. Sending half of the difference in loads to the least-loaded resource ($\Phi_1^* = \frac{1}{2}(L_2 - L_1)$) equitably-optimizes the vector of sum of completion times $[C_1, C_2]$. Moreover, the resulting load distribution equitably-dominates all other distributions.*

Proof: Solution $[C_1, C_2]$ is equitably-optimal, iff it is a Pareto-optimal solution concerning the minimum sum of criteria $\min C_1 + C_2$ and the maximum criterion $\min \max(C_1, C_2)$. In this proof we show that, firstly, Φ_1^* is optimal for the maximum criterion, and then that Φ_1^* optimizes also the sum of criteria.

A fraction Φ_1 of M_2's load sent to M_1 results in completion times $[C_1, C_2] = [L_1 + EG(L_1, \Phi_1), \max(L_1 + \Phi_1, L_2 - \Phi_1)]$.

C_2 is minimized when $L_1 + \Phi_1 = L_2 - \Phi_1 \Leftrightarrow \Phi_1^* = \frac{1}{2}(L_2 - L_1)$.

$C_1 \leq C_2$, as $C_1 = L_1 + EG(\Phi_1) \leq L_1 + \Phi_1 \leq L_2 - \Phi_1$ for $\Phi_1 \in [0, \Phi_1^*]$. Consequently, Φ_1^* minimizes the maximum criterion.

The sum of criteria $(C_1 + C_2)(\Phi_1)$ is equal to:

$$(C_1+C_2)(\Phi_1) = \begin{cases} L_1 + EG(\Phi_1) + L_2 - \Phi_1 & \text{if } L_1 + \Phi_1 \leq L_2 - \Phi_1 \Leftrightarrow \Phi_1 \leq \Phi_1^* \\ 2L_1 + EG(\Phi_1) + \Phi_1 & \text{if } \Phi_1 > \Phi_1^*. \end{cases}$$

When minimizing the function in the first case ($\Phi_1 \leq \Phi_1^*$), we may omit constants, so that the function becomes:

$$L_1 + EG(\Phi_1) + L_2 - \Phi_1 \rightarrow EG(\Phi_1) - \Phi_1 = \frac{e^{-\lambda_1 L_1}}{\lambda_1}\Big(e^{-\lambda_1 \Phi_1} - 1\Big) \rightarrow e^{-\lambda_1 \Phi_1},$$

a monotonically decreasing function having the minimum value for the maximum possible Φ_1, i.e. Φ_1^*.

In the second case ($\Phi_1 > \Phi_1^*$) we obtain a monotonically increasing function of Φ_1 that is minimized for $\Phi_1 \rightarrow \Phi_1^*$.

Note also that $C_1 + C_2$ is continuous. For all Φ_1 except Φ_1^* $C_1 + C_2$ is a composition of continuous functions. Upper-limit in $\Phi_1 = \Phi_1^*$ is equal to the value of function $(C_1 + C_2)(\Phi_1^*)$:

$$\begin{aligned} \lim_{\Phi_1 \rightarrow \Phi_1^* +} (C_1 + C_2)(\Phi_1) &= 2L_1 + EG(\Phi_1^*) + \Phi_1^* && (4.2) \\ &= L_1 + L_2 - 2\Phi_1^* + EG(\Phi_1^*) + \Phi_1^* && (4.3) \\ &= (C_1 + C_2)(\Phi_1^*). && (4.4) \end{aligned}$$

Thus, the global minimum of $(C_1 + C_2)(\Phi_1)$ is in $\Phi_1 = \Phi_1^*$.

Consequently, as $\Phi_1 = \Phi_1^*$ optimizes both the sum of criteria and the worst criterion, Φ_1^* is the only equitably-efficient solution of the problem. □

4.4 A Two-Player Sender-Receiver Game

We present now how the foregoing cost function $EG(L_k, \Phi_k)$ can be used to analyze the expected result of LB performed in a grid formed by two organizations.

Definition 4.1. *A simple load balancing game is a game in a normal form, where:*

- $\mathcal{P} = \{O_H, O_L\}$, *where* O_H, *the* sender *has heavier load on its local resource* M_H. ($L_H \geq L_L$). O_L *will be denoted as the* receiver;
- $\sigma_H = \{$*send*, *compute locally*$\}$, $\sigma_L = \{$*accept*, *reject*$\}$. O_H *send*s *part* Φ_L *of its load to the other resource or* compute*s everything locally.* O_L *either* accept*s the incoming foreign load, or* reject*s it.*
- *The payoff of each player is defined as the reduction of completion time of player's job. Each player maximizes the payoff. The table of payoffs is depicted in Table 4.1.a.*

In such game, LB (send and accept) is globally-optimal, thus it the best solution from the infrastructure's point of view.

Proposition 4.2. *The pair of strategies corresponding to LB (*send *and* accept*) is the social optimum (maximizes the sum of payoffs).*

Proof: If the sender sends and the receiver accepts the load, the receiver expects that its local jobs will be delayed by $EG(L_L, \Phi_L)$. As it will affect O_L's criterion F_L, O_L's payoff will be always negative. However, sender's jobs are finished faster by at least Φ_L (assuming that the sender's load after sending ends at time $L_L + \Phi_L$ or later, otherwise Φ_L should be reduced). Therefore, the sender's payoff is at least Φ_L. As $\Phi_k - EG(L_i, \Phi_i) > 0$, the sum of payoffs of both players, or the *social outcome* (plotted on Fig. 4.2.b) is always positive. □

Table 4.1: The structure of payoffs in a basic game (a) and when the organizations cooperate longer (b). The row player's payoff is in the top left corner, the column player's in the bottom right corner.

a

	receive	reject
send	Φ_k, $-EG(L_k, \Phi_k)$	0, 0
compute locally	0, 0	0, 0

b

	cooperate	compute locally
cooperate	$\frac{1}{2}(\Phi_k - EG(L_k, \Phi_k))$, $\frac{1}{2}(\Phi_k - EG(L_k, \Phi_k))$	0, 0
compute locally	0, 0	0, 0

The social outcome reaches the maximum for small local load L_k and large foreign load Φ_k as $\lim_{\Phi_k \to \infty} \Phi_k - EG(0, \Phi_k) = \frac{1}{\lambda_k}$. Social outcome diminishes with the increase of L_k, because the risk of delaying the next local job becomes greater, yet the gain of the sender is the same.

Corollary 4.1. *LB (send and accept) will never occur in simple load balancing game.*

Proof: For the receiver, the action accept is dominated by reject, as it always leads to a lower payoff ($-EG(L_L, \Phi_L)$ if the heavier loaded resource sends, 0 otherwise). Thus, the receiver will never play accept. □

However, the probability of being the receiver is similar to that of being the sender, if the resources are similarly loaded and the system forces the organizations to commit to their decisions for some period, instead of letting them cooperate only instantly. Consequently, both organizations should benefit from cooperation. Table 4.1.b informally shows the idea. In such a game, as $\Phi_k - EG(L_k, \Phi_k) > 0$, the action *cooperate* dominates the local computation, and, consequently, LB becomes the only profitable strategy.

The transition from the basic game to the cooperation game can be viewed as a transition from a one-shot game to a repeated game with players committing to their strategies. Actions which are dominated in a simple game become feasible when the game is repeated. There is, however, an important

difference between repeating the simple game and the cooperation game. The simple game is asymmetric, so in order to obtain payoffs from Table 4.1.b, every player must alternate between being the sender and the receiver.

4.5 Averaging Load Balancing Game

The last section shows that if players are forced to cooperate longer, load-balancing should be profitable. Since it is hard to extend the theoretical results to cases with more than two players and more than one next job, we decided to validate the proposed model empirically by constructing a grid simulator with learning participants. In the simulator each resource measures the performance of its local jobs and accordingly adjust its willingness to join the LB coalition.

A *coalition* C is formed by a subset of resources. Resources belonging to the coalition balance their load by a centralized, averaging LB algorithm. Periodically, each organization decides whether to join the coalition for the next time period $T \gg p_{i,j}$. An organization cannot leave, nor enter, the coalition until the next decision moment. This decision is made locally, and it depends only on the performance of local jobs $\mathcal{J}_k$. There are no outside, administrative rules that would enforce an organization to join the coalition. However, once an organization is in the coalition, it must obey the results of the LB algorithm. Organizations outside the coalition compute all the locally produced jobs and no foreign jobs.

If a resource M_k participating in C has been assigned some foreign load Φ_k (composed of fragments of foreign jobs), M_k must execute Φ_k immediately after time L_k, i.e. immediately after finishing the last local job known at the moment of the execution of the algorithm. As both local and foreign jobs are never postponed, we can guarantee that a job J_k^i will start no later than $\max(L_k, L'_k + \Phi_k)$ (where L'_k is the local load known in the moment of executing the last LB procedure before r_k^i). The job can be finished faster if the next LB is performed before the job's scheduled finish time.

Algorithm 4.1: Averaging Load Balancing.

Input: $\mathbf{L} = [L_1, \ldots, L_N]$, vector of current loads of resources (with true values for resources participating in the coalition)

$\overline{L} = \frac{1}{|C|} \sum_{k:O_k \in C} L_k$;

foreach $k : O_k \in C$ **do**

  if $L_k > \overline{L}$ **then**

    $s_k = L_k - \overline{L}$;

    $toSend = s_k$;

    foreach $l : O_l \in C$ **do**

      if $L_l < \overline{L}$ **then**

        $space = \overline{L} - L_l$;

        $\Phi_{l,k} = \min(space, toSend)$;

        `send`$(M_k, M_l, \Phi_{l,k})$;

        $toSend- = transmit$;

The decision taken by each organization whether to join the coalition is modeled by two pure strategies (corresponding to the ones from Table 4.1.b): s_1 – join the coalition; and s_2 – do not join the coalition. In this game, each organization uses mixed strategy σ_k, which specifies probability p_k that the organization O_k will join the coalition (i.e. the probability of using strategy s_1).

4.5.1 Description of Algorithms

Here, we describe the LB algorithm used to balance the load in the coalition and the algorithm that the organization execute to adjust the probability of participation in the coalition.

Load Balancing

The LB algorithm is executed periodically, and it balances the load by moving jobs or their fragments between resources. The algorithm is given as

Algorithm 4.1. The algorithm starts by computing the mean load $\overline{L}$ from the loads of resources in the coalition. Then, the resources with load greater than the mean send fractions of their load to the ones with the load lower than the mean. We assume no priorities in the load. Every job which parts were load balanced is completed in exactly the same time moment $\overline{L}$. Consequently, after LB, the load of all resources in the coalition finishes exactly at $\overline{L}$. The next iteration of LB is performed at $\overline{L}$.

The complexity of the algorithm is in $O(N^2)$.

Strategies

Organization O_k that participated in the coalition adjusts its strategy σ_k at the moment of deciding whether to join the coalition for the next time period by Algorithm 4.2. σ_k is based on the observed sum of flow times F_k of local jobs submitted during the last time period. O_k computes (loop in line 3) what would have been the sum of flow times $\widetilde{F_k}$ of local jobs if the organization had not joined the coalition.

By comparing F_k and $\widetilde{F_k}$, the organization can measure if LB was profitable, and adjust p_k accordingly. More specifically, each organization M_k that was participating in the coalition computes the ratio $R_k = \frac{F_k}{\widetilde{F_k}}$ (line 8). In order to limit the maximum possible changes, we bound the ratio (by $\frac{1}{2}$ and 2) (line 9). Then, the organization computes the one-round desire to join the coalition q_k, which depends on R_k. If $R_k > 1$, LB worsened the performance of local jobs (line 10). Otherwise, the ratio is inverted (line 11) (so that the impact of $R_k = 2$ and $R_k = \frac{1}{2}$ will be the same).

For instance, if the performance of LB is the same as the performance if all the jobs had been executed locally, the organization is indifferent about joining the coalition, $R_k = 1$, and $q_k = \frac{1}{2}$. If the time of the execution of local jobs during LB is twice as much, $R_k = 2$ and the organization does not want to join the coalition, $q_k = 0$.

The value of p_k is then adjusted by the value of q_k multiplied by a small number expressing the learning coefficient (line 12).

Algorithm 4.2: Algorithm adjusting p_k for $O_k \in C$.

Input:

- $\mathcal{J}_k$ list of jobs produced during last time period, ordered by non-decreasing release dates;
- $\mathbf{C_k} = [C_k^1, \ldots, C_k^{n_k}]$ completion times of $\mathcal{J}_k$ with LB;
- η learning coefficient, $0 < \eta \leq 1$;
- p_k current value of p_k.

Output: p_k adjusted by $\mathcal{J}_k$ performance

$\widetilde{F_k} = 0$;
$qEnd = 0$;
foreach $J_k^i \in \mathcal{J}_k$ **do**
    $qEnd = \max(qEnd, r_k^i) + p_k^i$;
    $\widetilde{F_k} += qEnd - r_k^i$;
$F_k = 0$;
foreach $J_k^i \in \mathcal{J}_k$ **do** $F_k += C_k^i - r_k^i$;
$R_k = \frac{F_k}{\widetilde{F_k}}$;
$R_k = \min(2, \max(0.5, R_k))$;
if $R_k > 1$ **then** $q_k = 1 - \frac{R_k}{2}$;
else $q_k = \frac{1}{2R_k}$;
$p_k = (1 - \eta)p_k + \eta q_k$;
return p_k ;

The complexity of the whole algorithm is in $O((n_k)^2)$.

If an organization remained outside of the coalition, it cannot compute the relative performance R_k. In this case p_k does not change. As initially organizations are indifferent to the coalition, the starting value of p_k is $\frac{1}{2}$.

4.5.2 Experimental Analysis

We simulated the grid environment with $N = 10$ organizations participating in the previously described game using a custom-built discrete event simulator.

Unless otherwise stated, we assume that the resources are similarly loaded, so the expected delay between two consecutive jobs is the same $\frac{1}{\lambda}$, $\lambda = 0.2$. Jobs' sizes follow the gamma distribution with parameters (α, β), $\alpha = 2$, $\beta = 2$. The period of cooperation in coalition is $T = 1000$. unless otherwise stated. We repeated each experiment 50 times. The results obtained in each repetition were very similar to each other and the same phenomena could be observed.

In algorithm adjusting p_i, when computing $\widetilde{F_i}$ and F_i, we took into account only jobs that were submitted between $t_{prev} + 0.2T$ and $t_{prev} + 0.8T$ (where t_{prev} is the time at which the previous decision was taken), i.e. satisfying $t_{prev} + 0.2T \leq r_k^i \leq t_{prev} + 0.8T$. This allows us to avoid transitory effects and to capture the performance of the system in the steady-state.

Fig. 4.3.a depicts a typical course of a simple cooperation game with identically-loaded resources. After a short period of adjustment, the probability of joining the coalition oscillates between 0.8 and 0.9 with a few peaks. We see that joining the coalition is, in general, more profitable than computing the whole load locally. Otherwise, p_k values would drop below 0.5. The probabilities do not, however, converge to 1.0. This suggests that it was common that LB delayed the execution of local jobs. We also varied the number of organizations N from 2 to 30. We observed similar results, although the more organizations, the higher final value of p_k was observed.

Table 4.2: Mean score and standard deviation achieved by individual organizations in 1000 rounds of the simple cooperation game. $\lambda = 0.2$

		score / standard deviation	
no. of organizations	parameters	typical	overloaded
homogeneous configuration			
2	$\alpha = \beta = 2.0$	0.50 ± 0.00 / 0.65 ± 0.00	
5	$\alpha = \beta = 2.0$	1.25 ± 0.06 / 1.29 ± 0.10	
10	$\alpha = \beta = 2.0$	1.76 ± 0.12 / 1.64 ± 0.11	
30	$\alpha = \beta = 2.0$	2.37 ± 0.11 / 2.06 ± 0.20	
2	$\alpha = \beta = 1.0$	0.13 ± 0.00 / 0.06 ± 0.005	
5	$\alpha = \beta = 1.0$	0.23 ± 0.00 / 0.10 ± 0.00	
10	$\alpha = \beta = 1.0$	0.28 ± 0.01 / 0.11 ± 0.01	
30	$\alpha = \beta = 1.0$	0.30 ± 0.01 / 0.12 ± 0.01	
heterogeneous configuration, $\alpha_1 = \beta_1 = 3$			
2	$\alpha = \beta = 2.0$	-7.13 ± 0.00 / 3.66 ± 0.00	0.93 / 0.24
5	$\alpha = \beta = 2.0$	-1.98 ± 0.06 / 1.53 ± 0.04	5.11 / 2.05
10	$\alpha = \beta = 2.0$	-0.14 ± 0.03 / 0.93 ± 0.04	14.23 / 5.53
30	$\alpha = \beta = 2.0$	1.55 ± 0.11 / 1.57 ± 1.17	36.99 / 9.29
2	$\alpha = \beta = 1.0$	-24.21 ± 0.00 /12.29 ± 0.00	4.45 / 2.14
5	$\alpha = \beta = 1.0$	-0.55 ± 0.01 / 0.27 ± 0.01	62.42 /11.92
10	$\alpha = \beta = 1.0$	0.01 ± 0.00 / 0.10 ± 0.01	98.67 /18.96
30	$\alpha = \beta = 1.0$	0.23 ± 0.01 / 0.11 ± 0.01	131.99 /25.42

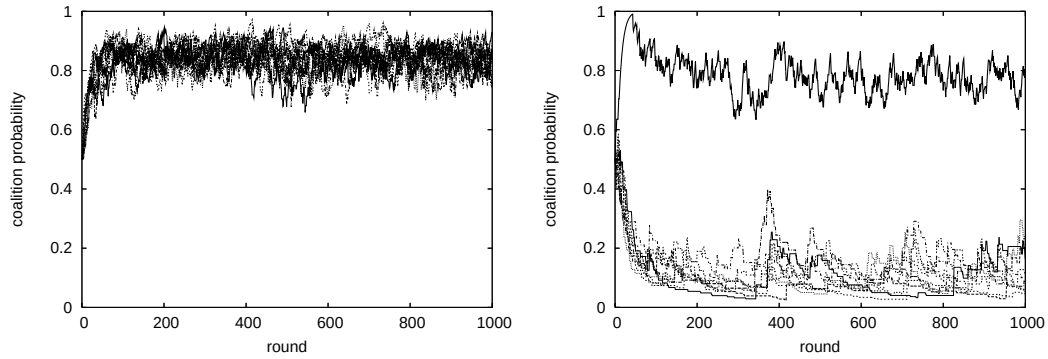


Figure 4.3: The probability of joining the load-balancing coalition p_k plotted against the round number in the homogeneous case (a) and when one of the resources (its p_k is represented by a thick black line) is overloaded (b). p_k values for individual organizations are depicted by different styles of lines.

The speed-up experienced by organization O_k can be defined as O_k's *score* $S_k = \frac{\widetilde{F_k}}{F_k} - 1$, if $F_k < \widetilde{F_k}$ (LB was better for O_k than local computation) and $S_k = -(\frac{F_k}{\widetilde{F_k}} - 1)$ otherwise. If $S_k = 1$, O_k's jobs' flow time when LB was used was reduced by a half comparing to local computation, if $S_k = -1$, O_k's jobs' flow time was twice as much. S_k expresses how well the LB algorithm performed.

In order to measure the score, we fixed $p_k = 1.0$ for all the organizations, and performed $N = 1000$ rounds of the game without using the p_k's adjustment algorithm described in the previous section. Table 4.2 presents the aggregated results. As scores for individual organizations were similar, for brevity we present only the average score (computed as an average from individual organizations' scores, which, in turn, are averages over 1000 runs), the actual range in which individual organizations' scores belonged, and the average standard deviation, computed as the average from standard deviations of score for each organization, again with the actual range. We can see that, on the average, if the resources were similarly loaded and their load was high ($\alpha = \beta = 2.0$ with $\lambda = 0.2$ gives average load of 80%), jobs were

speeded up from 50% to 237%. The more organizations in the grid, the better the results. This is because if there are more resources, more resources can actually send their load and profit from the LB. In the underloaded case ($\alpha = \beta = 1.0$, which gives average load of 20%), the algorithm was not as efficient, because there were not as many jobs to be load-balanced. However, performance varied heavily, as standard deviations are high.

