Algorithmic problems in power management of computing systems

Georgios Zois

Université Pierre et Marie Curie

Athens University of Economics and Business

École doctorale Informatique, Télécommunications et Électronique
(Paris)

Laboratoire d' Informatique de Paris 6

Décision, Systèmes Intelligents et Recherche Opérationnelle

Algorithmic problems in power management of computing systems

par Georgios Zois

Thèse de doctorat des Informatics

Dirigée par Evripidis Bampis et Ioannis Milis

Présentée et soutenue publiquement le 12 Décembre 2014

Devant le jury composé de:

Evripidis Bampis, Professeur	co-directeur de thèse
Christoph Dürr, CNRS-DR2	Examinateur
Stavros Kolliopoulos, Associate Professor	Rapporteur
Evangelos Markakis, Lecturer	Examinateur
Monaldo Mastrolilli, Professor	Rapporteur
Ioannis Milis, Professor	co-directeur de thèse
Denis Trystram, Professeur	Examinateur

To my parents

Acknowledgements

First of all, I would like to express my gratitude to my supervisors, Ioannis Milis and Evripidis Bampis, for accepting me as their Ph.D. student. Without their essential guidance and support, this thesis would not exist.

I am also grateful to Christoph Dürr for sharing with me his ideas and knowledge, as well as to Dimitrios Fotakis, Emmanouil Zampetakis and Vincent Chau, that I was fortunate to work with. Moreover, I am particularly thankful to my friends and colleagues Giorgio Lucarelli and Dimitrios Letsios for their valuable cooperation. I'm also glad that I have met Martha Sideri and Evangelos Markakis who introduced me to the theory group of AUEB.

During my studies at LIP6, I benefited from being a resident of Fondation Hellenique at CIUP. Its unique and motivating environment was ideal for accomplishing my goals, and I am surely going to miss the discussions with my friends Costas, Vasilis D, and Vasilis T.

I am deeply thankful to Chara for encouraging and supporting me during the years of my studies. Finally, I would like to thank my parents and my sister for their unconditional love and support through ups and downs.

Funding support: This thesis has been produced under the framework of the research program "HERACLEITUS II: strengthening the human research potential through implementing doctoral research". I gratefully acknowledge co-funding support by the European Union (European Social Fund ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF).

Abstract

This thesis is focused on energy-efficient algorithms for job scheduling problems on speed-scalable processors, as well as on processors operating under a thermal and cooling mechanism, where, for a given budget of energy or a thermal threshold, the goal is to optimize a Quality of Service criterion. A part of our research concerns scheduling problems arising in large-data processing environments. In this context, we focus on the MapReduce paradigm and we consider problems of energy-efficient scheduling on multiple speed-scalable processors as well as classical scheduling on a set of unrelated processors.

First, we study the minimization of the maximum lateness of a set of jobs on a single speed-scalable processor. We consider two variants of the problem: a *budget variant*, where we aim in minimizing maximum lateness for a given budget of energy and an *aggregated variant*, where we want to minimize a linear combination of maximum lateness and energy. We propose optimal algorithms for both variants in the *non-preemptive* case where jobs have common release dates. Our algorithms are based on a number of structural properties that can be obtained after applying the KKT (Karush-Kuhn-Tucker) conditions on a convex programming formulation of the problem. In the presence of arbitrary release dates, we prove that both variants become strongly $\mathcal{NP}$-hard. Moreover, for the budget variant we show that it does not admit any $O(1)$-competitive deterministic algorithm, while for the aggregated variant we propose a 2-competitive online algorithm.

Then, we study energy-aware MapReduce scheduling where the goal is to minimize the total weighted completion time of a set of MapReduce jobs under a given budget of energy. We first propose a convex programming relaxation of the problem, when the execution order of jobs is known. We combine the solution of this relaxation with two natural list scheduling policies (First Come First Served and Highest Density First) and compare experimentally their effectiveness. Although their performance for random instances is fairly good, we prove that there are instances for which it is very far from the optimal. Next, we propose a linear programming approach which is based on an interval indexed LP-relaxation of the problem that incorporates a discretization of the possible speed values. Our algorithm transforms an optimal solution to this LP into a feasible solution for the problem by list scheduling in the order of tasks' α-points, where $\alpha \in (0, 1)$. We obtain a constant factor approximation algorithm for the total weighted completion time of a set of MapReduce jobs using energy augmentation. In the context of classical MapReduce scheduling (where energy

is not our concern) we also study the scheduling of a set of MapReduce jobs on unrelated processors with the goal of minimizing their total weighted completion time. We propose a 54-approximation algorithm which computes a feasible schedule by merging two individual schedules (of either Map or Reduce tasks) into a single schedule. Moreover, we consider the significant part of data shuffle in MapReduce applications and extend our model to capture the shuffle phase. We manage to keep the same ratio of 54 when the Shuffle tasks are scheduled on the same processors with the corresponding Reduce tasks, which becomes 81 when the Shuffle and the Reduce tasks are scheduled on different processors.