However, one resource overloaded with local jobs, for instance O_1 in Fig. 4.3.b (with $\alpha_1 = 3$, $\beta_1 = 3$), is able to hinder others from cooperating. p_1 of such an organization quickly reaches a value slightly higher than in the previous experiment. The rest of the organizations does not want to collaborate with O_1, so their probabilities oscillate around 0.1. There are sharp increases, when LB gets profitable for an underloaded organization. After such an increase we usually observe a gradual downfall, meaning that the organization constantly loses performance in comparison to individual computing. Such an overloaded organization has also enormous impact on the mean score achieved by other organization. While the score achieved by M_1 is high, others are constantly loosing by cooperating with this organization. Only in largest grids (30 participants in the normally loaded case, 10 and 30 participants in underloaded case) other organizations gain, although this gain is much smaller compared with the homogeneous case.

4.6 Load-Balancing Based on Queue Length

Both effects observed in the previous section suggest that, although LB is indeed profitable, organizations should perhaps make their decisions on a finer level. However, it will be shown in this section that if each organization is able to decide about participating in the load-balancing process based on the current queue length on the local resource, cooperation will never appear, and therefore the grid will work inefficiently.

Let us assume a grid composed of two organizations O_1 and O_2 that participate in an averaging LB algorithm. The organization with bigger local

load sends a half of the difference in load to the less loaded organization. The amount of load s_k sent by organization O_k is:

$$s_k(L_k, L_l) = \frac{L_k - L_l}{2}.$$

If $s_k > 0$, organization O_k sends a fraction of its load to organization O_l. If $s_k < 0$, O_k receives a fraction of O_l's load.

Given resource's M_k current load L_k and its expectations about the distribution of load in the other resource, M_k can compute the expected result of LB process. Let us assume that organizations are identically loaded and they know their average arrival rate λ and the average service rate μ. Let's denote $\gamma = \mu - \lambda$ and $\rho = \frac{\lambda}{\mu}$. Then, for each resource, its queue length L_k has a mixed distribution having an impulse at $L_k = 0$ and being exponential for $L_k > 0$ [Larson and Odoni, 1981]. For our analysis, this distribution can be approximated by the following continuous distribution:

$$f(L_k) = \begin{cases} 0 & \text{for } L_k < 0 \\ (1-\rho)\delta(L_k) + \rho\gamma e^{-\gamma L_k} & \text{for } L_k \geq 0 \end{cases},$$

where $\delta(L_k)$ is Dirac delta function. Knowing this, M_k can compute the expected value of $s_k(L_k, L_l)$ as:

$$\begin{aligned} ES_k(L_k) &= E\Big(\frac{L_k - L_l}{2}\Big) = \frac{1}{2}\Big(L_k - EL_l\Big) = \\ &= \frac{1}{2}\Bigg\{L_k - \Big(0 \cdot (1-\rho) + \\ &+ \int_0^\infty \rho\gamma L_k e^{-\gamma L_k} dL_k\Big)\Bigg\} = \frac{1}{2}\Big(L_k - \frac{\rho}{\gamma}\Big). \end{aligned}$$

Consequently, organization O_k expects that it will send its jobs only if its current queue is longer than the expected queue length of M_l, $\frac{\rho}{\gamma}$. Only in this case LB is profitable for O_k. Organization O_l, however, is capable of the same analysis. If M_l's current queue length L_l is less than the average $\frac{\rho}{\gamma}$, M_l will not participate in LB process. In order to carry out the LB, we need two organizations willing to participate. Consequently, if O_k wants to participate,

it must know that if O_l participates (and the LB occurs), O_l's queue length will be longer than $\frac{\rho}{\gamma}$. This modifies the probability distribution function (pdf) of M_l's queue length. Following the general formula for conditional probability $Pr(A|B) = \frac{Pr(A \text{ and } B)}{Pr(B)}$ the pdf of M_l's queue length, given that $L_l > \frac{\rho}{\gamma}$ can be expressed as:

$$f^*(L_l|L_l > \frac{\rho}{\gamma}) = \begin{cases} \frac{f(L_l)}{1-F(\frac{\rho}{\gamma})} & \text{if } L_l \geq \frac{\rho}{\gamma} \\ 0 & \text{if } L_l < \frac{\rho}{\gamma} \end{cases},$$

where $f(L_l)$ is the original pdf and $F(L_l) = 1 - \rho e^{-\gamma L_l}$ is its cumulative distribution function. Then, the expected value of s becomes:

$$\begin{aligned} ES_k(L_k|L_l > \frac{1}{\gamma}) = & \frac{1}{2}\Big\{ L_k - \\ & - \Big(\int_{\frac{\rho}{\gamma}}^{\infty} \frac{\rho\gamma L_k e^{-\gamma L_k}}{1-(1-\rho e^{-\gamma\frac{\rho}{\gamma}})} dL_k \Big) \Big\} \\ & = \frac{1}{2}\Big(L_k - \frac{\rho+1}{\gamma} \Big). \end{aligned}$$

Consequently, LB becomes profitable for organization O_k only if its queue length is longer than $\frac{\rho+1}{\gamma}$. We can, however, iterate this reasoning further. Generally, the following formula holds:

$$ES_k(L_k|L_l > \frac{\rho+n}{\gamma}) = \frac{1}{2}\Big(L_k - \frac{\rho+n+1}{\gamma} \Big).$$

It makes the one-step LB analogous to the *beauty contest* game [Camerer, 2003]. The more iterations of the above reasoning the players make, the longer the minimum queue length which makes LB profitable. As the players are completely rational, the minimum profitable queue length increases indefinitely. The LB will thus never occur.

Obviously, in the longer time period, M_k's queue length will also follow the same distribution as M_k's. The expected result of LB (computed as: $ES_k = \frac{1}{2}(EL_k - EL_l)$) becomes then zero. Each organization can expect that it will send, on average, as much load as it will receive. In the analysis in Section 4.2 we showed that the cost EG of accepting a fraction of load of size Φ is always less than Φ, the profit the sender gets from sending it. Therefore, in the longer time period, the LB process is indeed profitable.

4.7 Bounded, Iterative Load Balancing

In Section 4.5 we showed that, although cooperation is profitable, one overloaded resource is able to make the LB unprofitable for the rest of the organizations. In Section 4.6 we demonstrated that it is impossible to formulate rational strategies based on the current queue length. Consequently, the only part of the system that can be improved is the LB algorithm. In this section we will propose Bounded, Iterative Load Balancing algorithm (BILB) that balances the load in a more equitable way. The algorithm uses two mechanisms: iterative LB, in which firstly the least-loaded resources are balanced, then the resources are iteratively added in the order of their local load; and bounded LB, in which no resource is forced to accept foreign load which would increase its local queue too much. The former technique favors underloaded resources, the latter directly uses conclusions from the cost model presented in Section 4.2, where we showed that accepting foreign load is cheaper when local queue is short.

This section presents the load balancing algorithm (Section 4.7.1), an analysis (Section 4.7.2), the algorithm the organizations use to modify their strategies (Section 4.7.3) and the experiments (Section 4.7.4).

4.7.1 BILB Algorithm Description

Principle

BILB allows each organization to control the maximum foreign load it will receive. Each organization, once every T time units, instead of declaring whether it participates to the coalition or not, announces its declared participation level l_k. l_k value is expressed in the same units as job length. The algorithm guarantees that, after the LB process completes, if a resource has been assigned some foreign load Φ_k, the resource's queue length will not extend l_k. In order to prevent that organizations declare $l_k = 0$, an overloaded resource will not be able to send more than l_k of its load. Let us denote the

Algorithm 4.3: Bounded Iterative Load Balancing. Φ_k denotes the total load received by M_k, Ψ_k the total load sent by M_k.

Input: $\mathbf{L} = [L_1, \ldots, L_N]$, vector of current loads of resources; without loss of generality we assume that $L_1 \leq L_2 \leq \ldots \leq L_N$

1 $\overline{L_1} = L_1$;
2 **for** $k = 2$ **to** N **do**
3 $\quad \overline{L_k} = \frac{(k-1)\overline{L_{k-1}}+L_k}{k}$;
4 $\quad surplus_k = \max(L_k - \overline{L_k}, l_k)$;
5 $\quad$ **for** $l = 1$ **to** $k-1$ **do**
6 $\quad\quad M_l.space = \min(\overline{L_k} - (L_l + \Phi_l), l_l - L_l - \Phi_l + \Psi_l)$;
7 $\quad receivers = \texttt{sort}((M_1, \ldots, M_{k-1}), M_l.space)$;
8 $\quad$ **for** $i = 1$ **to** $k-1$ **do**
9 $\quad\quad$ let l denote the index of machine M_l such that $receivers[i] = M_l$;
10 $\quad\quad \Phi_{l,k} = \min(\frac{surplus_k}{k-i}, M_l.space)$;
11 $\quad\quad \texttt{send}(M_k, M_l, \Phi_{l,k})$;
12 $\quad\quad surplus_k -= \Phi_{l,k}$;

load that M_k sends as a result of LB as Ψ_k. Using this notation, $\Psi_k \leq l_k$. By l_k, an organization can control the risk of taking part in LB. Lower values mean lower risk, but also smaller load to be sent if the resource is overloaded. Higher values enable the resource to send much load, if it is overloaded, but also could force it to accept much foreign load.

BILB balances the load of a machine with machines that have smaller loads. Consequently, the machines that have their load smaller than the overall average can gain from the process.

Algorithm

BILB is described as Algorithm 4.3. The algorithm starts by collecting the loads L_k, and the declarations of maximum participation l_k from organizations. Firstly, the resources are sorted according to the increasing queue lengths. Let us assume that $L_1 \leq L_2 \leq \ldots \leq L_n$. Then, for each resource M_k,

its load is balanced with $\{M_1, \ldots, M_{k-1}\}$ (loop starting at line 2). These resources have their load already balanced, with mean queue length $\overline{L_{k-1}}$. The maximum load M_k will send, $surplus_k$, is bounded by the average $L_k - \overline{L_k}$ and M_k's announced participation l_k, $\Psi_k^* = \max(L_k - \overline{L_k}, l_k)$ (line 4). $surplus_k$ is then divided onto $\{M_1, \ldots, M_{k-1}\}$. On each receiver M_l, the space available for $surplus_k$ is bounded by the delay to the average $(L_i + \Phi_i - \overline{L_k})$ and by the declared participation $l_i - L_i - \Phi_i + \Psi_i$ (Φ_i is the total load already received by M_i, Ψ_i is the total load sent by M_i) (line 6). Receivers are sorted by increasing space available for M_k load (line 7). Then, for each receiver M_l, the load $\Phi_{l,k}$ actually received is bounded by the space available and the number of remaining receivers $\Phi_{l,k} = \frac{surplus_k}{k-i}$ (line 10). In other words, if M_l receives anything, $L_l + \Phi_l - \Psi_l \leq l_l$. Note that after such a bounding, the actual load $\Psi_k = \sum_{i=1}^{k-1} \Phi_{i,k}$ that M_k sends can be lower than the value computed in line 4.

Next iteration of BILB is performed after $\min_k(L_k + \Phi_k - \Psi_k)$.

Complexity

The computational complexity of BILB is in $O(N^2)$.

4.7.2 Algorithm Analysis

The two mechanisms used by the algorithm are difficult to analyze theoretically together, however we show that they both contribute to the desired effects.

Firstly, iterative LB achieves more equitable solutions than averaging LB. Assuming that $L_1 \leq \ldots \leq L_k \leq \ldots \leq L_n$, as the load sent in the averaging LB depends on the average load $s_k = L_k - \overline{L}$, resource M_k will profit from averaging LB only if its load is greater than the average load $\overline{L}$. Otherwise it will be just receiving foreign load. However, in iterative LB, M_k sends a fraction $\Psi_k = L_k - \overline{L_k}$. Since the resource balances its load with less loaded resources, $L_1 \leq \ldots \leq L_k$, the average is smaller than or equal to the resource's load $\overline{L_k} \leq L_k$, so the load send $\Psi_k \geq 0$. Only the least loaded

resource will only receive the load. Iterative LB is also profitable for the heavier loaded resources. In fact, for each resource M_k, $\overline{L_k} \leq \overline{L}$, so $\Psi_k \geq s_k$ (and this becomes an equality only for the heaviest loaded resource). Each resource sends at least the same amount of load as in averaging LB.

Additionally, iterative LB makes jobs finish earlier. Resource M_k finishes all its jobs that are known in the moment of LB until the time $\overline{L_k}$, whereas in averaging LB jobs are finished later ($\overline{L} \geq L_k$). Only the jobs of the most loaded resource M_N are not accelerated in comparison with averaging LB.

Bounding the load that a receiver may accept has two goals. Firstly, through parameter l_k organization O_k can control its participation in the algorithm on a finer level, as it is able to balance the risk of getting a load of size l_k (which would induce the cost $EG(L_k, l_k)$), and the gain from sending its job of size l_k. Secondly, this mechanism uses the observation that the longer the resource's queue, the more expensive is foreign job's execution. The longer the queue, the more probable that the next job J_k^n arrives before L_k. Thus, it is more probable that J_k^n will be delayed by foreign load received as a result of LB. When a resource is heavily loaded, it should not receive any foreign load, as it is almost certain that next local job arrives before the foreign load is finished.

4.7.3 Strategies

The algorithm executed by organizations to modify their strategies is analogous to Algorithm 4.2. However, after computing R_k, each organization computes the declared participation as follows (this replaces lines 10-12 in Algorithm 4.2):

if $R_k > 1$ **then**
    $l_k = l_k - \eta l_k (R_k - 1)$;
else
    $l_k = l_k + \eta l_k (\frac{1}{R_k} - 1)$;
$l_k = \max(1, \min(l_k, 1000))$;

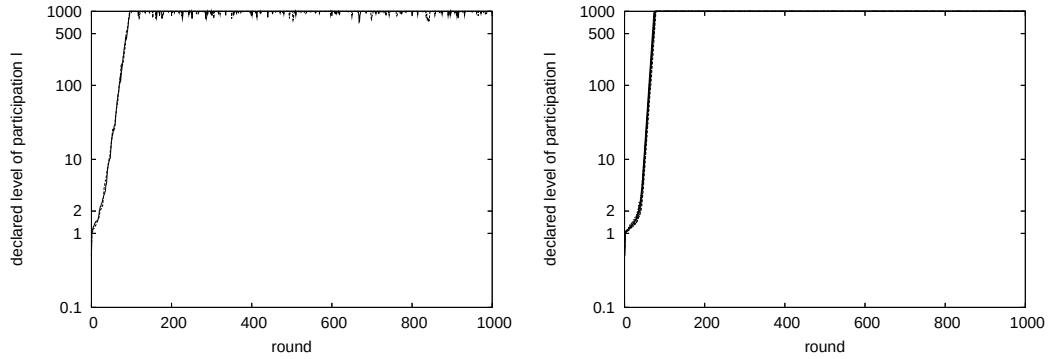


Figure 4.4: The declarations of maximum participation of each organization l_k plotted against the round number in the homogeneous case of two organizations (a) and of ten organizations (b). $\alpha = \beta = 2$, $\lambda = 0.2$, l_k values for individual organizations are depicted different styles of lines.

Similarly to Algorithm 4.2 lines 10-12, this operation makes the change of l_k proportional to the performance achieved.

4.7.4 Experimental Analysis

We simulated the BILB algorithm in a system similar to the one described in Section 4.5.2. The parameters of the system were the same as in the previous experiments. For the plots of optimum participation level, we repeated each experiment 10 times and observed no significant differences between individual runs.

When resources are similarly loaded, LB is highly profitable as the declared levels of cooperation l_k quickly reach the upper bound (Fig. 4.4). As such a bound is strongly greater than observed queue lengths, an organization, by declaring $l_k = 1000$, essentially turns off the bounding mechanism and declares that it will accept everything.

When resources' loads differ, both mechanism used in BILB are needed depending on the number of overloaded resources in the system. Firstly, in

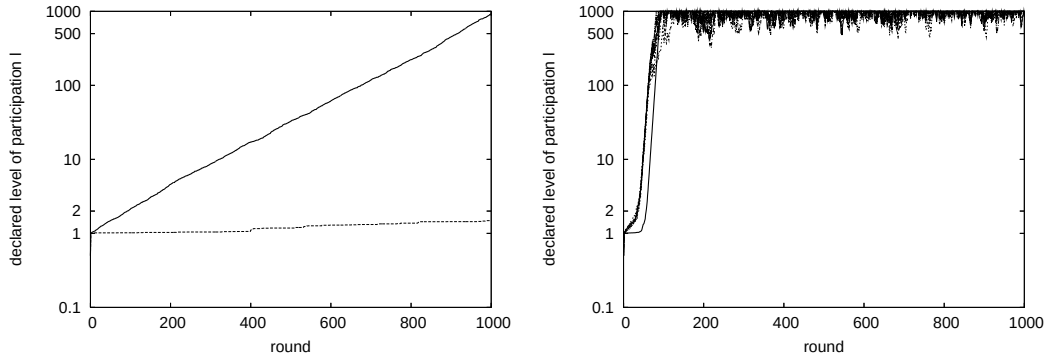


Figure 4.5: The declarations of maximum participation of each organization l_k plotted against the round number in the heterogeneous case of two organizations (a) and of ten organizations (b). For M_1, plotted in thick, black line, $\alpha_1 = \beta_1 = 3$, whereas for the other organizations, $\alpha = \beta = 2$. l_k values for individual organizations are depicted by different styles of lines. $\lambda = 0.2$

smaller grids (Fig. 4.5.a) composed of 2 organizations, the underloaded organization O_2 declares its participation level l_2 less than 1.5, which is rounded to 1.0 by the algorithm. This is a consequence of the discrete nature of our simulator, as the first job after LB may come at earliest in the next time unit. Clearly, if LB was performed before job submission, l_2 would be equal to 0. This result shows, however, that the algorithm used for l_k modification works correctly: O_2 cannot gain by load-balancing its jobs (as M_1's queue is with almost always longer), but it can accept a foreign load of size 1, since it will not delay its local jobs.

Secondly, in larger grids (Fig. 4.5.b), we can see that the declared participation level of all organizations eventually reaches the upper bound. This result, although surprising, is justified by the "iterativeness" of the LB algorithm. As resources with smaller loads are balanced firstly, they are able to gain from the process, without being flooded by overloaded resource's jobs.

Both results show that, in function of particular grid configuration and resources' load level, both mechanism used in BILB are needed.