Finally, we focus on temperature-aware scheduling on a single processor that operates under a strict thermal threshold, where each job has its own heat contribution and the goal is to maximize the schedule's throughput. We consider the case of unit-length jobs with a common deadline and we revisit the offline CoolestFirst scheduling, i.e., the job with the smaller heat contribution is scheduled first. We study the approximability of Algorithm CoolestFirst and propose two different rounding schemes that yield lower bounds on its approximation factor. The first is based on a partition of the schedule according to the heat contributions of the jobs, while the second is based on a linear programming approach. The latter, which is actually more refined, yields a lower bound of at least 0.72.

Sommaire

Cette thèse se focalise sur des algorithmes efficaces en énergie pour des problèmes d'ordonnancement de tâches sur des processeurs de variation de vitesse ainsi que sur des processeurs fonctionnant sous un mécanisme de réchauffement-refroidissement, où, pour un budget d'énergie donné ou un seuil thermique, l'objectif consiste à optimiser un critère de Qualité de Service. Une partie de notre recherche concerne des problèmes d'ordonnancement de tâches apparaissant dans des environnements de traitement de grandes données. Dans ce contexte, nous nous focalisons sur le paradigme MapReduce et nous considérons des problèmes d'ordonnancement efficaces en énergie sur un ensemble de processeurs pouvant varier leur vitesse, ainsi que des problèmes d'ordonnancements classiques sur un ensemble des processeurs non-reliés. Premièrement, nous étudions la minimisation du retard maximal d'un ensemble de tâches sur un seul processeur de variation de vitesse. Nous considérons deux variantes de ce problème : la *variante budgétaire*, où nous voulons minimiser le retard maximal étant donné un budget d'énergie et la *variante agrégée*, où nous voulons minimiser une combinaison linéaire du retard maximal et de l'énergie maximale. Nous proposons des algorithmes optimaux pour ces deux variantes dans le cas non-préemptif où les tâches ont des dates de disponibilités communes. Nos algorithmes sont basés sur un nombre de propriétés structurales qui peuvent être obtenues en appliquant les conditions KKT (Karush-Kuhn-Tucker) sur une formulation de programmation convexe du problème. Nous prouvons que les deux variantes deviennent fortement $\mathcal{NP}$-difficile lorsque les tâches ont des dates de disponibilités arbitraires. En outre, nous montrons que la variante budgétaire n'admet aucun algorithme déterministe $O(1)$-compétitif, alors que pour la variante agrégée nous proposons un algorithme en ligne 2-compétitif.

Par la suite, nous étudions l'ordonnancement MapReduce où le but est de minimiser le temps d'achèvement pondéré d'un ensemble de tâches MapReduce étant donné un budget d'énergie : d'abord, nous proposons un programme convexe relâché de ce problème, où l'ordre d'exécution des travaux est connu. Nous combinons la solution de ce relâchement avec deux politiques naturelles de listes (First Come First Served et Highest Density First) et nous comparons leur efficacité expérimentalement. Malgré leur bonne performance pour le cas aléatoire, nous prouvons qu'il y a des cas pour lesquels elles sont loin de l'optimum. Deuxièmement, nous proposons une approche d'ordonnancement linéaire qui est basée sur un intervalle indexé LP-relâchement du problème qui incorpore une discrétisation des va-

leurs de vitesse possibles. Notre algorithme transforme une solution optimale de ce programme linéaire en une solution réalisable du problème en ordonnançant les tâches dans l'ordre défini par les α-points des tâches, où $\alpha \in (0, 1)$. Nous obtenons un algorithme d'approximation de facteur constant pour le temps de complétude pondéré total d'un ensemble de tâches MapReduce en utilisant une augmentation d'énergie. Dans le contexte d'ordonnancement MapReduce classique (où l'énergie n'est pas prise en compte) nous étudions aussi l'ordonnancement d'un ensemble des travaux MapReduce sur des processeurs non-reliés en minimisant la somme des temps de complétude pondéré. Nous proposons un algorithme 54-approché qui calcule un ordonnancement réalisable en fusionnant deux ordonnancements individuels (de tâches Map ou Reduce) dans un ordonnancement unique. En outre, nous considérons la partie principale de data shuffle dans des applications shuffle phase. Nous arrivons à garder le même rapport d'approximation de 54 lorsque les tâches Shuffle sont ordonnancées sur les mêmes processeurs avec les tâches correspondantes, et devient 81 quand les tâches Shuffle et Reduce sont ordonnancées sur des processeurs différents.

Enfin, nous nous focalisons sur l'ordonnancement sous contraintes thermiques sur un seul processeur fonctionnant en-dessous d'un seuil de température stricte où chaque tâche a sa propre contribution thermique et le but est de maximiser le nombre de tâche exécutée. Nous considérons le cas où les tâches ont des durées unitaires ayant la même date d'échéance et nous revisitons l'algorithme hors-ligne CoolestFirst, c'est-à-dire la tâche ayant la contribution thermique la plus petite est ordonnancée en premier. Nous étudions l'approximabilité de l'Algorithme CoolestFirst et proposons deux différents schémas d'arrondis qui produisent des bornes maximales sur son facteur d'approximation. Le premier est basé sur une partition de l'ordonnancement selon les contributions thermiques des tâches, tandis que le second est basé sur un programme linéaire. Celui-ci, qui est en effet plus raffiné, produit une borne minimale d'au moins 0.72.

Περίληψη

Η εργασία αυτή επικεντρώνεται σε ενεργειακά αποδοτικούς αλγόριθμους για προβλήματα χρονοδομολόγησης σε επεξεργαστές δυναμικής κλιμάκωσης της ταχύτητας, καθώς επίσης και σε επεξεργαστές οι οποίοι λειτουργούν κάτω από ένα μηχανισμό θέρμανσης και ψύξης, με στόχο την ελαχιστοποίηση ενός ποιοτικού κριτηρίου απόδοσης. Ένα σημαντικό μέρος της έρευνάς μας έχει ώς πρωταρχικό κίνητρο τη χρονοδρομολόγηση σε περιβάλλοντα επεξεργασίας μεγάλου όγκου δεδομένων. Σε αυτό το πλαίσιο, επικεντρωνόμαστε στο πρότυπο MapReduce και μελετάμε προβλήματα ενεργειακά αποδοτικής χρονοδρομολόγησης σε πολλαπλούς επεξεργαστές κλιμακούμενης ταχύτητας, καθώς επίσης και τυπικά προβλήματα χρονοδρομολόγησης σε μη σχετιζόμενους επεξεργαστές.

Αρχικά, προτείνουμε το πρόβλημα ελαχιστοποίησης της μέγιστης καθυστέρησης ενός συνόλου εργασιών σε μοναδικό επεξεργαστή κλιμακούμενης ταχύτητας. Μελετάμε δύο εκδοχές του προβλήματος: μία *εκδοχή δεδομένου προϋπολογισμού*, όπου ο στόχος είναι η ελαχιστοποίηση της μέγιστης καθυστέρησης για δεδομένο προϋπολογισμό ενέργειας και μία *συγκεντρωτική εκδοχή* όπου ο στόχος είναι η ελαχιστοποίηση ενός γραμμικού συνδυασμού της μέγιστης καθυστέρησης και της ενέργειας που καταναλώνεται. Προτείνουμε βέλτιστους αλγόριθμους πολυωνυμικού χρόνου για τις δύο εκδοχές στην περίπτωση που οι εργασίες διαθέτουν κοινούς χρόνους αποδέσμευσης. Οι προτεινόμενοι αλγόριθμοι βασίζονται σε ένα σύνολο δομικών χαρακτηριστικών της βέλτιστης χρονοδρομολόγησης, τα οποία εξάγονται με την εφαρμογή των KKT (Karush-Kuhn-Tucker) συνθηκών σε ένα κυρτό πρόγραμμα αντίστοιχο του προβλήματός μας. Στην περίπτωση που οι εργασίες διαθέτουν αυθαίρετους χρόνους αποδέσμευσης, αποδεικνύουμε ότι και οι δύο εκδοχές είναι strongly **NP**-hard. Επιπλέον, στην τελευταία περίπτωση, για την εκδοχή δεδομένου προϋπολογισμού δείχνουμε ότι δεν επιδέχεται ντετερμιστικό αλγόριθμο σταθερού λόγου ανταγωνισμού, ενώ για την συγκεντρωτική εκδοχή προτείνουμε έναν 2-ανταγωνιστικό αλγόριθμο.