Table 4.3: Mean score and standard deviation achieved by individual organizations in 1000 rounds of the BILB in function of different levels of declared participation l_k. 2 and 10 organizations with $\lambda = 0.2$, $\alpha = \beta = 2$.

	two organizations				**ten organizations**			
l_k	score	$\pm$	std dev	$\pm$	score	$\pm$	std dev	$\pm$
1	0.125	0.006	0.123	0.014	0.076	0.006	0.055	0.006
2	0.258	0.003	0.251	0.012	0.261	0.016	0.190	0.016
3	0.387	0.01	0.364	0.001	0.710	0.043	0.515	0.039
4	0.48	0.006	0.467	0.004	1.429	0.072	1.059	0.131
5	0.574	0.006	0.568	0.008	2.148	0.092	1.541	0.153
6	0.587	0.018	0.587	0.021	2.334	0.085	1.701	0.054
7	0.609	0.013	0.642	0.033	2.518	0.124	1.878	0.150
8	0.606	0.017	0.639	0.019	2.661	0.095	1.999	0.174
9	0.608	0.011	0.617	0.007	2.793	0.114	2.095	0.184
10	0.592	0.017	0.648	0.026	2.912	0.083	2.197	0.155
20	0.613	0.007	0.713	0.019	3.116	0.140	2.413	0.195
50	0.623	0.004	0.737	0.031	3.134	0.230	2.372	0.253
100	0.636	0.039	0.774	0.035	3.135	0.153	2.427	0.169
1000	0.628	0.011	0.729	0.006	3.123	0.139	2.393	0.212

Table 4.4: Mean score and standard deviation achieved by individual organizations in 1000 rounds of the BILB in function of different levels of declared participation l_k. Heterogeneous grid, one overloaded resource M_1 has $\alpha_1 = \beta_1 = 3$, the rest $\alpha = \beta = 2$. $\lambda = 0.2$. The overloaded organization's declared participation level is fixed to $l_1 = 1000$.

	two organizations				**ten organizations**					
	typical		overloaded		typical				overloaded	
l_k	score	std dev	score	std dev	score	$\pm$	std dev	$\pm$	score	std dev
1	0.003	0.017	0.034	0.022	0.079	0.006	0.057	0.006	0.003	0.004
2	-0.005	0.047	0.074	0.046	0.267	0.014	0.196	0.021	0.019	0.017
3	-0.03	0.058	0.12	0.078	0.687	0.033	0.510	0.050	0.097	0.069
4	-0.062	0.062	0.152	0.106	1.290	0.055	0.946	0.084	0.382	0.251
5	-0.098	0.081	0.185	0.146	1.807	0.058	1.343	0.098	1.116	0.855
6	-0.138	0.101	0.194	0.144	1.748	0.069	1.370	0.152	1.921	1.737
7	-0.167	0.116	0.192	0.141	1.696	0.066	1.375	0.098	3.086	2.823
8	-0.202	0.128	0.195	0.133	1.605	0.049	1.363	0.088	4.201	3.845
9	-0.233	0.152	0.202	0.159	1.545	0.086	1.315	0.128	5.664	5.116
10	-0.278	0.167	0.203	0.149	1.448	0.071	1.290	0.108	6.651	5.938
20	-0.661	0.353	0.232	0.166	0.887	0.059	1.181	0.081	13.635	10.712
50	-1.822	0.903	0.316	0.189	0.607	0.026	1.226	0.126	20.841	10.551
100	-3.753	2.005	0.501	0.272	0.591	0.057	1.226	0.089	21.845	9.602
1000	-7.054	3.641	0.949	0.253	0.609	0.104	1.229	0.161	22.018	9.423

Table 4.5: Mean score and standard deviation achieved by individual organizations in 1000 rounds of the BILB. $\lambda = 0.2$. In heterogeneous configurations, the overloaded organization declared $l_1 = 1000$.

		score / standard deviation	
no. of organizations	parameters	typical	overloaded
homogeneous configuration			
2	$\alpha = \beta = 2.0$, $l = 1000$	0.65 ± 0.00 / 0.73 ± 0.010	
5	$\alpha = \beta = 2.0$, $l = 1000$	2.00 ± 0.10 / 1.72 ± 0.14	
10	$\alpha = \beta = 2.0$, $l = 1000$	3.15 ± 0.17 / 2.45 ± 0.22	
30	$\alpha = \beta = 2.0$, $l = 1000$	4.56 ± 0.30 / 3.36 ± 0.24	
2	$\alpha = \beta = 1.0$, $l = 1000$	0.64 ± 0.02 / 0.79 ± 0.01	
5	$\alpha = \beta = 1.0$, $l = 1000$	1.65 ± 0.07 / 1.64 ± 0.12	
10	$\alpha = \beta = 1.0$, $l = 1000$	3.15 ± 0.01 / 2.41 ± 0.14	
30	$\alpha = \beta = 1.0$, $l = 1000$	4.55 ± 0.40 / 3.32 ± 0.60	
heterogeneous configuration, $\alpha_1 = \beta_1 = 3$			
2	$\alpha = \beta = 2.0$, $l = 1$	0.03 ± 0.02 / 0.00 ± 0.00	0.02 / 0.00
5	$\alpha = \beta = 2.0$, $l = 3$	0.11 ± 0.07 / 0.56 ± 0.02	0.44 / 0.03
10	$\alpha = \beta = 2.0$, $l = 5$	1.12 ± 0.82 / 1.80 ± 0.10	1.33 / 0.15
30	$\alpha = \beta = 2.0$, $l = 10$	29.77 ± 8.73 / 3.40 ± 0.15	2.52 / 0.23
2	$\alpha = \beta = 1.0$, $l = 1$	0.18 ± 0.05 / 0.00 ± 0.00	0.00 / 0.00
5	$\alpha = \beta = 1.0$, $l = 3$	1.90 ± 1.14 / 0.14 ± 0.00	0.07 / 0.00
10	$\alpha = \beta = 1.0$, $l = 5$	8.75 ± 4.55 / 0.21 ± 0.01	0.10 / 0.00
30	$\alpha = \beta = 1.0$, $l = 10$	40.36 ± 10.87 / 0.31 ± 0.02	0.12 / 0.03

In order to measure the score (computed as in Section 4.5.2), we fixed the declared level of participation l_k and performed $N = 1000$ rounds of the game without using the l_k's adjustment algorithm described in the previous section.

Firstly, we tested different levels of participation l_k in order to assess the quality of the adjustment algorithm. Tables 4.3 and 4.4 presents the example results for grid composed of 2 and 10 organizations aggregated in the same way as in Section 4.5.2 (each row is an average over 1000 runs with the same settings). In homogeneous cases, all organizations fixed their values of l_k to the same value. In heterogeneous cases, the overloaded organization O_1 declared $l_1 = 1000$, because it almost never receives any load and it is profitable that it sends as much as possible. The rest of the organizations declared the same l_k, shown in the different rows of the table.

In the homogeneous case (Tables 4.3), we can see that, indeed the maximum participation leads to best performance, although the score achieved is similar as for $l_k = 4$. What is, however, important, is that for lower participation levels, the standard deviation is also lower, which means that the risk is smaller for individual organizations.

In the heterogeneous case (Table 4.4) with two organizations, the minimum participation of the least loaded organization, which was proposed by the adjustment algorithm, indeed leads to the best performance of this organization. Nevertheless, in heterogeneous grid composed of 10 resources, for less loaded organizations it is better to reduce the l_k to 5. Performance achieved with $l_k = 5$ is significantly better than the performance for $l_k = 1000$, proposed by the adjustment algorithm. It should be noted that, with $\lambda = 0.2$, 5 is the average time between two jobs are released. If a resource accepts a foreign load which extends its queue length to 5, in average half of the local jobs will be delayed. Resources may also use more sophisticated techniques for learning their optimal participation level [Marks, 2001, Sastry et al., 1994].

In order to compare scores achieved by organizations using BILB with the simple algorithm (Table 4.2), we set l_k values which maximized the average

score in the same analysis as in Table 4.4. The results are shown in Table 4.5. We see that, firstly, BILB provides better results in the homogeneous setting, and secondly, it allows positive gains for all the organizations even when one of the resources is overloaded. BILB is especially profitable in heterogeneous configurations with many resources.

4.8 Discussion

The proposed solutions are simple, yet distributed systems applying them manage to avoid some common pitfalls. Firstly, every job submitted to the system has guaranteed completion time. This is a clear advantage over the best-effort mode used commonly for grid jobs.

Typical free-riding is impossible. By participating in the coalition, organizations will share their resources, as long as they are underloaded. If a resource is heavily loaded, the resource does not impede others from LB. It simply will use the resources that remain free after the less loaded organizations have balanced their load.

There are several ways in which organizations can attempt to cheat the algorithm in order to obtain better gain. Firstly, an organization can delay the execution of foreign load. However, others can immediately detect such behavior, because of the guarantees on the completion time. As we showed, generally, LB is profitable for each organization. Therefore, by declaring that such delayers will be penalized by temporal or permanent exclusion, the community is able to impose the desired behavior on every organization. This is a consequence of folk theorem in repeated games.

Secondly, organizations may be tempted to overstate the queue length on their resources in order to reduce, or even suppress the load assigned to them by the algorithm. However, such behavior may reduce organization's performance when the BILB algorithm is used. As the algorithm sorts resources by increasing queue length, the resource which declares larger local load can be overtaken by another resource, which, in turn, can fill the queues

of less loaded resources with its jobs, leaving no space for the cheater's jobs. Only the least-loaded resource will profit from overstating its actual load. In the general case, this problem is, unfortunately, similar to the n-player repeated Prisoner's Dilemma (PD). In PD, although mutual *cooperation* is socially-optimal, the Nash equilibrium is mutual *defection*. In our game, organization O_k *defects* when it declares its queue length $\widetilde{L_k}$ equal to the declared participation level l_k ($\widetilde{L_k} = l_k$) (higher declarations are dominated by this action), although the true queue L_k is shorter $L_k < \widetilde{L_k}$. Additionally, other organizations cannot observe the true L_k, and therefore can guess the action undertaken by O_k only by the observed result of the LB algorithm. We think that, similarly to other real-world PD occurrences, the only way to prevent such situations is an out-of-game verification of organizations. Such control can be performed e.g. by sporadic test of the true queue length. Each organization must provide an account, indistinguishable from other accounts of that organization, by which an auditor would submit a job and note the current queue length looking at the guaranteed finish time for that job. Alternatively, the community may impose a particular, trusted queuing software to be installed on each resource. Although the software may be modified locally, periodic updates (that are, again, imposed by the community) make such efforts unprofitable. This approach is used in EGEE grid [EGEE, 2007].

One of the drawbacks of the proposed solutions is the centralization of the load balancing algorithm. This, however, should not be a problem in typical grid systems, composed of tens (rather than thousands) resources. Recall that the computational complexity of the algorithm is small ($O(N^2)$). In larger systems, the load could be balanced in a more distributed fashion, with a number of instances of BILB running on overlapping fragments of the system.

4.9 Summary

In this chapter we proposed a new method for load balancing in decentralized systems owned by multiple parties. We presented a new cost function in which the cost of execution depends both on the size of a job and on the local load (Section 4.2). Although load balancing is equitably-optimal (Section 4.3), we demonstrated that it will never be performed when players are myopic (Section 4.4). We overcame this issue by forcing players to commit to their decisions for longer periods of time (Section 4.5. In the basic setting, when resources were similarly loaded, even a simple queue-averaging load balancing algorithm gave acceptable results for all the players. Then we proved that if organizations choose their actions according to the current queue length, cooperation will never occur (Section 4.6). Finally, we addressed the issues of load heterogeneity by designing another algorithm, called Bounded Iterative Load Balancing, which delivers more equitable solutions by employing two techniques: bounding the foreign load assigned for execution on a resource by limiting the maximum length of the queue; and load balancing the least-loaded resources first, so that they could also gain from the process (Section 4.7). This algorithm delivers better results than the previous algorithm in all the settings considered. Moreover, it is able to produce gain even for the least-loaded organizations. Additionally, our system provides basic Quality of Service parameters for every submitted job, as it is capable of determining the job's worst-case finish time in the moment of the submission of the job.

Chapter 5

Rigid Load Balancing

In this chapter we study the multi-organizational extension of the classic parallel job scheduling problem. The classic version of the problem consists of scheduling jobs on a multiprocessor resource. Each job must be executed in parallel on a specific number of processors. The goal is to minimize the makespan, the time the last job completes. This model adapts well to a popular supercomputing scenario, in which a cluster is shared between a number of MPI jobs.

We extend the classic model to multiple organizations. In our model, each organization owns a multi-processor resource and produces jobs that are scheduled locally on its resource. Through grid-level load balancing, these jobs can be then executed on different resources. In this chapter we study approximation algorithms with a guarantee on the worst-case performance ratio. We aim to measure the impact of organizational and hardware decentralization on the approximation ratio.

After formally defining the model in Section 5.1, we show some basic results on scheduling jobs on one resource in Section 5.2. Then, we study the model with the three approaches defined in Section 2.4. Firstly, in Section 5.3, we show that, even if the grid scheduler retains complete control over all resources, the discontinuity of resources causes the approximation ratio of a simple List Scheduling algorithm to rise to 3. Moreover, the equity

of results cannot be guaranteed. Then, in Section 5.4, we show that the cost of organizational decentralization is not as high as in previously considered grids, because the Price of Anarchy is between 3/2 and 2. Finally, in Section 5.5, we propose a scheduling algorithm that has a fixed worst-case approximation ratio on the system makespan and at the same time does not worsen local makespans of any organization. We provide an experimental evaluation of the algorithms in Section 5.6.

5.1 Grid Model

In this chapter, we make the following assumptions (in addition to ones in Section 2.3) about the system:

- the model is *off-line*;
- the scheduler is *clairvoyant*;
- each resource has m processors;
- jobs are *parallel* and *rigid*. J_k^i must be executed in parallel on q_k^i processors of exactly one resource.
- there are no release dates;
- processors are *time-shared* between jobs. Resources are *space-shared* and *time-shared* between jobs;
- *preemption* is not allowed;
- in order to measure the performance, each organization computes the *makespan* $C_{\max}(O_k) = \max C_k^i$ of locally produced jobs.

Note that we do not allow a job to be executed in parallel on two, or more resources. This assumption is caused by the underlying heterogeneity of network links. As a job must be executed in parallel, we assume that fragments of the job executed on different processors will actively communicate with

each other. Inside one resource, such communication is fast. However, links between resources are a few orders of magnitude slower, so communication between resources would increase job's execution time.

Note also that with the assumption that each organization measures the completion time of its locally-produced jobs, and not the completion time of the local resource, executing a foreign job at the end of the schedule does not cause any cost for the owner. This is a direct consequence of the off-line assumption of the model.

We will now introduce notation and definitions used in the chapter. J_k^i is *low* if it needs no more than a half of resource's processors ($q_k^i \leq \frac{m}{2}$), otherwise it is *high*. Two *low* jobs can be executed in parallel on a resource.

We denote by $p_{\max} = \max p_k^i$ the maximum length of a job. By $W = \sum p_k^i q_k^i$ we denote the total surface of the jobs.

We denote by $\mathcal{I}_k$ the set of jobs allocated to be executed on resource M_k.

For resource M_k, a *schedule* π_k is a mapping of jobs $\mathcal{I}_k$ to processors and start times in such a way that in each time moment no processor is assigned to more than one job. A schedule is usually represented by a Gannt chart.

At any time moment t, $u_k(t)$ is defined as the number of currently used processors of M_k (i.e. processors that compute jobs). *Utilization* $U_k(t)$ is the ratio of the number of currently used processors to the total number of processors m, $U_k(t) = \frac{u_k(t)}{m}$.

5.2 Scheduling on One Resource

Let us first focus on the simple case of scheduling rigid parallel jobs on one resource consisting of m identical processors. As in this section there is no reason to distinguish between a resource and an organization, we simply use $C_{\max}$ to denote the makespan and omit the index of the organization in other notations (e.g. q_k^i becomes q^i). We use the classic list scheduling algorithm, which has an approximation ratio equal to $2-\frac{1}{m}$ [Eyraud-Dubois et al., 2007]. We show that if the jobs are ordered according to decreasing number of

Algorithm 5.1: List Scheduling (LS) algorithm.

Input:

- $\mathbf{J} = (J^1, \ldots, J^n)$, ordered list of jobs
- m number of processors

```
1  unscheduled = J ;
2  t = 0 ;
3  while not empty(unscheduled) do
4      m' = m − u(t) ;
5      foreach J^i ∈ unscheduled do
6          if q^i ≤ m' then
7              schedule(J^i, t) ;
8              m' = m' − q^i ;
9              unscheduled.remove(J^i) ;
10     t = min_{i:J_i∈(J−unscheduled)∧C^i>t} C^i ;
```

required processors, the resulting schedule can be divided into at most two periods regarding the utilization of the resource. Firstly, more than the half of the processors are used. Then, less than the half of the processors are used, during at most p_{max}. Preliminary results established in this section will be used later to solve the general problem of multi-organization scheduling in the rest of the chapter..

5.2.1 List Scheduling

List scheduling [Graham, 1969] is a class of heuristics which work in two phases. In the first phase, jobs are ordered into a list. In the second phase (detailed as Algorithm 5.1), the schedule is constructed by assigning jobs to processors in a greedy manner. Let us assume that in time moment t, m' processors are free in the schedule under construction (line 4). The scheduler chooses from the list the first job J^i requiring no more than m' processors

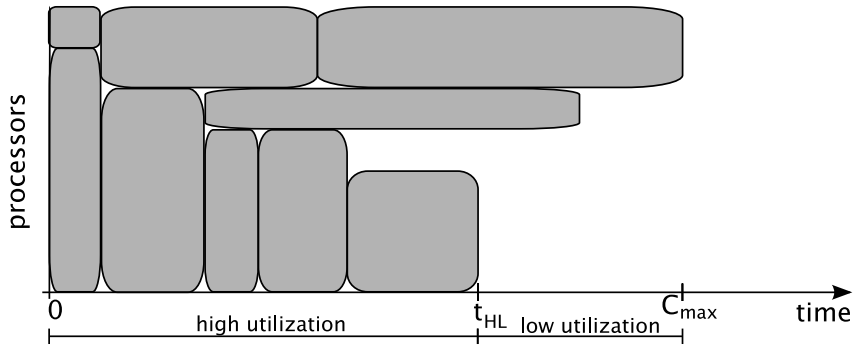


Figure 5.1: When jobs are presorted according to number of required processors, the schedule can be divided into two regions with utilization $U(t) > \frac{1}{2}$ (up to t_{HL}) and $U(t) \leq \frac{1}{2}$ (after that moment).

(line 6), schedules it to be started at t (line 7), and removes it from the list. If there is no such job, the scheduler advances to the earliest time moment t when one of scheduled jobs finishes (line 10).

List scheduling of rigid parallel jobs is an approximation algorithm with guaranteed worst case performance of $2 - \frac{1}{m}$ [Eyraud-Dubois et al., 2007]. The ordering of jobs in the first phase is not important. A polynomial time algorithm with better approximation ratio is not known.

The following lemma describes the utilization of the schedules returned by LS.

Lemma 5.1. *[Eyraud-Dubois et al., 2007] In a schedule constructed by LS with makespan C_{max}, the following property holds:*

$$\forall\ t, t' \in [0, C_{max}] : t' \geq t + p_{max} \Rightarrow u(t) + u(t + p_{max}) > m.$$

5.2.2 Highest First (HF) Job Order

The $2 - \frac{1}{m}$ approximation ratio of list scheduling does not depend on the particular order of jobs in the list. Therefore, we may choose a criterion which gives some interesting properties of the resulting schedule without loosing the approximation ratio.

Let us consider jobs ordered according to the Highest First (HF) rule, i.e. by non-increasing q^i. The following proposition holds.

Proposition 5.1. *All HF schedules have the same structure consisting of two consecutive regions of high ($t \in [0, t_{HL}) : U(t) > \frac{1}{2}$) and low ($t \in [t_{HL}, C_{\max}(M_k) : U(t) \leq \frac{1}{2}$) utilization, where $0 \leq t_{HL} \leq C_{\max}(M_k)$ (Figure 5.1).*

Proof: The proof is by contradiction. Let us assume that at time moment t the utilization is low ($U(t) \leq \frac{1}{2}$), and that at time $t' > t$ the utilization is high ($U(t') > \frac{1}{2}$). Note that all the jobs executed at t are *low* (they all require less than half of the available processors). Let us consider a job scheduled after t requiring less processors than it is available at t. If there is no such job, yet $U(t') > \frac{1}{2}$, at least one *high* job follows a *low* job, which contradicts the assumption of HF order of jobs (as, according to HF, the *high* job should have been scheduled before scheduling jobs that execute in parallel when job executing at t starts). If such a job exists, scheduling it at t would result in a smaller completion time. Consequently, scheduling it after t contradicts the greedy principle of the list scheduling algorithm. □

Proposition 5.2. *In the region $[t_{HL}, C_{\max}(M_k)]$ of low utilization, the function $U(t)$ is non-increasing.*

Proof: The proof is analogous to the previous proof. Let us assume that $U(t)$ increases at time moment t'' ($t'' > t_{HL}$). Consequently, a job starts at t''. This job must be *low*, otherwise utilization would increase to more than $\frac{1}{2}$, the case included in Proposition 5.1. However, at time t_{HL} there are at least $\frac{1}{2}$ processors free, so a *low* job could be started. Hence, scheduling the job at t'' contradicts the greed of the list scheduling algorithm. □

Proposition 5.3. *In a HF list schedule, no job starts after t_{HL}.*

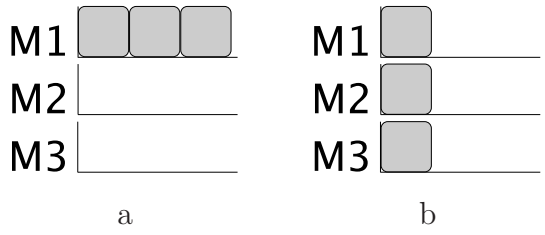


Figure 5.2: Executing all the jobs locally may lead to N approximation (a) ratio regarding the globally-optimal solution (b). All the jobs were produced by organization O_1, the owner of M_1.

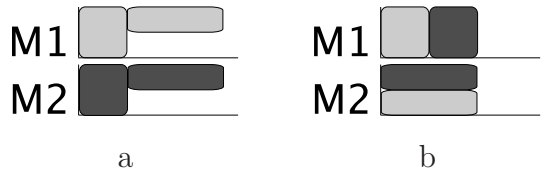


Figure 5.3: By matching certain types of jobs, cooperative solution (b) delivers better makespans for both organizations than a solution scheduling all the jobs locally (a). Light gray jobs were produced by organization O_1, dark gray by O_2.