Στη συνέχεια μελετάμε προβλήματα MapReduce χρονοδρομολόγησης με επίγνωση της ενέργειας και στόχο την ελαχιστοποίηση του συνολικού βεβαρημένου χρόνου ολοκλήρωσης ενός συνόλου MapReduce εργασιών, δεδομένου ενός προϋπολογισμού ενέργειας. Αρχικά προτείνουμε τη διατύπωση ενός χαλαρωμένου κυρτού προγράμματος για το πρόβλημα, δεδομένης μίας διάταξης εκτέλεσης των εργασιών. Συνδυάζουμε τη λύση της κυρτής χαλάρωσης με δύο συνήθεις στρατηγικές χρονοδρομολόγησης (First Come First Served and Highest Density First) και συγκρίνουμε πειραματικά τις λύσεις τους. Μολονότι η απόδοση τους για τυχαία

στιγμιότυπα είναι αρκετά καλή, όπως αποδεικνύουμε, υπάρχουν στιγμιότυπα για τα οποία αποκλίνει αρκετά από αυτή της βέλτιστη λύσης. Επομένως, προτείνουμε μία μεθόδευση γραμμικού προγραμματισμού, η οποία βασίζεται στη διατύπωση μίας γραμμικής χαλάρωσης του προβλήματος μέσω διακριτοποίησης τόσο του χρονικού ορίζοντα καθώς επίσης και των πιθανών τιμών των ταχυτήτων εκτέλεσης. Ο προτεινόμενος αλγόριθμος μετασχηματίζει μία βέλτιστη λύση της γραμμικής χαλάρωσης σε μία εφικτή λύση του προβλήματος εκτελώντας τις εργασίες βάσει της διάταξης που υποδεικνύεται από τα α-points, $\alpha \in (0, 1)$ των εργασιών στη λύση του γραμμικού προγράμματος. Πετυχαίνουμε έναν αλγόριθμο σταθερής προσέγγισης για το πρόβλημα ελαχιστοποίησης του συνολικού βεβαρημένου χρόνου ολοκλήρωσης ενός συνόλου MapReduce εργασιών, ο οποίος χρησιμοποιεί προσαύξηση της ενέργειας. Στο πλαίσιο της MapReduce χρονοδρομολόγησης, όταν η ενέργεια δεν αποτελεί στόχο, μελετάμε επίσης την πιο γενική περίπτωση χρονοδρομολόγησης ενός συνόλου MapReduce εργασιών σε μη σχετιζόμενους επεξεργαστές, με στόχο την ελαχιστοποίηση της συνολικής βεβαρημένης ολοκλήρωσής τους. Προτείνουμε έναν 54-προσεγγιστικό αλγόριθμο ο οποίος υπολογίζει μία εφικτή χρονοδρομολόγηση για το πρόβλημα, συνενώνοντας δύο ξεχωριστές χρονοδρομολογήσεις (για τις Map ή τις Reduce εργασίες) σε μία. Επιπλέον, επεκτείνουμε το μοντέλο μας ώστε να συμπεριλάβει μία επιπλέον σημαντική παράμετρο στις MapReduce εφαρμογές, αυτή του *data shuffle*. Επιτυγχάνουμε να διατηρήσουμε τον παράγοντα προσέγγισης ίσο με 54 στην περίπτωση που οι Shuffle εργασίες εκτελούνται στους ίδιους επεξεργαστές με τις Reduce εργασίες, ο οποίος γίνεται 84 στην περίπτωση που οι Shuffle και οι Reduce εργασίες εκτελούνται σε διαφορετικούς επεξεργαστές.

Τέλος, επικεντρωνόμαστε σε προβλήματα χρονοδρομολόγησης με επίγνωση της θερμοκρασίας, σε ένα μοναδικό επεξεργαστή που λειτουργεί βάσει ενός αυστηρού θερμικού κατωφλίου, όπου κάθε εργασία έχει τη δική της θερμική συνεισφορά και ο στόχος είναι η μεγιστοποίηση του throughput της χρονοδρομολόγησης. Μελετάμε την περίπτωση όπου οι εργασίες είναι μοναδιαίου μήκους και έχουν μία κοινή προθεσμία και επανεξετάζουμε μία κλάση προβλημάτων όπου μία εργασία δεν εκτελείται εφόσον μία άλλη, μικρότερης θερμικής συνεισφοράς ή προθεσμίας, έχει αποδεσμευτεί και είναι διαθέσιμη και επικεντρωνόμαστε στη μεγιστοποίηση του throughput στην offline εκδοχή του προβλήματος, κάτω από την CoolestFirst χρονοδρομολόγηση. Αναλύουμε την προσεγγισιμότητα του αλγόριθμου CoolestFirst και προτείνουμε δύο διαφορετικά σχήματα στρογγυλοποίησης. Το πρώτο βασίζεται στη διαμέριση της χρονοδρομολόγησης σύμφωνα με τις θερμικές συνεισφορές των εργασιών, ενώ το δεύτερο σε μία θεώρηση μέσω γραμμικού προγραμματισμού. Το δεύτερο σχήμα βελτιώνει το πρώτο και δίνει ένα κάτω φράγμα μεγαλύτερο ή ίσο του 0.72.

Contents

Chapter 1

Introduction

The exponential growth in power consumption and the widespread use of computing devices over the last decade, have rendered energy saving as a major concern today in terms of both cost and availability in computing systems. For instance, energy consumption dominates the operating cost of large data centers. As noted in [68] the cost of energy for data-center operators may constitute half of their total cost of ownership. Moreover, mainly due to the extensive use of mobiles, tablets, etc. [1] as well as the use of sensor devices [2] that are battery operated, limiting energy consumption has become a major challenge in their design to extend batteries' lifetime. Furthermore, as a major part of energy consumption is converted into heat, there is an exponential rise in heat density causing difficulties in cooling microprocessor chips (as processor's speed increases, so does the heat that is generated). This reduces the reliability and increases the manufacturing costs of the hardware components resulting in an urgent need for regulating the operating temperature of processor systems.

Energy and temperature management have been extensively studied by computer engineers at hardware and system design level (see e.g., [102, 34]). In fact, although similar computational models (Random Access Machine, or Turing Machine) and general scientific approaches are used to study computational resources of both time and space, the physics of energy is not captured by them. So, completely different mechanisms, oriented to energy management, have been proposed to study energy and temperature as a computational resource, aiming to reduce the energy consumption and the temperature of a device. In such mechanisms the question generally addressed is the trade-off between the conflicting objectives of performance and energy-efficiency, the more power is available, the better performance can be achieved. However, as noted in [45], despite the advances in battery technology and in low-power micro-architecture design, it seems impossible to meet the energy needs of future computing devices. So, an interesting direction for research is to

1. Total mobile subscriptions are expected to grow from 6.8 billion in 2014 to 9.2 billion by the end of 2019 [46].

2. In sensor networks, where charging the battery is difficult, there is a great interest in the development of low-cost and low-power sensor devices (see e.g., SmartDust [106]).

address energy and temperature management at the higher levels of Operating Systems and applications.

In this context, over the last decade the goal of energy-efficiency gave rise to challenging algorithmic problems involving the management/optimization of energy and temperature as resources. Some reasonable questions that arise when designing algorithms for such problems are, what policy should the Operating System use to save energy (resp. heat dissipation)? It seems natural, therefore, to address energy and temperature management problems as *scheduling problems*, motivated by the allocation of limited resources (e.g., processors, energy, temperature) to a set of activities (e.g., computer applications/jobs). In general, the goal of a scheduler is to find an allocation that optimizes some objective function, which represents a Quality of Service (QoS) criterion. Some common QoS criteria for scheduling problems in computing environments are: (i) the makespan, which is the time where all jobs are completed (ii) the maximum lateness, which quantifies the failure of each job meeting its deadline, (iii) the throughput, which corresponds to the number of jobs that finish their execution on time and (iv) the total completion (or finishing) time of a set of jobs in a schedule. The main research objective of this thesis is to develop efficient algorithms for deterministic scheduling problems arising in energy-bounded and temperature-bounded computation, with the goal of optimizing some of the above QoS criteria.

1.1 Energy and temperature management

Various mechanisms have been proposed in algorithmic research for energy and temperature management. We describe here in details the two mechanisms on which our research is focused and at the end of this section we give a brief review of other mechanisms.