Proof: The proof is by contradiction. Let us assume that a job J^i is started at $t' > t_{HL}$. J^i is *low*, because otherwise $U(t') > \frac{1}{2}$, which is excluded by Proposition 5.1. However, at time t_{HL} there are enough free processors to start J^i. Consequently, starting J^i at t' contradicts the greed of the list scheduling algorithm. □

5.3 Optimization Approach

5.3.1 Motivation

A number of instances motivate organizations to cooperate and accept non-local jobs, even if the resulting configuration is not necessary globally optimal. In a non-cooperative solution all the organizations compute their jobs on their

local resources. However, such a solution can be as far as N times worse than the optimal one (see Figure 5.2). Note also that careful scheduling offers more than simple load balancing of the previous example. By matching certain types of jobs, bilaterally profitable solutions are also possible (see Figure 5.3).

5.3.2 Problem Complexity

By restricting the number of organizations to $N = 1$, the size of the resource to $m = 2$ and the jobs to sequential ones ($q_k^i = 1$), we obtain the classic, NP-hard problem of scheduling sequential jobs on two processors $P2||C_{max}$ [Garey and Johnson, 1979]. Therefore, the considered problem is also NP-hard.

5.3.3 Approximation Ratio of List Scheduling

In this section we consider the approximation ratio of the LS algorithm. In the case of one resource, LS has approximation ratio of $2 - \frac{1}{m}$. However, as we assumed that a job cannot be executed in parallel on two resources, we cannot treat N resources as one resource having Nm processors. Such discontinuousness worsens the approximation ratio of the LS algorithm, even if we assume that all the resources and jobs are owned by one party.

Proposition 5.4. *Approximation ratio of LS algorithm scheduling parallel jobs on N identical multiprocessor resources is at least $2 + \frac{1}{4}$.*

Proof: We construct an instance for $N = 2$ and then generalize the result to N.

Let us consider an instance (depicted in Figure 5.4), in which there are three jobs of length 1 and of heights $q^1 = \frac{m}{2} - 1, q^2 = \frac{m}{2} + 1, q^3 = \frac{m}{2} + 2$ and four jobs of height $\frac{m}{2} - 2$ and of lengths $p^4 = p^5 = \frac{1}{4} + \epsilon, p^6 = p^7 = \frac{1}{4} - \epsilon$. The makespan of the optimal schedule is 1. However, a LS algorithm with priority list ($J^4, J^5, J^6, J^7, J^1, J^2, J^3$, results in makespan $2 + \frac{1}{4} - \epsilon$.

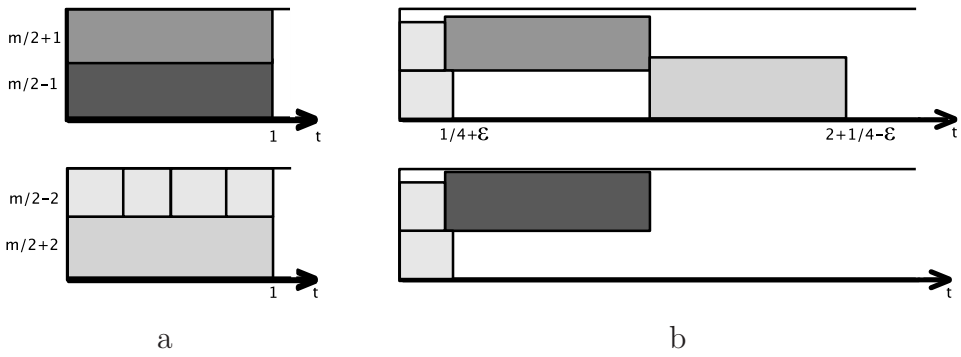


Figure 5.4: Even with no notion of organizations, LS on discontinuous resources has the approximation ratio of at least $2+\frac{1}{4}$ (b) regarding the optimal makespan (a). $N = 2$ resources with m processors, short jobs have lengths of $\frac{1}{4}+\epsilon$, $\frac{1}{4}-\epsilon$, $\frac{1}{4}+\epsilon$, $\frac{1}{4}-\epsilon$.

If N is even, similar instance can be constructed by considering $\frac{N}{2}$ copies of $\{J^1, \ldots, J^7\}$. The optimal schedule has makespan of 1, yet the worst LS schedule is the same as the worst schedule with $N = 2$. If N is odd, we construct the instance that is formed by jobs $\{J^3, \ldots, J^7\}$ and $\lfloor \frac{N}{2} \rfloor$ copies of $\{J^1, \ldots, J^7\}$. The optimal schedule has makespan of 1. The worst LS schedule repeats $\lfloor \frac{N}{2} \rfloor$ times the worst schedule for $N = 2$ and in the last resource schedules short jobs before J^7, thus it also has makespan $2 + \frac{1}{4} - \epsilon$. □

Two lower bounds on the global makespan can be defined. Let us denote as $W = \sum p_k^i q_k^i$ the total *surface* of the jobs, and as $p_{\max} = \max p_k^i$ the length of the longest job. Firstly, all the jobs must fit into available processors, so $C^*_{\max} \geq \bar{W} = \frac{W}{Nm}$. Secondly, the longest job must be executed, so $C^*_{\max} \geq p_{\max}$.

Proposition 5.5. *LS is a 3-approximation of* $C^*_{\max}$.

Proof: The proof is by contradiction. Let us assume that the last job finishes after $3C^*_{\max}$. It is thus started after $2C^*_{\max}$. At the moment the last job is started, all the other resources are busy (otherwise, the job would have

been started earlier). Let us denote as $LB = \max(\bar{W}, p_{max})$. Consequently, on each resource M_k we have $C_{\max}(M_k) \geq 2C^*_{\max} \geq 2LB$. Thus, From Lemma 5.1, on each resource M_k, we have (as $LB \geq p_{max}$):

$$u_k(t) + u_k(t + LB) \geq m \text{ for } 0 \leq t \leq \bar{W}.$$

After integrating this inequality, we get:

$$\int_0^{\bar{W}} u_k(t)dt + \int_0^{\bar{W}} u_k(t + LB)dt \geq m \int_0^{\bar{W}} 1dt.$$

$$\int_0^{\bar{W}} u_k(t)dt + \int_{LB}^{LB+\bar{W}} u_k(t)dt \geq m\bar{W}.$$

After adding inequalities for every resource M_k, we get:

$$\sum_{1 \leq k \leq N} \left(\int_0^{\bar{W}} u_k(t)dt + \int_{LB}^{LB+\bar{W}} u_k(t)dt \right) \geq Nm\bar{W} \geq W.$$

Left-hand side of the inequality is the surface of the jobs computed on all resources in periods $[0, \bar{W}]$ and $[LB, LB + \bar{W}]$. Those periods do not overlap. The surface computed is thus greater than the surface of all the tasks available, which leads to a contradiction. □

It rests an open problem whether this ratio can be improved.

5.3.4 Equitable Optimization

The above analysis concerned optimization of only one goal, i.e. the system-level makespan $C_{\max}$, albeit on multiple and non-contiguous resources. As all the jobs are mixed, we cannot have any guarantee on fairness between organizations' makespans $C_{\max}(O_k)$. In this section, we briefly analyze two ways in which LS can be extended to improve the fairness of the resulting schedules. Generally, we use techniques similar to iterative LB, proposed in Section 4.7.

Any fair algorithm should favor jobs of organizations that are less-loaded (regarding the organization's total work $W(O_k)$ or the length of the longest

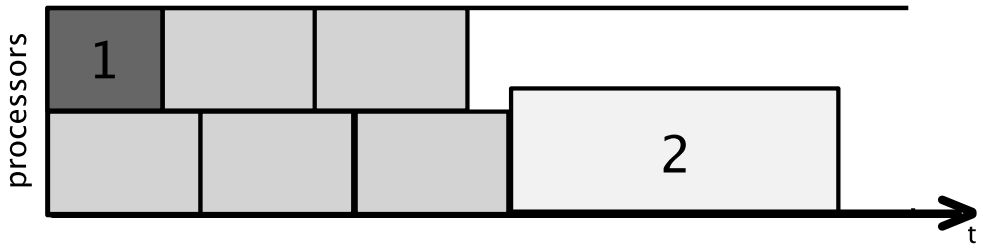


Figure 5.5: The greedy behavior of list-scheduling algorithm results in placing the job that has the second-highest priority (J^2, $q^2 = 6$ light gray) as the last job. $m = 10$ processors. The algorithm starts by scheduling the highest-priority job J^1 ($q^1 = 5$, dark gray), and then, hence there are 5 free processors left, schedules the gray jobs that have lower priority, but fit better into the available space.

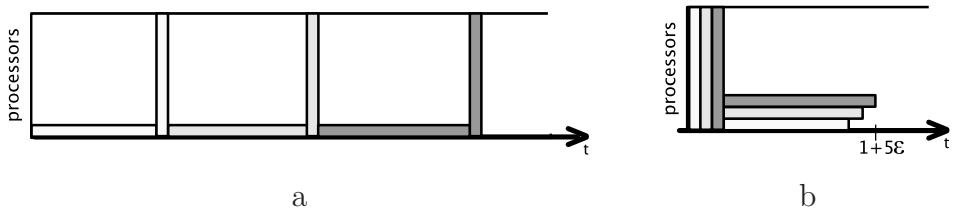


Figure 5.6: When less prioritized jobs cannot delay the already placed ones, the resulting schedule (a) can be N times longer than the optimal one (b). Here, different shades of gray correspond to different organizations owning the jobs. Long, sequential jobs have increasing lengths of $1, 1+\epsilon, 1+2\epsilon, \ldots$.

job $p_{\max}(O_k)$). However, such priorities are impossible to apply in a LS algorithm.

Firstly, job's priority can be expressed by ordering the job earlier in the first phase of LS algorithm. However, the greedy behavior of LS algorithm in the second phase can result in scheduling jobs having lower priority, but requiring less processors before higher priority jobs. Figure 5.5 shows an instance in which J^2, second-highest priority job, is scheduled as the last one.

Secondly, LS algorithm can schedule jobs of different organizations in

backfilling-like fashion. Assuming that O_1 has priority over O_2, which has priority over O_3,..., all the O_1's jobs would be LS on all available resources. Then, O_2 would be scheduled by LS between them with an additional restriction that a O_2's job cannot delay any O_1's job. The algorithm would then schedule O_3,...,O_N jobs similarly. Unfortunately, such an algorithm does not have guarantee on the approximation ratio. Figure 5.6 shows an instance that, scheduled with such an algorithm, has the makespan N times longer than the optimal one.

5.4 Game-Theoretic Approach

Let us firstly assume that each organization can control the schedule on the local resource. However, the allocation of jobs to resources is given in advance (i.e. determined by an independent, external entity), and cannot be thus modified.

Definition 5.1. *The Parallel Scheduling Game (PSG) is a strategic game in which:*

- *the set of players is equal to the set of organizations* $\mathcal{P} = \mathcal{O}$*;*
- *a strategy of* O_k *is schedule* π_k *of jobs* $\mathcal{I}_k$ *assigned for execution on* M_k*.* $\mathcal{I}_k$ *is known and cannot be altered by any player. The set of strategies of player* O_k *contains all possible schedules;*
- *the payoff of player* O_k *is the maximum completion time* $C_{\max}(O_k)$ *of* O_k*'s jobs* $\mathcal{J}_k$*.*

A strategy similar to My Job First (MJF, defined in Section 3.4) is as follows. An organization firstly schedules its local jobs with LS. Then, foreign jobs are scheduled either at the end of the schedule, or in gaps so that they do not delay any local job.

Proposition 5.6. MJF *is the only Nash equilibrium of PSG.*

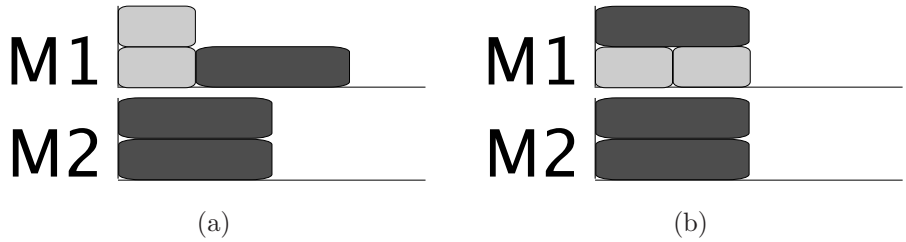


Figure 5.7: MJF strategy (a) results in extending the system makespan comparing to the globally-optimal strategy (b). Globally-optimal solution is dominated, as it extends the makespan of organization O_1 (the producer of light gray jobs) in comparison with MJF.

Proof: The proof is analogous to the proof of Proposition 3.8. For any organization O_k, given any scheduling on non-local resources, the strategy minimizing $C_{\max}(O_k)$ is a strategy that schedules the foreign jobs so that they do not delay any local job (MJF strategy). □

The inefficiency resulting from MJF strategies is, however, relatively low.

Proposition 5.7. *The Price of Anarchy is at least* $\frac{3}{2}$.

Proof: Consider an instance in Figure 5.7. **MJF** solution, depicted in (a), has $C_{\max} = 3$, whereas the global optimum has $C_{\max} = 2$. □

Proposition 5.8. *The Price of Anarchy is at most* 2.

Proof: The proof is constructive. We show how to build a **MJF** schedule from an optimal schedule by extending the makespan on each resource at most twice.

Let us consider an optimal schedule OPT_k on resource M_k resulting in $C^*_{\max}(M_k)$. MJF_k can be constructed as follows. Firstly, copy OPT_k and append it at the end of the original schedule (i.e. start the copy at $C^*_{\max}(M_k)$). Secondly, remove all the foreign jobs that start before $C^*_{\max}(M_k)$. Thirdly, remove all the local jobs that start at or after $C^*_{\max}(M_k)$. The schedule can be then compacted by removing the idle time between jobs.

The resulting schedule is a **MJF** schedule, as, on each resource M_k, all the local jobs start before $C^*_{\max}(M_k)$, which is the earliest start time for foreign jobs. Moreover, on each resource the resulting makespan is at most $2C^*_{\max}(M_k)$, thus the global makespan is also at most $2C^*_{\max}$. □

Let us now assume that each organization O_k is able to send any local job $J^i_k \in \mathcal{J}_k$ to a non-local resource M_l. After migrating to M_l, J^i_k is scheduled by the owner O_l of the host resource M_l. In such a game, the strategy of an organization consists of two parts: deciding which local jobs should be migrated (migration strategy); and scheduling foreign jobs on the local resource (scheduling strategy).

Definition 5.2. *The Parallel Scheduling Game with Migration (PSGM) is a strategic game in which:*

- *the set of players is equal to the set of organizations $\mathcal{P} = \mathcal{O}$;*
- *a strategy of O_k is a tuple (π_k, μ_k) where π_k is the schedule of jobs $\mathcal{I}_k$ on M_k and $\mu_k : \mathcal{J}_k \to \mathcal{M}$, the migration function, maps each local job to a machine where this job will be executed. The set of strategies of player O_k contains all possible tuples;*
- *the payoff of player O_k is the maximum completion time $C_{\max}(O_k)$ of O_k's jobs $\mathcal{J}_k$.*

Any *rational migration strategy* migrates O_k's job if, given schedules and scheduling strategies of other players, $C_{\max}(O_k)$ is not increased after this operation.

The problem of determining optimal migration strategy is NP-hard. By restricting the number of organizations to $N = 2$, the number of processors on each resource to $m = 1$ and the number of jobs produced by one of the organizations, $|\mathcal{J}_2| = 0$, we obtain the problem of minimization of the total completion time of sequential jobs on two resources, $P2||C_{\max}$, which is NP-hard.

An example rational migration strategy is the greedy migration strategy, an equivalent of the conservative backfilling algorithm. Let us assume that the other players use MJF as the strategy for scheduling foreign jobs. Greedy migration strategy analyzes all the areas of free processors in the schedules on other resources. If there are m' free processors in period $[t', t'')$ (at the end of the schedule, it is possible that $t'' \to \infty$), the algorithm migrates the first local job J_k^i starting at or after t' that requires $q_k^i \leq m'$ during $p_k^i \leq (t'' - t')$. As all the migrated jobs start earlier, greedy migration is able to reduce the local makespan, and thus is a rational migration strategy.

Proposition 5.9. MJF *and any rational migration strategy is the Nash equilibrium of PSGM.*

Proof: The proof is analogous to the proof of Proposition 5.6. For any organization O_k, given any scheduling on non-local resources, the strategy minimizing $C_{\max}(O_k)$ is a strategy that schedules the foreign jobs so that they do not delay any local job (MJF strategy). Regarding migration, the best O_k can do is to try to fill the gaps in other schedules by its jobs (rational migration), if the gaps occur before O_k's local makespan. □

5.5 Multi-Organization Scheduling

The *Multi-Organization Scheduling Problem* (MOSP) is the minimization of the makespan of all the jobs (the moment when the last job finishes) with an additional constraint that no makespan is increased compared to a preliminary schedule in which all the resources compute only locally produced jobs. More formally, let us denote $C_{\max}^{loc}(O_k)$ as a makespan of O_k when $\mathcal{J}_k$, the set of jobs executed by M_k is equal to the set of locally produced jobs, i.e. $\mathcal{J}_k = \mathcal{I}_k$. MOSP can be defined as:

$$\min C_{\max} \text{ such that } \forall_k \; C_{\max}(O_k) \leq C_{\max}^{loc}(O_k). \tag{5.1}$$

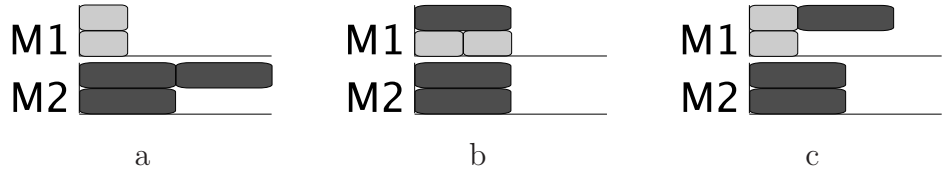


Figure 5.8: Globally-optimal solution (b) is inadmissible, as it extends the makespan of organization O_1 (the producer of light gray jobs) in comparison with the local solution (a). The best solution not extending O_1's makespan (c) is $\frac{3}{2}$ from the global optimum.

5.5.1 Impact of the Constraint

Because of the constraint of not worsening any local makespan, some globally-optimal solutions are not feasible. The following proposition formally states this results.

Proposition 5.10. *Any scheduling algorithm solving MOSP has an approximation ratio of at least $\frac{3}{2}$ regarding the optimal solution of the unconstrained minimization of system-wide $C_{\max}$.*

Proof: Consider an instance in Figure 5.8. The best solution not worsening O_1 makespan, depicted in (c), has $C_{\max} = 3$, whereas the global optimum has $C_{\max} = 2$. □

5.5.2 Multi-Organization Load Balancing Algorithm

We can use load-balancing techniques similar to these presented in the previous chapter to optimize the system-wide makespan, at the same time not worsening makespans of individual organizations. The algorithm presented in this section guarantees the approximation ratio of the global makespan. The algorithm presented in the following section (Section 5.5.3) further improves both the global makespan and the equity between organization's makespans.

Algorithm 5.2: Backfill List Scheduling (BLS) algorithm. The algorithm schedules jobs $\mathbf{J}$ on $\mathbf{M}$ without rescheduling any job already scheduled on $\mathbf{M}$. `procFree`$(t,\ M_l)$ denotes the function returning set of free processors from machine M_l.

Input:

- $\mathbf{J} = (J^1, \ldots, J^n)$, ordered list of jobs;
- $\mathbf{M}$ set of machines, possibly with other jobs scheduled;

1 $unscheduled = \mathbf{J}$;
2 $t = \min_{i:J^i \text{ scheduled on } \mathbf{M}} C^i$;
3 **while** *not empty(unscheduled)* **do**
4 **foreach** $M_l : \exists J^i : J^i \in \mathcal{I}_l \wedge C^i = t$ **do**
5 $procs =$`procFree(`$t,\ M_l$`)` ;
6 **foreach** $J^i \in unscheduled$ **do**
7 **if** $q^i \leq |procs|$ **then**
8 $t' = t$;
9 **while** $((t' = \min_{j:J^j \in \mathcal{I}_l \wedge C^j > t'}) < t + p^i) \wedge (t' < C_{\max}(M_l))$ **do**
10 $procs \cap =$`procFree(`$t,\ M_l$`)` ;
11 **if** $q^i \leq |procs|$ **then**
12 `schedule(`$J^i,\ procs,\ t$`)` ;
13 $unscheduled.remove(J^i)$;
14 $procs =$`procFree(`$t,\ M_l$`)` ;
15 $t = \min_{i:J^i \text{ scheduled on } \mathbf{M} \wedge C^i > t} C^i$;

Algorithm 5.3: Multi-Organizational Load Balancing Algorithm (MOLBA).

Input: $\mathcal{J}_k$ sets of jobs produced by each organization

1 **for** $k = 1$ **to** N **do**
2 $\quad \mathbf{J}_k =$`sort(`$\mathcal{J}_k$, $q^i \searrow$`)` ;
3 $\quad$ `LS(`$\mathbf{J}_k$, M_k`)` ;
4 $\mathcal{O}' = \emptyset; \mathcal{J}' = \emptyset; \mathbf{M}' = \emptyset$;
5 **for** $k = 1$ **to** N **do**
6 $\quad$ **if** $C^{loc}_{\max}(O_k) > (\alpha\bar{W} + p_{\max})$ **then**
7 $\quad\quad \mathcal{O}'.add(O_k)$;
8 $\quad\quad \mathcal{J}' \cup = \mathcal{J}_k$;
9 $\quad\quad \mathbf{M}'.add(M_k)$;
10 unschedule($\mathcal{J}'$) ;
11 $\mathbf{J}' =$`sort(`$\mathcal{J}'$, $q^i \searrow$`)` ;
12 `BLS(`$\mathbf{J}', \mathbf{M}'$`)` ;
13 $\mathcal{J}'' = \emptyset$;
14 **foreach** $J^i \in \mathcal{J}'$ **do**
15 $\quad$ **if** $J^i.start > \alpha\bar{W}$ **then** $\mathcal{J}''.add(J^i)$;
16 unschedule($\mathcal{J}''$) ;
17 `BLS(`$\mathcal{J}'', \mathbf{M}$`)` ;

Algorithm

Multi-Organizational Load Balancing Algorithm (MOLBA), denoted by MOLBA(α) ($\alpha \geq 1$), computes a lower bound of the global makespan, selects all the organizations whose makespan is larger than $(\alpha + 1)$ times the lower bound and then reschedules some of their jobs onto the less loaded resources. α is a parameter of the algorithm which expresses the trade-off between the global efficiency and the constraints on local makespans (see the proofs in the following sections). Later on, we will refer to the algorithm with a particular value of α as MOLBA(α), e.g. MOLBA(2). MOLBA is described as Algorithm 5.3 and uses Backfilling List Scheduling described as Algorithm 5.2.

MOLBA starts with scheduling jobs $\mathcal{J}_k$ on local resource M_k with LS in Highest-First order (i.e. non-increasing number of required processors) in lines 1-3. Resulting $C_{\max}(O_k)$ form the constraint for the rest of the algorithm.

Then, the algorithm computes the subset $\mathcal{O}'$ of all organizations that have local makespan $C_{\max}(O_k) \geq \alpha\bar{W} + p_{\max}$ (lines 4-9). Jobs of these organizations are unscheduled, mixed and list-scheduled on resources belonging to $\mathcal{O}'$ in HF order (lines 10-12). Note that in this case, BLS works as LS on multiple resources, as no job is scheduled on $\mathbf{M}'$ before the start of the algorithm.

Next, jobs belonging to $\mathcal{O}'$ that start after $\alpha\bar{W}$ are removed from local resources and rescheduled by BLS algorithm on all resources (lines 13-17). The jobs are scheduled sequentially in a greedy manner by BLS (Algorithm 5.2). A job J^i is scheduled on a resource M_l in time t if there are then q^i free processors (Algorithm 5.2, line 7). However, no migrated job can delay a local job: a job J^i is scheduled before the original makespan of the host resource M_l ($t < C^{loc}_{\max}(O_l)$) only if at least q^i_k processors are free on M_l from time t to time $t + p^i_k$ (Algorithm 5.2, line 9). Such a strategy is similar to the well-known conservative backfilling in FCFS (First Come First Serve)

[Lifka, 1995]. Therefore, a job incoming to one of resources that do not belong to $\mathcal{O}'$, is scheduled either in a gap, or at the end of the schedule.

Computational Complexity

The basic component of MOLBA(α) is (B)LS algorithm, performed firstly for local scheduling, and then in rescheduling jobs starting after $\alpha\bar{W}$. LS, at each time moment when utilization changes, examines the list of not executed jobs in order to find a job that can fit into the available processors. As utilization changes whenever a job is started or finished, there are at most $O(n)$ such time moments. The list of not executed jobs contains at most n jobs. Consequently, the complexity of LS is $O(n^2)$. MOLBA(α) repeats basic LS two times, thus the complexity remains the same ($O(n^2)$).

Analysis of Approximation Ratio

As no local job is delayed, MOLBA(α) guarantees that no organization O_k which hosts migrated jobs has its makespan extended beyond $C^{loc}_{\max}(O_k)$. We now prove that the algorithm reduces $C_{\max}(O_k)$ for the rest of the organizations (therefore holding the constraint in Eq. 5.1), that MOLBA(2) is a $3-$approximation of the global makespan $C_{\max}$ if the last completed job is *low*, and finally, that MOLBA(3) is a $4-$approximation in the general case. We start with a lemma that characterizes the structure of all the resources' schedules.

In the schedule returned by MOLBA, on each resource, we denote by t_L^{start} the first moment when the utilization is lower than or equal to $\frac{1}{2}$. Similarly, t_L^{end} is the last moment when the utilization is larger than 0 and lower than or equal to $\frac{1}{2}$.

Lemma 5.2. *In the schedule returned by MOLBA(α), on each resource, the length of the time interval between t_L^{start} and t_L^{end} (denoted by P_L) is shorter than or equal to $p_{\max}$.*

Proof: Each resource schedules its local jobs with HF (i.e. according to non-increasing required number of processors). Then, it may add jobs from other organizations, also in HF order. Proposition 5.1 shows that, in a schedule returned by HF, the only zone of low utilization is at the end of the schedule. Thus, on each resource, there are at most two zones of low utilization: possibly one at the end of the schedule of the local jobs, and also possibly one at the end of the schedule.

Let $J_{k,i}$ be the *low* job that finishes last on resource M_j. After $J_{k,i}$ finishes, utilization is either high, or zero. Thus, by P_L definition, $J_{k,i}$ cannot finish before t_L^{end}. $J_{k,i}$ does not start after t_L^{start}, as utilization at t_L^{start} is low, so there are enough free processors to execute a *low* job. Thus, the length of P_L is smaller than or equal to the length of $J_{k,i}$, which is not longer than p_{max}. □

The following proposition enables us to place t_L^{start}, the moment of transition between the first two zones.

Proposition 5.11. *After MOLBA finishes, there is at least one resource whose* $t_L^{start} \leq 2\bar{W}$.

Proof: The proof is by contradiction. Suppose that there exists $\epsilon > 0$ such that all the resources have high utilization until time $2\bar{W} + \epsilon$. Then, the total surface of jobs computed by all the resources is greater than $2\bar{W} \cdot mN \cdot 0.5 = W$, i.e. greater than the total work available, which leads to a contradiction. □

In a special case, when the last completed job is *low*, MOLBA(2) is a 3-approximation of the optimal makespan. The following proposition formally states this result.

Proposition 5.12. *If the last completed job is* low*, MOLBA*(2) *is a* 3*-approximation of* $C^*_{\max}$*. Moreover, all the organizations have incentive to cooperate.*

Proof: Let us assume that J_k^i is the last completed job. By Proposition 5.11, there is at least one resource M_l whose $t_L^{start} \leq 2\bar{W}$. Consequently, if J_k^i started after $2\bar{W}$, scheduling it on M_l would reduce J_k^i's completion time. Thus, J_k^i starts at $2\bar{W}$ at the latest (as we use a greedy algorithm to schedule migrated jobs). J_k^i finishes thus before $2\bar{W} + p_k^i \leq 2\bar{W} + p_{\max} \leq 3C_{\max}^*$.

As no migrated job can delay a local job, makespans of organizations that were receiving jobs are not modified. The organizations that were sending jobs have their makespan reduced because of the global approximation ratio. The schedule of the rest of organizations is not modified. Thus, the constraint in Equation (5.1) is satisfied and all the organizations have incentive to cooperate. □

In the general case however, MOLBA(3) is a 4-approximation.

Proposition 5.13. *MOLBA*(3) *is a* 4*-approximation of* $C_{\max}^*$*. Moreover, all the organizations have incentive to cooperate.*

Proof: The proof is by contradiction. Let C_{max}^* be the makespan of the optimal schedule, and let us suppose that a job starts after time $3\,C_{max}^*$ in the schedule returned by MOLBA(3). This means that this job could not have been started before, so all the resources are busy until $3\,C_{max}^*$, i.e. for all $k \in \{1, \ldots, n\}$, $C_{max}(M_k) > 3\,C_{max}^*$. Proposition 5.2 shows that, for each resource, the zone where at most half of the processors are busy is smaller than or equal to $p_{max} \geq C_{max}^*$. Thus, on each resource, the zone where more than the half of the processors are busy is larger than $2\,C_{max}^*$. As we have seen it previously, $C_{max}^* \geq \frac{W}{N\,m}$, where $W = \sum p_k^i q_k^i$. Thus, the total work which is done until $3\,C_{max}^*$ in the zones of high utilization is larger than $\frac{N\,m}{2}(2\,\frac{W}{N\,m}) = W$, i.e. larger than the total surface of the tasks available. As tasks are not repeated, this is not possible. Thus, no job starts after $3\,C_{max}^*$, and no job is completed after $4\,C_{max}^*$.

The proof that all the organizations have incentive to cooperate is analogous to the proof of Proposition 5.12. □

5.5.3 Iterative Load Balancing Algorithm (ILBA)

MOLBA(3) guarantees that the global makespan $C_{\max}$ is not higher than $4C^*_{\max}$. Nevertheless, the load distribution resulting from running the algorithm can be uneven. For instance, MOLBA(3) does not modify the schedule of an organization with $C_{\max}(O_k) = 4C^*_{\max} - \epsilon$. Yet, it is possible that jobs on M_k start after all the other resources finish. The Iterative Load Balancing Algorithm (ILBA) presented in this section improves the schedule returned by MOLBA(α) by balancing the load between resources. ILBA does not delay any job, thus, given a schedule produced by MOLBA(α), ILBA fulfills the constraint in Eq. 5.1 and does not worsen the approximation ratio. We also show that in a restricted case of scheduling sequential jobs, ILBA has the same approximation ratio as any list-scheduling algorithm.

By starting with the least-loaded resources and iteratively adding the more loaded ones, ILBA permits also to reduce the makespans on the less loaded resources.

Note that, as ILBA does not take into account the owner of a job, in this section we will omit the index of organization, similarly to Section 5.2.

Algorithm

ILBA is described as Algorithm 5.4. Resources are sorted by non-decreasing makespans. Suppose that $C_{\max}(M_1) \leq C_{\max}(M_2) \leq \cdots \leq C_{\max}(M_N)$. The algorithm, for all $k = 2, \ldots, N$, reschedules all the jobs executed on M_k on $\{M_1, \ldots, M_k\}$. Jobs are scheduled sequentially in order of their in the order of execution of the original schedule. Each job J^i is allocated to the resource that has an earliest strip of free processors of width at least p^i and height at least q^i. Thus, similarly to the previous stage of the algorithm, a job incoming from M_k cannot delay any job already scheduled on $\{M_1, \ldots, M_k\}$.

Algorithm 5.4: Iterative Load Balancing Algorithm (ILBA).

Input: $\mathbf{M} = [M_1, \ldots, M_N]$ machines, without loss of generality we assume that $C_{max}(M_1) \leq C_{max}(M_2) \leq \ldots C_{max}(M_N)$

```
1 M' = [M_1] ;
2 for k = 2 to N do
3     J = sort(I_k, start times ↗) ;
4     foreach J^i ∈ J do
5         unschedule(J^i) ;
6     M'.add[M_k] ;
7     foreach J^i ∈ J do
8         BLS(J^i, M') ;
```

Computational Complexity

ILBA reschedules each job of all, but first, organizations. For each organization, at most n jobs are rescheduled, which involves examining the schedules on the rest of machines, that, in total, have at most n events. Consequently, the complexity of the whole algorithm is in $O(N \log N + Nn^2)$.

Analysis of Approximation Ratio

We start with proving that ILBA does not delay any job, and thus can be used after MOLBA without deteriorating its approximation ratio. Then, we prove that when all the jobs are sequential, or all the jobs are high, ILBA can be used directly after local scheduling with an approximation ratio of, accordingly, $2 - \frac{1}{Nm}$ and $2 - \frac{1}{N}$.

Proposition 5.14. *ILBA does not delay any job comparing with the base schedule.*

Proof: As ILBA considers resources sequentially, it is sufficient to show that the proposition holds for any resource M_k (as the jobs from the following resources do not modify the jobs already scheduled).

Let us assume that jobs are numbered according to non-decreasing start times in the base schedule.

The proof is by induction on the scheduled jobs. In ILBA, a scheduled job J^i cannot be influenced by jobs $J^{i+1}, \ldots, J^n$ scheduled afterwards (nor by jobs incoming from different resources in the subsequent parts of the algorithm). Note also that at this phase of the algorithm no new job is allocated to M_k. Thus, it is sufficient to show that if jobs $J^1, \ldots, J^{i-1}$ are completed no later than in the original schedule, J^i is also not delayed.

The proposition trivally holds for J^1, as the first job will be started at $t = 0$.

Let us now assume that jobs $J^1, \ldots, J^{i-1}$ are scheduled no later than their start times in the original schedule. Consequently, $J^1, \ldots, J^{i-1}$ finish no later than in the original schedule. Thus, at J^i's original start time, the number of free processors is at least the same as in the original schedule. Hence, J^i can be scheduled at its original start time. J^i is migrated only if it can be started earlier than this orginal start time. Consequently, J^i finishes at latest at time when it finished in the base schedule. □

Furthermore, in some special cases, ILBA can be run directly after running local scheduling (i.e. executing MOLBA(α) is not needed). In the following parts, we will prove approximation ratios in two special cases. If all the jobs are sequential, ILBA is a $(2 - \frac{1}{N\,m})$-approximation of the optimal C^*_{max}. If all the jobs are high, ILBA is a $(2 - \frac{1}{N})$-approximation..

Proposition 5.15. *If all the jobs are sequential (i.e. $q^i_k = 1$ for all i, k), then ILBA is a $(2 - \frac{1}{N\,m})$-approximate algorithm where all the organizations have incentive to cooperate.*

Proof: In the obtained schedule, no job is delayed comparing with the local schedule (Proposition 5.14), thus no organization has its makespan increased. Consequently, all the organizations have incentives to cooperate.

In ILBA schedule, no job starts after any processor (of any resource) becomes idle. Thus, the schedule could have been returned by a list scheduling

algorithm. The approximation ratio of a list algorithm which schedules jobs on p processors is $2 - \frac{1}{p}$ [Graham, 1969]. Thus, ILBA also has the approximation ratio of $2 - \frac{1}{N\,m}$. □

The proof in the special case in which all the jobs in the system are high, i.e. require more than the half of the processors ($q_k^i > \frac{m}{2}$) is similar. If all the jobs are high, it is not possible to schedule two jobs at the same time on the same resource. Consequently, the scheduling problem can be considered as scheduling sequential jobs on N parallel processors. Each resource corresponds to a processor, and each job J_k^i corresponds to a sequential job of length p_k^i.

Proposition 5.16. *ILBA is a* $(2-\frac{1}{N})$*-approximation of the global makespan* $C_{\max}$ *if all the jobs are* high *(i.e.* $q_k^i > \frac{m}{2}$ *for all* i, k*).*

Proof: The proof is similar to the proof of Proposition 5.15. In the obtained schedule, no job is delayed comparing with the local schedule (Proposition 5.14), thus no organization has its makespan increased. Consequently, all the organizations have incentive to cooperate.

Observe that, if all the jobs are high, the schedule returned by ILBA can be obtained by running a list scheduling algorithm, as no job starts after any resource becomes idle. Consequently, the approximation ratio of ILBA is the same as the approximation ratio of the LS, i.e. $(2 - \frac{1}{N})$ [Graham, 1969]. □

5.6 Experiments

In this section, we carry out the experimental evaluation of MOLBA. We start with performance measures and the methods used to generate workload. Then, we show the performance of the considered algorithms.

5.6.1 Methods

We compared three algorithms:

local that, for each organization, schedules its jobs on its local resource with a list scheduling algorithm;

MOLBA the algorithm presented in Section 5.5.2 as MOLBA(2), that, however, does not perform the final load balancing of ILBA (Algorithm 5.4);

MOLBA+ILBA the complete algorithm, i.e. ILBA (Section 5.5.3) ran on schedules returned by MOLBA(2).

Note that MOLBA(2) is an approximation algorithm only when the last job is *low*; the theoretical analysis in the previous section showed that, in the general case, MOLBA(3) should be used. In our experiments, we were using the following meta-algorithm. Firstly, we run MOLBA(2). Then, if in the resulting schedule any organization had its makespan increased, or the global makespan was greater than $3\max(p_{max}, \bar{W})$, the schedule was discarded and MOLBA(3) was run. However, during our experiments, those conditions were never fulfilled. Thus, all the presented results were obtained with MOLBA(2).

The algorithms were compared regarding the system-wide makespan that was produced. As the instances differed e.g. in the number of jobs, or their sizes, for each instance and each algorithm we computed the factor of the produced makespan to the lower bounds defined in Section 5.3.3, i.e.

$$s(alg, I) = \frac{makespan(alg, I)}{\max(\bar{W}(I), \max p_k^i(I))},$$

where $s(alg, I)$ is the algorithm alg score on instance I, $makespan(alg, I)$ is the makespan of the schedule produced by algorithm alg on instance I, $\bar{W}(I)$ is the average work in the instance and $\max p_k^i(I)$ is the length of the longest job in the instance.

We were testing the algorithms on two sets of randomly-generated instances:

- *uni* instances in which the sizes p_k^i and the numbers of required processors q_k^i of jobs were produced by two independent, uniform distributions over, respectively, $\{1, \ldots, 50\}$ and $\{1, \ldots, m\}$;

- *swf* instances produced by a realistic workload generator [Lublin and Feitelson, 2003]. We adjusted the generator as follows. The maximum required number of processors is smaller than m. The number of generated jobs is set to n. All the release dates are set to 0.

Each instance produced consisted of n (a parameter of the experiments) jobs. To determine the owner O_k of each job, we used Zipf distribution with number of elements equal to the total number of organizations (N) and the exponent of $s = 1.4267$. This parameters corresponded to the distribution of Virtual Organizations owning the jobs in data analyzed in [Iosup et al., 2006].

Both *uni* and *swf* instances were generated with the total number of organizations $N \in \{2, 5, 10, 20\}$, the total number of jobs $n \in \{10, 50, 100, 500\}$ and the number of processors in each resource $m \in \{32, 128, 512\}$. N was chosen to represent academic grids, where the number of participating laboratories is about 10. The number of processors varied between small and large clusters currently used in grid systems. For instance, clusters in Grid'5000 [Bolze et al., 2006] currently have between 32 and 342 nodes.

For each possible values of n, N and m, 50 instances were generated. In total, 4800 instances were generated.

5.6.2 Results

Results of experiments are presented in Table 5.1 (*uni* dataset) and in Table 5.2 (*swf* dataset).

Firstly, the performance of all algorithms is much better on the instances created with realistic workload generator (*swf* dataset). Because of the increased heterogeneity of both the sizes of jobs and the number of required processors, these instances simply schedule better. In *swf* dataset, MOLBA+ILBA algorithm was optimal in all smallest instances considered

Table 5.1: Scores s, or ratios of makespan to lower bound, of local, MOLBA and MOLBA+ILBA and their standard deviations σ. Each row is an aggregation over 150 random instances. *uni* dataset.

		local		MOLBA		MOLBA+ILBA	
N	n	s	σ	s	σ	s	σ
2	average	1.64	0.19	1.64	0.19	1.19	0.06
	10	1.85	0.38	1.85	0.38	1.42	0.17
	50	1.63	0.20	1.63	0.19	1.17	0.04
	100	1.59	0.13	1.59	0.13	1.12	0.03
	500	1.50	0.06	1.50	0.06	1.05	0.02
5	average	2.74	0.40	2.03	0.14	1.21	0.09
	10	1.97	0.53	1.84	0.42	1.29	0.26
	50	3.11	0.52	2.17	0.09	1.29	0.06
	100	3.04	0.39	2.09	0.05	1.19	0.03
	500	2.85	0.17	2.02	0.01	1.08	0.01
10	average	4.19	0.65	2.08	0.18	1.27	0.10
	10	1.98	0.62	1.76	0.40	1.19	0.22
	50	4.69	0.96	2.32	0.18	1.46	0.10
	100	5.10	0.71	2.20	0.10	1.31	0.05
	500	5.01	0.29	2.04	0.02	1.12	0.02
20	average	5.96	0.97	2.10	0.21	1.32	0.11
	10	1.78	0.71	1.53	0.44	1.13	0.20
	50	5.72	1.20	2.44	0.23	1.55	0.12
	100	7.29	1.30	2.35	0.13	1.43	0.08
	500	9.05	0.69	2.10	0.03	1.18	0.02
average		3.64	2.31	1.96	0.37	1.25	0.19

Table 5.2: Scores s, or ratios of makespan to lower bound, of local, MOLBA and MOLBA+ILBA and their standard deviations σ. Each row is an aggregation over 150 random instances. *swf* dataset.

		local		MOLBA		MOLBA+ILBA	
N	n	s	σ	s	σ	s	σ
2	average	1.22	0.28	1.22	0.28	1.09	0.11
	10	1.03	0.16	1.03	0.16	1.00	0.00
	50	1.10	0.27	1.10	0.27	1.02	0.08
	100	1.21	0.33	1.21	0.33	1.07	0.15
	500	1.53	0.36	1.53	0.36	1.26	0.20
5	average	1.17	0.29	1.10	0.18	1.02	0.05
	10	1.02	0.14	1.01	0.08	1.00	0.00
	50	1.08	0.24	1.02	0.08	1.01	0.04
	100	1.14	0.28	1.07	0.17	1.01	0.05
	500	1.46	0.47	1.32	0.38	1.05	0.11
10	average	1.15	0.27	1.03	0.06	1.00	0.01
	10	1.01	0.08	1.00	0.02	1.00	0.00
	50	1.06	0.22	1.00	0.03	1.00	0.00
	100	1.10	0.22	1.01	0.04	1.00	0.00
	500	1.44	0.55	1.09	0.16	1.01	0.05
20	average	1.12	0.26	1.01	0.03	1.00	0.00
	10	1.02	0.12	1.00	0.00	1.00	0.00
	50	1.05	0.18	1.00	0.01	1.00	0.00
	100	1.10	0.24	1.00	0.01	1.00	0.00
	500	1.32	0.48	1.03	0.08	1.00	0.02
average		1.17	0.35	1.09	0.23	1.03	0.10

(with $n = 10$), in which the makespan was determined by the length of the longest job.

The inefficiency of *local* scheduling can be observed in *uni* instances. The average score of *local* algorithm is deteriorating quickly with the increased number of organizations for all but the smallest instances. For larger instances (with $n \in \{50, 100, 500\}$) , the average score grows from 1.57 ($N = 2$), through 3.00 ($N = 2$) and 4.93 ($N = 5$), till 7.35 ($N = 20$). The linear correlation coefficient between the N and the score of *local* is equal to 0.69 for *uni* (and 0.38 for the both datasets combined). The scores of MOLBA and MOLBA+ILBA also deteriorate, but not as quickly (the correlation coefficients are 0.36 and 0.27, accordingly). The degradation of *local* results indicates that the lack of cooperation negatively influences the performance of the system.

The load balancing performed by ILBA considerably improves the results. The average score in *uni* has been improved from 1.96 to 1.25. In *swf*, the improvement is not as significant, but nevertheless important, especially in smaller grids.

The number of processors on each node m does not influence the performance of the considered algorithms. After averaging over all the other variables, p-values of two-tailed, type 3 t-test, for $m = 32$ and $m = 128$ are 0.35, 0.17 and 0.53, for, respectively, *local*, MOLBA and MOLBA+ILBA scores. Similarly, for the difference between $m = 128$ and $m = 512$, we have 0.56, 0.62 and 0.35. Consequently, we do not present results averaged over different values of m.

The worst result for *local* algorithm was 12.12, for an *uni* instance with $N = 20$ organizations and $n = 100$ jobs. The makespan is determined by a number of relatively short jobs executed on one resource. The worst result of MOLBA was 2.98 for an *uni* instance with $N = 20$ and $n = 50$. In this instance, the makespan is determined by an organization that does not take part in the MOLBA algorithm, as its local makespan is between $2\bar{W}$ and $2\bar{W} + p_{\max}$. The worst result of MOLBA+ILBA is 1.92 (also for $N = 20$

and $n = 50$), which is significantly lower that the theoretical upper bound of 3. In this instance, the last finished job is a long, *high* job, started after a period of low utilization.

The results of our experiments show that MOLBA+ILBA is an efficient algorithm that considerably improves the performance in comparison with local scheduling. However, we also observed that the workloads generated by the realistic workload generator were easier to schedule, and thus the improvement by MOLBA+ILBA was not as considerable.

5.7 Summary

In this chapter, we applied the multi-organization grid model to extend one of the classic problems of the scheduling theory. More specifically, we studied a model of the grid in which selfish organizations minimize the maximum completion time of locally-produced jobs. We proposed a way of ordering jobs which achieves homogeneous utilization (Section 5.2.2). Then, we proved that the performance of classic, list scheduling algorithm is degradated when the algorithm is scheduling jobs on many resources (Section 5.3.3). We demonstrated that using list scheduling it is impossible to prefer certain jobs in order to realize equitable optimization (Section 5.3.4). Using game theory, we proved that the resulting game has a low price of anarchy (at most 2), as we suppose that the idle time of machines is free (Section 5.4). Finally, we demonstrated that it is always possible to respect the selfish goals and improve the performance of the whole system at the same time. The cooperative solutions produced by proposed algorithm (Section 5.5) have a constant worst case performance (4 in the general case and 3, $2-\frac{1}{Nm}$ or $2-\frac{1}{N}$ in some special cases), a significant gain compared to selfish solutions that can be arbitrary far from the optimum. Although the theoretical worst-case performance can be considered high, we showed, through experimental evaluation (Section 5.6), that the performance on randomly-generated instances is significantly better.

Chapter 6

Summary and Conclusions

In this thesis we proposed a framework model of a system that gathers independent individuals (Chapter 2). Each individual grants some resources and, in return, wants to use a part of the whole system. The resulting model can be analyzed by mathematical tools such as game theory or equitable optimization. Moreover, the model is generic enough to be adapted to a specific system, without loosing the possibility of being analyzed. We presented three applications of the model to the various types of computational grids. Additionally, it should be possible to adapt the model to other types of modern systems, such as peer-to-peer systems or web 2.0 communities.

One of the main results of this research is a demonstration that the complete organizational decentralization of a system is costly in terms of performance. In such an organizationally-decentralized system, individual organizations maintain complete control over their resources. We analyzed such systems with game-theoretic approach. In resource sharing grids, the worst-case loss of performance was linear on the number of jobs. In divisible load model, load balancing between resources was never performed. Only in one of three models considered, the loss of performance was acceptable. However, this model required some strong assumptions.

We also demonstrated that it is possible to respect goals of individual members and to keep satisfactory performance of the whole system at the

same time. To this end, the members are obliged to grant at least partial control over their resources to a central entity. Constraints of the optimization problem guarantee that the goals of the members are respected and realized.

We applied equitable multicriteria optimization to guarantee fairness of the results regarding individual participants. We characterized properties of a fair solution through axioms of anonymity, monotony and the principle of transfers. Equitable optimization turned out to be a convenient tool, as axiomatic fairness is able to balance the performance of the whole system with the fairness of individual participants. It was not always possible, however, to formally guarantee the fairness of results returned by algorithms, because of either the lack of information (in on-line models) or the lack of algorithmic means (in parallel job scheduling).

Analyzing the computational grid that shared dedicated resources (Chapter 3) we demonstrated that organizational decentralization causes significant loss of performance even in an off-line model. Nevertheless, if organizations grant control over the local schedules to the centralized scheduler, an efficient and fair solution can be produced.

Computational grid that balanced divisible load across resources (Chapter 4) was also sensitive to the lack of centralized control. In decentralized systems, load balancing is never performed, even though it would guarantee an equitable load distribution. We managed to balance the load of the system by obliging organizations to commit to their decisions (to balance the load or not) for some period, regardless of their queue lengths. Moreover, we proposed an iterative load balancing algorithm that distributes the gains from load balancing in more equitable manner than the classic, averaging algorithm.

We also considered the problem of scheduling parallel jobs on multiple multiprocessor resources (Chapter 5). Here, the decentralization did not cause a drop of performance of the whole system. The model is off-line, thus a foreign job executed at the end of the schedule does not worsen the performance of the resource's owner. At the same time, the owner of the

job can reduce its makespan. We proposed a scheduling algorithm that has a fixed worst-case performance on the system makespan and that do not increase the makespans of individual organizations.

The results presented in this work required many assumptions. For instance, we assumed that the scheduler possesses information about the exact size of each job (clairvoyance) or that all the jobs in the system are known and ready to be executed before the scheduling algorithm starts (off-line). Such assumptions are common in scheduling theory, as they allow to derive some precise mathematical results on the simplified models. Clearly, at the same, such assumptions simplify complex reality, so algorithms in the form presented in this work cannot be directly implemented in grid scheduling software. However, the algorithms that turned out to be efficient on simplified models can be ported in future to the real environments. In this case, clairvoyance would be replaced with user run-time estimates. Jobs that are produced during the lifetime of the system (on-line setting) would be considered in batches, inside which an off-line algorithm can be run [Shmoys et al., 1995]. Similarly, algorithms for scheduling divisible load could be easily adapted to bag-of-tasks applications, by fixing the minimal size of the grain.

This theorized approach allowed us to derive some precise, mathematical results on the performance of the considered systems and algorithms. We modeled what we consider the essence of the grid. Afterward, we used this simplified model to validate our algorithms. The alternative approach would be to build a much more realistic model of a grid and then to evaluate the algorithms by simulation. We claim that it is not possible to prepare an accurate simulation as grids are considerably complex systems and, thus, they deal with phenomena spanning through almost every computer science discipline. A realistic grid simulator would model the reliability of nodes, sites and interconnection network; network links, along with the background load; applications with their usage patterns for computational power, network and other resources; users that submit applications based on the observed

load of the system; etc. Clearly, it is not possible to build accurate models for all these phenomena, specifically as they additionally interact with each other. Moreover, at this stage of development of grids, it is not yet clear which phenomena are crucial, and which can be omitted without loosing the realism of simulation. Thus, one cannot have realistic expectations on the performance of a scheduling algorithm evaluated solely through simulation.

The main conclusion of this study is that grids without any form of centralized control or coordination work inefficiently. The resulting loss of performance can be proportional to the system usage. Yet, with some level of coordination, it is possible to share the pool of available resources fairly amongst participants, so that no-one loses by cooperating.

Chapter 7

Résumé étendu en français

Ce texte est un résumé étendu écrit en français du document complet de thèse. Le lecteur intéressé pourra se référer aux articles publiés suivants:

- Rzadca K., Trystram D., *Promoting Cooperation in Selfish Computational Grids*, European Journal of Operational Research, Special Issue on Cooperative Combinatorial Optimization, Elsevier (accepted for publication, April 2007)

- Rzadca K., *Scheduling in Multi-Organization Grids: Measuring the Inefficiency of Decentralization*, PPAM 2007 (International Conference on Parallel Processing and Applied Mathematics) Proceedings, LNCS, Springer (in print)

- Pascual F., Rzadca K., Trystram D., *Cooperation in Multi-Organization Scheduling*, Euro-Par 2007 Conference Proceedings, LNCS 4641, Springer 2007, pp. 224–233

- Rzadca K., Trystram D., Wierzbicki A., *Fair Game-Theoretic Resource Management in Dedicated Grids*, IEEE CCGrid 2007 (International Symposium on Cluster Computing and the Grid) Proceedings, IEEE Computer Society, pp. 343-350

- Rzadca K., Trystram D. *Brief Announcement: Promoting Cooperation in Selfish Grids*, SPAA 2006 (Annual ACM Symposium on Parallelism in Algorithms & Architectures) Proceedings, ACM Press 2006, pp. 332.

7.1 Introduction

Les grappes et grilles de calcul sont des super-ordinateurs à grande échelle qui permettent l'utilisation coordonnée de ressources possédées et contrôlées par différentes organisations. Les grappes introduisent de nouvelles questions au problème de l'ordonnancement et de la gestion des ressources. Dans le même temps, un grand nombre d'applications importantes a besoin d'une puissance du calcul qui, aujourd'hui peut être fournie que par des grilles.

Les grappes et grilles sont un exemple d'un système informatique moderne fondée sur le partage. Ainsi, les études, méthodes et résultats menés dans cette thèse peuvent être adaptés à des domaines connexes, tels que les systèmes pair à pair, et dans une certyaine mesure, les systèmes embarqués.

Le problème de la gestion des ressources dans les grappes est difficile. Même au niveau d'un seul super-ordinateur, la performance dépend de la gestion efficace de ses ressources, et en particulier de l'ordonnancement des tâches qu'il exécute. La plupart de ces problèmes sont des problèmes algorithmiques difficiles. Une grappe, par son caractère décentralisé, ajoute de nouvelles difficultés à ces problèmes, comme par exemple assurer l'équité entre les participants, ou faire face à leur comportement égoïste.

Les approches actuelles pour la gestion des ressources dans les grilles ont d'importants inconvénients. Un certain nombre de travaux antérieures ignorent la décentralisation dans l'organisation de la grille. Les approches qui utilisent les intermédiaires (en anglais, les brokers) sont incapables de modéliser la grille dans son ensemble, car ils sont basés sur des tarifs (fixés à l'extérieur du modèle) pour contrôler l'accès aux ressources. Enfin, dans les approches économiques, même si le prix fait partie du modèle, elle est généralement exprimée en une sorte de monnaie, ce qui peut avoir une inci-

dence négative sur le partage et la coopération, et donc la communauté des utilisateurs qui participent à la grille.

Le but de ce travail de thèse est d'étudier les effets de la décentralisation sur le problème d'ordonnancement dans les grappes et grilles de calcul, et ce grâce à des modèles mathématiques simples capturant les caractéristiques fondamentales qui distinguent les grappes des ordinateurs parallèles classiques. Nous faisons notamment appel à la théorie des jeux afin de mesurer les conséquences de la prise de décision décentralisée par les participants égoïstes. Nous utilisons également la théorie de l'optimisation équitable pour garantir que toutes les parties sont traitées de manière équitable.

Les grappes se composent de ressources des différents domaines administratifs. Nous prétendons que les grappes et grilles doivent être étudiés avec des approches capables de voir cette hétérogénéité. Nous ainsi proposons deux types de cadres de mathématiques: *la théorie des jeux* dans les systèmes où les propriétaires des ressources individuelles ont un contrôle complet sur leurs machines; *l'optimisation multicritère équitable* dans des systèmes plus centralisés, dans laquelle un gestionnaire de ressources peut ordonnancer des tâches sur les ressources, mais les différentes parties prenantes doivent être traités équitablement.

Notre but n'est pas de proposer un logiciel riche en fonctionnalités pour ordonnancer les tâches, pour un certain nombre de raisons. Tout d'abord, bon nombre de ces logiciels sont déjà disponibles. Deuxièmement, au niveau actuel de développement des grappes, un Ordonnanceur ne peut pas être utilisé dans des systèmes autres que précisément celui pour lequel il a été créé. Troisièmement, la relative nouveauté des grappes et, par conséquent, l'absence des techniques de la validation largement adoptés rend difficile la validation scientifique de ces logiciels. Au lieu de cela, nous analysons nos modèles, avec des moyens similaires à ceux utilisées dans la théorie d'ordonnancement. Nous fournissons une analyse théorique des scénarios dans le pire cas, des bornes inférieures sur les performances des algorithmes et des résultats de simulation, le cas échéant.

7.2 Modèle de grilles Multi-Organizations

La notion centrale du modèle proposé dans cette thèse est celle d'*organisation*: un établissement participant, une entité qui regroupe une ressource donnée à la grappe et les utilisateurs locaux qui veulent utiliser l'ensemble du système. Ainsi, une grille de calcul interconnecte logiquement plusieurs ressources telles que les grappes d'ordinateurs, mais aussi des pièces de matériel spécialisé comme des dispositifs d'affichages sophistiqués, des microscopes, ou des séquenceurs d'ADN. Les utilisateurs du réseau sont regroupés dans des organisations, comme les laboratoires ou les facultés. Chaque organisation est propriétaire de l'une des ressources, qui est la contribution de l'organisation à la grappe. En contribuant, une organisation s'attend à ce que ses utilisateurs soieont traités de façon équitable lors de l'accès à d'autres ressources. Nous partons du principe que les organismes sont indépendants les uns des autres. En conséquence, chaque organisation ne s'intéresse qu'à la performance des tâches produites par ses membres. En outre, les ressources locales peuvent avoir leurs ordonnanceurs qui ordonnancent les tâches en fonction de certains critères particuliers. Un ordonnanceur centralisé coordonne ces ordonnanceurs locaux. Toutefois, comme les ressources appartiennent à des organisations locales, un ordonnanceur local n'est pas obligé de suivre les conseils de l'ordonnanceur de grappe. Par exemple, une organisation peut modifier localement la solution proposée ou même quitter complètement la grappe, si l'organisation trouve sa performance inacceptable.

Nous supposons aussi qu'il n'y a pas de moyens extérieurs de compensations pour l'accès aux ressources. Une organisation ne peut pas explicitement "payer" autre organisation ni sous forme d'argent, ni sous frome de troc.

Dans ce travail, nous considérons la minimisation des délais d'achèvement des tâches dans la grappe. Nous introduisons un ordonnanceur centralisé au niveau de la grappe qui propose un ordonnancement pour chaque ressource. Cependant, la puissance de l'ordonnanceur centralisé et, par conséquent, le type de solutions qu'il peut imposer sur les différents processeurs, dépend

en grande partie du niveau de contrôle qu'ont les différentes organisations sur leurs ressources. Nous étudions les problèmes de trois points de vue, ce qui conduit à trois approches différentes pour l'optimisation: l'optimisation multi-critères, la théorie des jeux et l'optimisation multi-critères avec contraintes.

Tout d'abord, dans le cas le plus restreint, on peut supposer que l'organisation n'est pas en mesure d'imposer un quelconque ordonancement sur ses ressources locales, ni qu'elle est en mesure de quitter la grappe. Cette hypothèse reflète les modèles de toutes les ressources qui sont détenues et contrôlées directement par la grille. Le problème se transforme alors en optimisation multi-critères de mesures de la performance des organisations.

Deuxièmement, chaque organisation peut avoir un contrôle complet sur l'ordonnancement dans la ressource locale. Dans une telle situation, l'ordonnanceur de la grappe agit seulement comme un conseiller. La solution proposée doit être rentable pour chaque organisation. Cependant, chaque organisation est tentée de le modifier localement, si cela augmente la performance de l'organisation. Par conséquent, un tel problème doit être analysé avec la théorie de jeux.

La troisième situation est une combinaison des deux premières. Nous supposons que chaque organisation décide de joindre ou de quitter la grille de la façon indépendante. Cependant, une fois à l'intérieur, l'organisation accorde un contrôle total sur ses ressources à l'ordonannceur de la grappe. Pourtant, une organisation va quitter le système, si le profit expérimenté est inférieur à la performance de l'organisation pourrait parvenir à se trouver à l'extérieur. Par conséquent, afin de maintenir le réseau, le système de gestion des ressources doit obtenir une performance acceptable, non seulement au niveau de la communauté des utilisateurs (comme en l'ordonnancement classique), mais aussi sur le niveau entre les organisations. Ce problème peut être défini comme l'optimisation multi-critères avec contraintes du vecteur de mesures du rendement de leurs organisations respectives.

7.3 Partage de Resources

Dans ce chapitre, nous considérons la grappe comme un outil pour accéder à des ressources spécialisées. Nous supposons que des ressources sont dédiées. Chaque tâche dans le système doit être calculée sur une ressource déterminée, pas nécessairement celui appartenant au propriétaire de la tâche. Le principal problème est alors de déterminer quand ces tâches étrangères doivent être exécutées. La solution proposée par les approches classiques est d'exécuter des tâches en vue de l'accroissement de leur temps de calcul sur chaque processeur (Shortest Processing Times, SPT). Cependant, cette solution peut être inefficace pour les organisations possédant des équipements très demandés. Une autre solution possible est de mélanger l'exécution avec des tâches "étrangers" sur chaque processeur, mais il est généralement globalement inefficace.

Nous étudions trois modèles qui diffèrent par le niveau de contrôle de l'ordonnanceur de la grappe sur les ressources.

Tout d'abord, nous analysons le problème de l'ordonnancement équitable dans les systèmes de planification des ressources qui sont la propriété de la communauté et dont le ordonnanceur céntralisé a un contrôle complet. Nous formulons le problème comme l'optimisation multi-critères. Nous étudions la complexité du problème en comptant le nombre possible des solutions Pareto-optimale existent pour une instance donnée. Un algorithme optimal devrait retourner toutes les solutions optimales, entre lesquelles un choix peut être fait par un gérant la grappe. Cependant, nous montrons que le nombre des solutions Pareto-optimale peut être exponentielle en fonction du nombre des tâches. Par conséquent, il n'existe aucun algorithme efficace énumérant tous les meilleurs ordonnancements.

Comme la production de l'ensemble des solutions n'est pas efficace, nous étudions le problème de trouver un seul ordonnancement avec les performances données de chaque organisation. Nous considérons comme un cas limité une ressource et deux organisations. Nous montrons que même dans ce cas limité, le problème de décision est NP-complet. Ainsi, le problème

d'ordonnancement dans la grappe défini dans le présent chapitre est également NP-dur.

Ensuite, nous proposons trois algorithmes d'optimisation multicritères: une recherche exhaustive qui essaye de tous ordonnancements possibles et supprime celles qui sont équitablement dominés; la programmation dynamique qui trouve l'ensemble des ordonnancements Pareto-optimales par la construction partielle des ordonnancements (en précisant les propriétaires des premières i tâches sur une ressource) et les élargit pour contenir toutes les tâches sur l'ensemble des ressources; et une heuristique gourmande qui modifie les ordonnancements de la façon itérative par changer l'ordre de deux tâches exécutés l'un après l'autre sur la même ressource.

Deuxièmement, nous étudions le problème de l'ordonnancement dans le cas décentralisé, dans laquelle l'ordonnanceur de la grappe suggère seulement un calendrier, qui peut être modifié ultérieurement par le propriétaire d'une ressource. Ce problème sera analysé avec la théorie de jeux. Nous montrons que le jeu correspondant à la planification décentralisée est analogue au problème connu du dilemme du prisonnier. Bien que les organisations puissent gagner en suivant les conseils de l'ordonnanceur de la grappe, chaque organisation est tentée de changer l'ordonnancement local et d'exécuter ses tâches locales au début. On note que cette stratégie par My First Job (MJF). Dans l'équilibre de Nash, toutes les organisations utilisent les stratégies MJF. Cela se traduit par d'importantes ($O(n)$) baisses de performance. Par conséquent, une forte communauté qui excerce un contrôle est nécessaire pour rendre la performance acceptable.

Troisièmement, nous analysons le problème de l'ordonnancement réalisable en cas semi-décentralisée. Ici, les organisations ne peuvent pas modifier-localement le calendrier, mais ils peuvent quitter la grappe, si le rendement est insatisfaisant. Ce problème correspond à l'approche multi-critères avec contraintes. Pour trouver une solution faisable, nous utilisons des algorithmes similaires à ceux utilisés dans l'optimisation multi-critères.

Avec un vaste validation expérimentale nous montrons que ces heuristiques produisent des résultats qui ne sont pas loin de l'optimum et, en même temps, leur temps d'exécution et plus rapide de quelques ordres de grandeur. Nos expériences montrent également que les contraintes pesant sur l'optimisation n'aggravent pas les résultats de façon significative. C'est pourquoi un fort contrôle exercé de façon communautaire par la grappe est indispensable pour atteindre une bonne performance.

7.4 Tâches Divisibles

Dans ce chapitre, nous considérons la grappe comme un outil pour calculer les tâches divisibles (divisible load). Suite à notre modèle général de la grille, nous supposons que chaque tâche a été préparée par une organisation. Pourtant, elle peut être calculée en parallèle sur des ressources des autres organisations, si ces organisations conviennent d'accepter la tâche. Nous étudions quelle décision devraient prendre les organisations en présence d'incertitude. On considère un modèle en ligne (on-line), dans laquelle les tâches sont produites pendant la durée de vie du système et sont inconnus avant d'avoir été produite.

Tant qu'il n'y a pas de tâches locales, le calcul de la tâche étrangère ne causer aucun coût supplémentaire. Toutefois, si une nouvelle tâche locale est produite, la tâche étrangère bloque les ressources nécessaires et retarde la nouvelle tâche locale. Dans tel cas, la tâche étrangère est habituellement suspendue. En conséquence, le propriétaire de cette tâche ne peut pas avoir des garanties sur leur temps de l'achèvement.

Par opposition à cette approche, nous supposons que le système doit garantir le définitif temps de fin de chaque tâche soumise, qui est annoncé à l'utilisateur pendant que le travail est soumis. Nous réclamons que telle garantie donne la meilleure qualité de service que l'exécution de meilleur-effort, normalement utilise dans le calcul distribuée des tâches divisibles.

Afin d'analyser précisément l'équité du modèle, nous commençons avec calculant le retard prévu EG causé par une tâche étrangere de longueur Φ_k dans le calcul d'une prochaine tâche local, donné la taille de file d'attend actuel L_k. Après quelques suppositions, nous calculons EG comme $EG(L_k, \Phi_k) = \Phi_k + \frac{e^{-\lambda_k L_k}}{\lambda_k}\Big(e^{-\lambda_k \Phi_k} - 1\Big)$. Le retard prévu $EG(L_k, \Phi_k)$ n'est pas linéaire avec le chargement étranger Φ_k. De plus, $\Phi_k - EG\left(L_k, \Phi_k\right) > 0$ pour tous possible (positif) valeurs de Φ_k et L_k. Par conséquent, le gain que l'expéditeur obtient de l'envoi d'une partie de son chargement est toujours plus grande que le retard prévu des tâches locales futurs du récepteur

Cette fonction nous permet d'analyser un cas simplifié avec l'approche d'optimisation. Nous étudions un algorithme classique de répartition de charge, Averaging Load Balancing, qui distribue le chargement pour que chaque ressource finit le calcul au même moment. Nous montrons que la solution proposée est non seulement globalement optimale, mais aussi le plus juste, mais il détériore toujours la performance de la ressource le moins chargée.

Avec l'approche de théorie des jeux, nous démontrons que la solution juste ne sera jamais réalisable. Bien que la balance du chargement a pour résultat la solution globalement optimale, cette action est dominée par aucune balance de chargement par l'organisation qui reçoit la charge.

Dans le cas de ressources chargées pareillement, nous montrons que, si nous obligeons les organisations à collaborer pendant les périodes plus longues, Averaging Load Balancing devient la stratégie dominante pour chaque organisation. La probabilité d'être le récepteur est similaire à cela d'être l'expéditeur, si les ressources sont chargées pareillement et si le système force les organisations à commettre à leurs décisions pour quelque période, au lieu d'eux permettre de coopérer seulement instantanément. Par conséquent, les deux organisations doivent profiter de la coopération.

Nous validons le modèle proposé empiriquement en construisant un simulateur de la grappe avec les participants qui apprennent automatiquement. Dans le simulateur, chaque ressource mesure le temps d'exécution de ses

tâches locales et ajuste en conséquence sa complaisance pour joindre la coalition de la répartition de charge. Dans la coalition, l'algorithme de la répartition de charge est périodiquement exécuté. L'algorithme équilibre la charge en déplaçant des tâches ou leurs fragments entre les ressources. Dans les expériences, si les ressources ont été chargées pareil et leur chargement était haut, les travaux ont été accélère de 50% à 237%. Les résultats sont mieux quand il y a plus d'organisations dans la grappe. Ceci est parce que s'il y a plus de ressources, plus de ressources en fait peut envoyer leur chargement et profiter de la répartition.

Pourtant, quand la charge est inégale parmi les ressources, le simple algorithme de la répartition de charge devient inadmissible pour les ressources moins chargées. Pendant que les tâches des organisations surchargées sont considérablement accélères, les autres organisations perdent constamment en coopérant.

De plus, nous montrons que si les organisations prendre des décisions dépendent de leur longueur de file actuelle, la répartition de charge ne sera jamais exécutée. Une organisation O_k prévoit qu'il envoie ses tâches seulement si sa file actuelle est plus longue que la longueur de file prévue de l'autre organisation O_l. Seulement dans ce cas, la répartition de charge est profitable pour O_k. L'organisation O_l, cependant, est capable de la même analyse. Si la longueur de file L_l de M_l est moins que le moyen $\frac{\rho}{\gamma}$, M_l ne participera pas dans la répartition de charge. Afin d'équilibrer la charge, nous avons besoin de deux organisations qui vont participer. Par conséquent, si O_k veut participer, il doit savoir que si O_l participe, la longueur de file de M_l sera plus longue que $\frac{\rho}{\gamma}$. Ceci modifie la fonction de distribution de probabilité de M_l. Par conséquent, la répartition de charge devient profitable pour l'organisation O_k seulement si sa longueur de file est plus longue que $\frac{\rho+1}{\gamma}$. Nous pouvons itérer ce raisonnement plus. Il fait la répartition de charge d'une étape analogue au jeu d'un concours de beauté. Le plus de répétitions de ce raisonnant font les joueurs, le plus long la longueur de file minimum qui fait la répartition de charge profitable. Comme les joueurs sont

complètement rationnels, la longueur de file profitable minimum augmente indéfiniment. Ainsi, la volonté de participation à la répartition de charge n'arrive jamais.

Nous résolvons ce problème en introduisant un autre algorithme de distribution de chargement (appelé la répartition de charge limitée et itérative, Bounded Iterative Load Balancing, BILB), qui emploie les idées générales de l'optimisation équitable contrainte. L'algorithme utilise deux mécanismes : la répartition de charge itérative, dans laquelle premièrement les ressources de moindre-chargé sont équilibrées, alors les ressources sont itérativement ajoutées dans l'ordre de leur charge locale; et la répartition de charge limitée, dans laquelle aucune ressource est forcée d'accepter la charge étrangère qui augmenterait sa file locale trop. La première technique favorise les ressources moins charges, le deuxième utilise directement des conclusions du modèle de coût proposé, où nous avons montré qu'acceptant la charge étrangère coût moins quand la file locale est courte. Dans les expériences, cet algorithme livre de meilleurs résultats que l'algorithme précédent dans tous les cadres considérés. De plus, on est capable de produire le gain même pour les organisations moins chargés.

7.5 Tâches parallèles

Dans ce chapitre nous étudions l'extension multi-organisationnel du problème classique de l'ordonnancement de tâches parallèles. La version classique du problème consiste en l'ordonnancer les tâches sur une ressource multiprocesseur. Chaque tâche doit être exécutée en parallèle sur un nombre spécifique de processeurs. Le but est de minimiser le makespan, la date de terminaison de la dernière tâche. Ce modèle adapte bien à un scénario populaire de l'usage d'un super-ordinateur, dans lequel une grappe de serveurs (un cluster) est partagé entre des tâches MPI.

Nous étendons le modèle classique aux organisations multiples. Dans notre modèle, chaque organisation possède une ressource de multi-processeur

et produit des tâches qui sont localement ordonnancées sur sa ressource. Après avoir exécuté un algorithme de répartition de charge dans la grille, ces tâches peuvent être alors exécutés sur les autres ressources. Dans ce chapitre nous étudions les algorithmes d'approximation qui donnent une garantie sur la performance dans le pire des cas. Nous visons à mesurer l'impact de décentralisation organisationnelle et de matériel sur le rapport d'approximation de l'algorithme d'ordonnancement.

Nous ne permettons pas à une tâche d'être exécutée en parallèle sur deux, ou plus de ressources. Cette supposition vient l'hétérogénéité implicite de liens de réseau. Comme une tâche doit être exécutée en parallèle, nous supposons que les fragments de la tâche exécutée sur les processeurs différents communiqueront activement l'un avec l'autre. Dans une ressource, telle communication est rapide. Cependant, les liens entre les ressources sont quelques ordres de magnitude plus lente, donc communication entre les ressources augmenterait le temps d'exécution du travail.

Avec la supposition que chaque organisation mesure le temps de terminaison de ses tâches localement produites et pas le temps de terminaison sur la ressource locale, l'exécution d'une tâche étrangère à la fin de l'ordonnancement ne cause pas de coût pour le propriétaire. Ceci est une conséquence directe de la supposition que le modèle est différé (off-line).

Nous commençons sur le cas simple de l'ordonnancement des tâches parallèles rigides sur une ressource avec m processeurs identiques. Nous utilisons l'algorithme classique de l'ordonnancement de liste (list scheduling), qui a un rapport d'approximation $2 - frac1m$. Nous montrons que si les tâches sont triées selon le nombre décroissant de processeurs exigés, l'ordonnancement résultant peut être divisé au plus en deux périodes en ce qui concerne l'utilisation de la ressource. Au commencement, plus que la moitié des processeurs est utilisée. Alors, moins que la moitié des processeurs est utilisée, pendant au plus le temps de p_{max}.

Nous montrons que, même si l'ordonnanceur de la grappe retient le contrôle complet sur toutes les ressources, la discontinuité de ressources conduit

à un rapport d'approximation de 3 pour l'algorithme d'ordonnancement de liste. De plus, l'équité des résultats ne peut pas être garantie. Comme l'ordonnancement de liste mélange toutes les tâches, nous ne pouvons pas avoir de garantie sur l'impartialité entre makespans des organisations. Nous analysons brièvement deux façons dont l'ordonnancement de liste peut être étendu pour améliorer l'impartialité des ordonnancements résultants. L'algorithme juste doit favoriser des travaux d'organisations les moins-chargent (en ce qui concerne le travail total de l'organisation $W(O_k)$ ou la longueur de la tâche la plus longue $p_{\max}(O_k)$). Cependant, telles priorités sont impossibles de s'appliquer dans un algorithme de liste. Premièrement, la priorité d'une tâche peut être exprimée en commandant la tâche plus tôt possible dans la première phase d'algorithme d'ordonnancement. Cependant, le comportement avide d'algorithme de liste dans la deuxième phase peut résulter en ordonnancer les tâches ayant la priorité plus basse, mais exigeant moins de processeurs avant les plus hauts travaux de priorité. Deuxièmement, l'algorithme de liste peut planifier des tâches d'organisations différentes dans la mode de backfilling. Si on suppose que O_1 a la priorité par-dessus O_2, qui a la priorité par-dessus O_3,..., toutes les tâches de O_1 seraient ordonnancées sur toutes ressources disponibles. Alors, les tâches de O_2 seraient ordonnancées entre eux avec une restriction supplémentaire qui une tâche de O_2 ne peut pas retarder une tâche de O_1. L'algorithme ordonnance alors les tâches de O_3,...,O_N d'une façon similaire. Malheureusement, tel un algorithme n'a pas la garantie fixée sur le rapport d'approximation.

Alors, nous montrons que le coût de décentralisation organisationnelle n'est pas aussi haute que dans les grappes précédemment considérées, parce que le Prix d'Anarchie est entre 3/2 et 2. Pour une organisation, une stratégie avide appelé mes tâches premierèment (My Job First, MJF), ordonnance premièrement ses tâches locales avec un algorithme de liste. Alors, les tâches étrangères sont ordonnancées ou à la fin de l'ordonnancement, ou dans les écarts pour qu'ils ne retardent pas de tâches locales. Telles stratégies forment le seul équilibre de Nash dans le jeu d'ordonnancement. Nous montrons que

le prix d'anarchie est au plus 2 en construisant un ordonnancement **MJF** d'un ordonnancement optimal en étendant le makespan sur chaque ressource au plus deux fois.

Finalement, nous proposons un algorithme d'ordonnancement qui a un rapport d'approximation fixe sur le makespan de système et ne dégrade pas en même temps les makespans des locales d'organisations.

A cause de la contrainte qu'aucun makespan local ne peut être augmenté, quelques solutions globalement-optimaux ne sont pas faisables. Nous montrons que n'importe quel algorithme d'ordonnancement résolvant le problème contraint a un rapport d'approximation d'au moins $\frac{3}{2}$ en ce qui concerne la solution optimale de la minimisation de $C_{\max}$ du système.

L'algorithme de repartition de charge des organisations multiples (Multi-Organisational Load Balancing Algorithm, MOLBA), indiqué par MOLBA(α) ($\alpha geq 1$), calcule une borne inferieuré du makespan global, choisit toutes les organisations dont makespan est plus grand que $(\alpha + 1)$, et ordonnance alors certains de leurs tâches sur les ressources moins chargées. α est un paramètre de l'algorithme qui exprime le compromis entre l'efficacité globale et les contraintes sur les makespans locales.

Dans le cas spécial où la dernière tâche terminée est *basse* (la tâche utilise moins que la moitié des processeurs), MOLBA(2) est un algorithme 3-approché. Dans le cas général MOLBA(3) est 4-approché.

MOLBA(3) garanties que le makespan global $C_{\max}$ n'est pas pire que quatre fois l'optimum, $4C^*_{\max}$. Néanmoins, la distribution de chargement résultant d'exécution de l'algorithme peut être inégal. Par exemple, MOLBA(3) ne modifie pas l'ordonnancement d'une organisation avec $C_{\max}(O_k) = 4C^*_{\max} - epsilon$. Pourtant, c'est possible qu'il y ait des tâches sur M_k qui commencent après toutes les autres ressources ont fini.

Nous proposons un autre algorithme, l'algorithme itératif de repartition de charge (Iterative Load Balancing Algorithm, ILBA), qui améliore l'ordonnancement retourné par MOLBA(α) en équilibrant la charge entre les ressources. ILBA ne retarde pas de tâches, ainsi, donné un ordonnancement

produit par MOLBA(α), ILBA remplit la contrainte et n'empire pas le rapport d'approximation. Nous montrons aussi que dans un cas d'ordonnancement des tâches séquentiels, ILBA a le même rapport d'approximation que n'importe quel algorithme d'ordonnancement de liste. En commençant avec les ressources moins chargées et en ajoutant itérativement les plus chargées, ILBA permet aussi de réduire les makespans sur les ressources moins chargé.

ILBA trie les ressources par des makespans non-diminuant. Suppose que $C_{\max}(M_1) \leq C_{\max}(M_2) \leq \cdots \leq C_{\max}(M_N)$. L'algorithme, pour tout $k = 2, \ldots, N$, re-ordonnance toutes les tâches exécutées sur M_k sur $\{M_1, \ldots, M_k\}$. Les tâches sont ordonnancées de façon séquentielle dans l'ordre de leur exécution d'ordonnancement original. Chaque tâche J^i est ordonnancée à la ressource qui a la plus première bande de processeurs libres de largeur au moins p^i et la hauteur au moins q^i. Ainsi, de même à l'étape précédente de l'algorithme, une tâche reçue de M_k ne peut pas retarder des tâches déjà planifiées sur $\{M_1, \ldots, M_k\}$.

Dans quelques cas spéciaux, ILBA peut être exécuté directement après avoir exécuté l'ordonnancement local (il n'y a pas besoin d'exécuter MOLBA(α)). Nous prouvons les rapports d'approximation dans deux cas spéciaux. Si toutes les tâches sont séquentielles, ILBA est un $(2 - \frac{1}{Nm})$-approximation de l'optimal C^*_{max}. Si toutes les tâches sont hautes (ont besoin de plus de la moitié des processeurs disponibles sur chaque ressource), ILBA est un $(2 - \frac{1}{N})$-approximation..

Nous fournissons une évaluation expérimentale des algorithmes. Les algorithmes ont été comparés en ce qui concerne le makespan du système qui a été produit. Comme les instances ont différé par exemple dans le nombre de tâches, ou leurs tailles, pour chaque instance et chaque algorithme nous avons calculé le rapport entre le makespan produit et les bornes inférieurs (la longueur de la tâche la plus longue et le chargement moyen). Les résultats de nos expériences montrent que MOLBA avec ILBA est un algorithme efficace qui améliore considérablement l'exécution en comparaison de l'ordonnancement local. Cependant, nous avons observé aussi que les

charges de travail produites par le générateur réaliste étaient plus faciles à ordonnancer.

7.6 Conclusions

Dans cette thèse nous avons proposé un modèle de système qui rassemble des organisations indépendantes. Chaque organisation dispose de quelques ressources et, en échange, veut utiliser une partie du système entier. Le modèle résultant peut être analysé par les outils mathématiques telles que la théorie des jeux ou l'optimisation équitable. De plus, le modèle est assez générique pour être adapté à un système spécifique, sans perdre la possibilité d'être analysé. Nous avons présenté trois applications du modèle aux divers types des grappes de calcul. En plus, le modèle doit être possible d'adapter aux autres types de systèmes modernes, tels que les systèmes pair-à-pair ou les communautés web 2.0.

Un des résultats principaux de cette recherche est une démonstration que la décentralisation organisationnelle complète d'un système est coûteux sur le plan de la performance. Dans tel un système organisationnellement décentralisé, des organisations individuelles maintiennent le contrôle complet pardessus leurs ressources. Nous avons analysé de tels systèmes avec l'approche jeu-théorique. Dans les grappes qui partagent les ressources dédiées, dans le pire cas la perte était linéaire sur le nombre de tâches. Dans le modèle des tâches divisibles, la répartition de charge entre les ressources n'a jamais été exécutée. Seulement dans un de trois modèles considérés, la perte de performance était acceptable. Cependant, ce modèle a exigé quelques suppositions fortes.

Nous avons démontré aussi qu'il était impossible de respecter des buts des membres individuels tout en gardant la performance satisfaisante du système entier en même temps. A cette fin, les membres sont obligés d'accorder un contrôle même partiel sur leurs ressources par rapport à une entité centrale.

Les contraintes du problème d'optimisation garantissent que les buts des membres sont respectés et accomplis.

Nous avons appliqué l'optimisation multicritère équitable pour garantir l'équité des résultats entre les participants individuels. Nous avons caractérisé des propriétés d'une solution juste par les axiomes d'anonymat, de la monotonie et du principe de transferts. L'optimisation équitable s'est avérée être un outil commode, comme l'équité axiomatique peut équilibrer la performance du système entier avec l'équité entre des participants individuels. Ce n'était pas toujours possible, cependant, formellement garantir l'équité des résultats retournés par les algorithmes, à cause soit du manque d'information (dans les modèles en ligne) soit du manque de moyens algorithmiques (dans l'ordonnancement des tâches parallèles).

En analysant le cas d'une grappe qui partage des ressources dédiées, nous avons démontré que la décentralisation organisationnelle cause la perte significative de la performance même dans un modèle différé. Néanmoins, si les organisations contrôlent les ordonnancements locales, une solution efficace et juste peut être produite.

Le cas de la répartition de la charge des tâches divisibles est aussi sensible au manque de contrôle centralisé. Dans les systèmes décentralisé, la répartition de charge n'est jamais exécutée, bien qu'il garantît une distribution équitable des tâches. Nous avons géré pour répartir la charge du système en obligeant les organisations à pendre leurs décisions pour quelques périodes de temps, sans tenir compte des longueurs des files d'attente. De plus, nous avons proposé un algorithme de répartition itérative de charge qui distribue les profits de façon plus équitable que l'algorithme classique.

Nous avons considéré aussi le problème de l'ordonnancement des tâches parallèles sur des ressources multiprocesseurs. Ici, la décentralisation n'a pas causé de baisse de la performance du système entier. Le modèle est différé, ainsi une tâche étrangère exécutée à la fin de l'ordonnancement n'empire pas la performance du propriétaire de la ressource. En même temps, le propriétaire de la tâche peut réduire son makespan. Nous avons proposé

un algorithme de l'ordonnancement qui a la performance dans le pire des cas fixe sur le makespan de système et qui n'augmente pas les makespans d'organisations individuelles.

Les résultats présentés dans ce travail ont exigé beaucoup d'hypothèses. Par exemple, nous avons supposé que l'ordonnanceur connaît la taille exacte de chaque tâche (la clairvoyance) ou que toutes les tâches dans le système sont connues et prêtes à être exécutées avant le début d'algorithme d'ordonnancement (des modèles différés). De telles suppositions sont communes dans la théorie d'ordonnancement, elles permettent de dériver quelques résultats mathématiques précis sur les modèles simplifiés. Clairement, en même temps, elles simplifient la réalité complexe, donc les algorithmes dans la forme présentée dans ce travail ne peuvent pas être directement appliqués dans un logiciel d'ordonnancement de grappe. Cependant, les algorithmes qui se sont avéré être efficaces sur les modèles simplifiés peuvent être transférés aux vrais environnements. Dans ce cas, la clairvoyance serait remplacée par les estimations du temps d'exécution fournis par l'utilisateur. Les tâches qui sont produites pendant la vie du système (en ligne) seraient considérées dans les groupes (batches), dans lesquels un algorithme différé peut être utilisé. De même, les algorithmes pour ordonnancer des tâches divisibles pourraient être facilement adaptés pour des applications des groupes des tâches (bag-of-tasks), en fixant la taille minimale du grain.

Cette approche théorisée nous a permis de dériver quelques résultats précis et mathématiques sur la performance des systèmes et des algorithmes considérés. Nous avons modélisé ce que nous considérons l'essence de la grappe. Après, nous avons utilisé ce modèle simplifié pour valider nos algorithmes. L'approche alternative serait de construire un modèle beaucoup plus réaliste de grappe et alors d'évaluer les algorithmes par simulation. Nous prétendons cependant que ce n'est pas possible de préparer une simulation précise des grappes, qui sont des systèmes considérablement complexes et, ainsi, ils font intervenir des phénomènes traverses à presque toutes les disciplines de l'informatique. Un simulateur réaliste de la grappe modèliserait la fiabilité de

noeuds, les sites et le réseau; les liens de réseau, avec le chargement de fond ; les applications avec leurs modèles d'usage pour le pouvoir computationnel, le réseau et les autres ressources ; les utilisateurs qui soumettent des applications en fonction du chargement observé du système ; etc. , Ce n'est pas possible clairement de construire des modèles précis pour tous ces phénomènes, en particulier comme ils réagissent réciproquement l'un avec l'autre. De plus, à cette étape de développement des grappes, ce n'est pas encore clair quels phénomènes sont cruciaux, et qui peut être omis sans la détérioration du réalisme de simulation. Ainsi, l'un ne peut pas avoir d'espérances réalistes sur la performance d'un algorithme d'ordonnancement qui a été évalué uniquement par la simulation.

La conclusion principale de cette étude est que les grappes sans aucune forme de coordination ou contrôle centralisé fonctionnent inefficacement. La perte résultante de performance peut être proportionnelle à l'usage de système. Pourtant, en rajoutant un quelconque niveau de coordination, il est possible de partager le groupe de ressources disponibles parmi les participants pour que personne ne perde en coopérant.

Bibliography

[Acar et al., 2002] Acar, U. A., Blelloch, G. E., and Blumofe, R. D. (2002). The data locality of work stealing. *Theory Comput. Syst.*, 35(3):321–347.

[Agnetis et al., 2004] Agnetis, A., Mirchandani, P., Pacciarelli, D., and Pacifici, A. (2004). Scheduling Problems with Two Competing Agents. *Operations Research*, 52(2):229–242.

[Albers et al., 2007] Albers, S., Mueller, F., and Schemzer, S. (2007). Speed scaling on parallel processors. In *Proceedings of the 19th Annual Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–298. ACM.

[Anderson, 2004] Anderson, D. (2004). BOINC: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10.

[Anderson and Fedak, 2006] Anderson, D. and Fedak, G. (2006). The Computational and Storage Potential of Volunteer Computing. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*.

[Andrade et al., 2003] Andrade, N., Cirne, W., Brasileiro, F., and Roisenberg, P. (2003). Ourgrid: An approach to easily assemble grids with equitable resource sharing. In *9th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862 of *LNCS*, pages 61–86. Springer.

[Angel et al., 2006] Angel, E., Bampis, E., and Pascual, F. (2006). The price of approximate stability for a scheduling game problem. In *Proceedings of Euro-Par*, volume 4128 of *LNCS*. Springer.

[Berman et al., 2003] Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., Shao, G., Smallen, S., Spring, S., Su, A., and Zagorodnov, D. (2003). Adaptive computing on the grid using apples. *IEEE Trans. on Parallel and Distributed Systems*, 14(4):369–382.

[Błażewicz, 1996] Błażewicz, J. (1996). *Scheduling in Computer and Manufacturing Systems*. Springer.

[Bolze et al., 2006] Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lantéri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.-A., and Touche, I. (2006). Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494.

[Brucker, 2004] Brucker, P. (2004). *Scheduling Algorithms*. Springer.

[Buragohain et al., 2003] Buragohain, C., Agrawal, D., and Suri, S. (2003). A game theoretic framework for incentives in p2p systems. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 48. IEEE Press.

[Buyya et al., 2005] Buyya, R., Abramson, D., and Venugopal, S. (2005). The grid economy. In *Special Issue on Grid Computing*, volume 93, pages 698–714. IEEE Press.

[Calas et al., 2005] Calas, Y., Capit, N., and Gabarron, E. (2005). Cigri : Expériences autour de l'exploitation d'une grille légère. In *6ème Journées Réseaux (JRES)*.

[Camerer, 2003] Camerer, C. F. (2003). *Behavioral Game Theory: Experiments in Strategic Interaction*. Princeton University Press.

[Capit et al., 2005] Capit, N., Costa, G. D., Georgiou, Y., Huard, G., n, C. M., Mounié, G., Neyron, P., and Richard, O. (2005). A batch scheduler with high level components. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*.

[Chun et al., 2004] Chun, B.-G., Chaudhuri, K., Wee, H., Barreno, M., Papadimitriou, C. H., and Kubiatowicz, J. (2004). Selfish caching in distributed systems: a game-theoretic analysis. In Chaudhuri, S. and Kutten, S., editors, *PODC*, pages 21–30. ACM.

[CiGri, 2007] CiGri (2007). Cigri: Ligthweight grid solution. `http://cigri.imag.fr/`.

[Dongarra et al., 2007] Dongarra, J., Jeannot, E., Saule, E., and Shi, Z. (2007). Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *Proceedings of the 19th Annual Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 280–288. ACM.

[Dumitrescu et al., 2005] Dumitrescu, C., Raicu, I., and Foster, I. T. (2005). Di-gruber: A distributed approach to grid resource brokering. In *SuperComputing*.

[Dutot et al., 2005] Dutot, P., Eyraud, L., Mounié, G., and Trystram, D. (2005). Scheduling on large scale distributed platforms: from models to implementations. *Int. J. Found. Comput. Sci.*, 16(2):217–237.

[EGEE, 2007] EGEE (2007). Enabling grids for e-science. `http://www.eu-egee.org/`.

[Eyraud-Dubois et al., 2007] Eyraud-Dubois, L., Mounie, G., and Trystram, D. (2007). Analysis of scheduling algorithms with reservations. In *Pro-*

ceedings of the International Parallel & Distributed Processing Symposium (IPDPS), pages 1–8. IEEE Computer Society.

[Feitelson et al., 2005] Feitelson, D. G., Rudolph, L., and Schwiegelshohn, U. (2005). Parallel job scheduling — a status report. In *Proceedings of JSSPP 2004*, volume 3277 of *LNCS*, pages 1–16. Springer.

[Foster, 2002] Foster, I. (2002). What is the grid. `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`.

[Foster, 2005] Foster, I. (2005). Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, volume 3779 of *LNCS*, pages 2–13. Springer.

[Foster and Kesselman, 2004] Foster, I. and Kesselman, C., editors (2004). *The Grid 2. Blueprint for a New Computing Infrastructure*. Elsevier.

[Fudenberg and Tirole, 1991] Fudenberg, D. and Tirole, J. (1991). *Game Theory*. MIT Press.

[Garey and Johnson, 1979] Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman & Co. New York, NY, USA.

[Ghosal et al., 2005] Ghosal, D., Poon, B. K., and Kong, K. (2005). P2p contracts: a framework for resource and service exchange. *Future Generation Comp. Syst.*, 21(3):333–347.

[gLite, 2007] gLite (2007). glite: Lightweight middleware for grid computing. `http://glite.web.cern.ch/glite/`.

[Golle et al., 2001] Golle, P., Leyton-Brown, K., and Mironov, I. (2001). Incentives for sharing in peer-to-peer networks. In *EC '01: Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 264–267. ACM Press.

[Graham, 1969] Graham, R. (1969). Bounds on multiprocessor timing anomalies. *SIAM J. Appl. Math*, 17(2):416–429.

[Graybill and Melhem, 2002] Graybill, R. and Melhem, R., editors (2002). *Power Aware Computing*. Springer.

[Grosu and Chronopoulos, 2005] Grosu, D. and Chronopoulos, A. T. (2005). Noncooperative load balancing in distributed systems. *J. Parallel Distrib. Comput.*, 65(9):1022–1034.

[Grosu and Chronopoulos, 2004] Grosu, D. and Chronopoulos, T. (2004). Algorithmic mechanism design for load balancing in distributed systems. *IEEE Trans. on Systems, Man and Cybernetics–Part B: Cybernetics*, 34(1):77–84.

[Gruber et al., 2006] Gruber, R., Keller, V., Kuonen, P., Sawley, M.-C., Schaeli, B., Tolou, A., Torruella, M., and Tran, T.-M. (2006). Towards an intelligent grid scheduling system. In *Proceedings of International Conference on Parallel Processing and Applied Mathematics (PPAM) 2005*, volume 3911 of *LNCS*, pages 751–757.

[Hochbaum, 1997] Hochbaum, D. S., editor (1997). *Approximation Algorithms for NP-Hard Problems*. PWS Publishing.

[Iosup et al., 2006] Iosup, A., Dumitrescu, C., Epema, D., Li, H., and Wolters, L. (2006). How are real grids used? the analysis of four grid traces and its implications. In *Grid Computing, 7th IEEE/ACM International Conference on*, pages 262–269.

[Kenyon and Cheliotis, 2004] Kenyon, C. and Cheliotis, G. (2004). Grid resource commercialization: economic engineering and delivery scenarios. In Nabrzyski, J., Schopf, J. M., and Weglarz, J., editors, *Grid resource management: state of the art and future trends*, pages 465–478. Kluwer Academic Publishers, Norwell, MA, USA.

[Kim et al., 2007] Kim, K. H., Buyya, R., and Kim, J. (2007). Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'07)*.

[Kostreva et al., 2004] Kostreva, M. M., Ogryczak, W., and Wierzbicki, A. (2004). Equitable aggregations and multiple criteria analysis. *European Journal of Operational Research*, 158:362–377.

[Koutsoupias and Papadimitriou, 1999] Koutsoupias, E. and Papadimitriou, C. (1999). Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *LNCS*, pages 404–413. Springer.

[Kurowski et al., 2004a] Kurowski, K., Ludwiczak, B., Nabrzyski, J., Oleksiak, A., and Pukacki, J. (2004a). Dynamic grid scheduling with job migration and rescheduling in the gridlab resource management system. *Scientific Programming*, 12(4):263–273.

[Kurowski et al., 2004b] Kurowski, K., Nabrzyski, J., Oleksiak, A., and Weglarz, J. (2004b). Multicriteria aspects of grid resource management. In Nabrzyski, J., Schopf, J. M., and Weglarz, J., editors, *Grid resource management: state of the art and future trends*, pages 271–293. Kluwer, Norwell, MA, USA.

[Kwok et al., 2005] Kwok, Y.-K., Song, S., and Hwang, K. (2005). Selfish grid computing: Game-theoretic modeling and nash performance results. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*.

[Laoutaris et al., 2005] Laoutaris, N., Telelis, O., Zissimopoulos, V., and Stavrakakis, I. (2005). Distributed selfish replication. *IEEE Trans. on Parallel and Distributed Systems*, 17(12):1401–1413.

[Larson and Odoni, 1981] Larson, R. C. and Odoni, A. R. (1981). *Urban Operations Research*. Pretince Hall. Available from: `http://web.mit.edu/urban_or_book/www/book/`.

[Lifka, 1995] Lifka, D. (1995). The ANL/IBM SP scheduling system. In *Job Scheduling Strategies for Parallel Processing*, volume 949 of *LNCS*. Springer.

[Liu et al., 2005] Liu, J., Jin, X., and Wang, Y. (2005). Agent-based load balancing on homogeneous minigrids: Macroscopic modeling and characterization. *IEEE Trans. on Parallel and Distributed Systems*, 16(7):586–598.

[Lublin and Feitelson, 2003] Lublin, U. and Feitelson, D. (2003). The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel & Distributed Computing*, 11(63):1105–1122.

[Marchal et al., 2005] Marchal, L., Yang, Y., Casanova, H., and Robert, Y. (2005). A realistic network/application model for scheduling divisible loads on large-scale platforms. *Proceedings of the International Parallel & Distributed Processing Symposium (IPDPS)*, 01:48b.

[Marks, 2001] Marks, R. (2001). Playing games with genetic algorithms. In Chen, S.-H., editor, *Evolutionary Computation in Economics and Finance*. Springer.

[Michalewicz and Fogel, 2004] Michalewicz, Z. and Fogel, D. (2004). *How to Solve It: Modern Heuristics*. Springer Verlag.

[Osborne and Rubinstein, 1994] Osborne, M. and Rubinstein, A. (1994). *A Course in Game Theory*. MIT Press.

[Osborne, 2004] Osborne, M. J. (2004). *An Introduction to Game Theory*. Oxford.

[Papadimitriou and Steiglitz, 1998] Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, INC., dover edition.

[Pascual et al., 2007] Pascual, F., Rzadca, K., and Trystram, D. (2007). Cooperation in multi-organization scheduling. In *Proceedings of the Euro-Par 2007*, volume 4641 of *LNCS*. Springer.

[Psyco, 2007] Psyco (2007). Psyco: Just-in-time (jit) python compiler. `http://psyco.sourceforge.net/`.

[Ranganathan et al., 2004] Ranganathan, K., Ripeanu, M., Sarin, A., and Foster, I. (2004). Incentive mechanisms for large collaborative resource sharing. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 1–8.

[Rawls, 1971] Rawls, J. (1971). *The Theory of Justice*. Harvard Univ. Press.

[Robertazzi, 2003] Robertazzi, T. (2003). Ten reasons to use divisible load theory. *Computer*, 36(5):63–68.

[Rzadca, 2007] Rzadca, K. (2007). Scheduling in multi-organization grids: Measuring the inefficiency of decentralization. In *PPAM 2007 Proceedings (to appear)*, LNCS. Springer.

[Rzadca and Trystram, 2007] Rzadca, K. and Trystram, D. (2007). Promoting cooperation in selfish computational grids. *European Journal of Operational Research (to appear)*.

[Rzadca et al., 2007] Rzadca, K., Trystram, D., and Wierzbicki, A. (2007). Fair game-theoretic resource management in dedicated grids. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'07)*.

[Samuelson and Nordhaus, 1998] Samuelson, P. and Nordhaus, W. (1998). *Economics*. McGraw-Hill, 16th edition.

[Sastry et al., 1994] Sastry, P., Phansalkar, V., and Thathachar, M. (1994). Decentralized learning of nash equilibria in multi-person stochastic games with incomplete information. *IEEE Trans. on Systems, Man, and Cybernetics*, 24.

[Sgal, 1998] Sgal, J. (1998). On-line scheduling. In *Developments from a June 1996 seminar on Online algorithms*, pages 196–231, London, UK. Springer-Verlag.

[Shmoys et al., 1995] Shmoys, D., Wein, J., and Williamson, D. (1995). Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24.

[SunGrid, 2005] SunGrid (2005). Sun grid compute utility. `http://www.sun.com/service/sungrid/`.

[Vohs et al., 2006] Vohs, K., Mead, N., and Goode, M. (2006). The psychological consequences of money. *Science*, 314(5802):1154 – 1156.

[Volper et al., 2004] Volper, D. E., Oh, J. C., and Jung, M. (2004). Game theoretical middleware for cpu sharing in untrusted p2p environment. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS-2004)*.

[Walsh et al., 2001] Walsh, W. E., Wellman, M. P., Wurman, P. R., and MacKie-Mason, J. K. (2001). Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303.

[Wolski et al., 2003] Wolski, R., Brevik, J., Plank, J. S., and Bryan, T. (2003). Grid resource allocation and control using computational economies. In Berman, F., Fox, G., and Hey, A., editors, *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley & Sons.

[Wolski et al., 2001] Wolski, R., Plank, J. S., Brevik, J., and Bryan, T. (2001). G-commerce: Market formulations controlling resource allocation on the computational grid. In *Proceedings of the International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE Computer Society.

[Young, 1994] Young, H. P. (1994). *Equity: In Theory and Practice*. Princeton University Press.