Speed scaling mechanism. A standard way to handle energy consumption is through Dynamic Voltage and Frequency Scaling (DVFS) of a processor. In this setting, also known as *speed scaling*, the processor can run at variable speeds based on demand and performance constraints. For instance, speed scaling is applied to Intel processors through the "Turbo Boost" technology, while on AMD processors it is achieved with the "PowerNow" technology. According to the well-known cube-root rule for the CMOS devices the speed of a processor is proportional to cube-root of the power [34]. Algorithms for these problems involve increasing power when the improvement in performance justifies the increased energy or temperature.

The theoretical study of speed scaling was initiated in a seminal paper by Yao et al. [111]. The authors proposed to formulate the speed scaling problems as scheduling problems and, by generalizing on the cube-root rule of CMOS devices they assumed that if a processor runs at speed s then the power consumption is s^β, where $\beta > 1$ is a processor dependent constant. Actually, β is in the range $(1, 3]$ for essentially all technologies [34, 107] (e.g., for

Intel PXA 270 is equal to 1.1, for Pentium M770 1.6 and for a TCP offload engine 1.66). A key fact of this power function is that it is a strictly convex function [3] of speed. Since energy consumption of the processor is power integrated over time (see Figure 1.1(i)), this intuitively means that high processor's speed implies high performance (with respect to some QoS criterion) at the price of high energy consumption, while lower speeds can save energy but performance degrades.

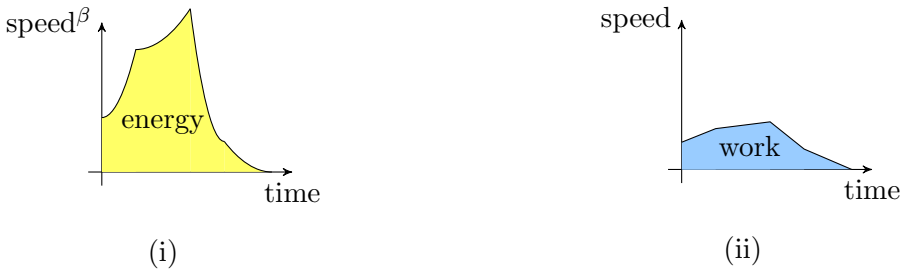


Figure 1.1: (i) The energy consumption over time and (ii) the work volume accomplished by a job.

The initial problem studied by Yao et al. [111] concerns a set of jobs, each one associated with an amount of work (CPU cycles), that must be accomplished in order to be completed (see Figure 1.1(ii)), a release date and a deadline. The goal is to schedule the jobs into a single speed-scalable processor, in order to minimize the total energy consumption while respecting the deadline feasibility QoS criterion, i.e., each job must be completed by its end-time. Moreover, during its execution, a job may be interrupted (preempted) and resumed later.

In the same context, where speed scaling is used to minimize the energy consumption, two different models have been also proposed: (i) the *bounded speeds model*, where the processors' speeds are bounded above and below (see e.g., [37]), and (ii) the *discrete speeds model*, where the processors' speeds are chosen among a set of discrete speeds (see e.g., [79]).

The problems studied in this thesis (in terms of energy management) adopt the speed scaling model proposed by Yao et al. [111] for different objective functions, as it will be described in the following sections. For an extensive review on speed scaling scheduling, the reader is referred to the survey of Albers [6].

Thermal and cooling mechanism. A different kind of mechanisms that form a critical research topic, concern the temperature management in computer systems. In terms of thermal behavior the proposed mechanisms are motivated by the fact that the processors

3. A function is *convex* if for any two points of its curve their line segment lies above or on the curve and is *strictly convex* if the line segment between any two points lies strictly above the curve (except if there are endpoints).

operate so as to avoid exceeding a *thermal threshold*, i.e., the maximum safe operating temperature; the violation of such a threshold reduces the lifetime or even damages the processors. In fact, the temperature of a processor is dynamically controlled by the hardware dynamic thermal management system, which automatically reduces the processor's speed when the thermal threshold is violated. Furthermore, concerning the running jobs, some of them might be more CPU-intensive, bearing more heat contribution than others. Thus, by considering the order of their execution, the thermal behavior of the processor (and so its performance) may vary. A significant part of the proposed work in this context deals with single or multi-core systems, where the heat contribution of jobs is varying, and the system has to decide an ordering of the jobs to the processor(s) so as to improve the thermal behavior and, consequently, the system's performance (see e.g., [44, 54, 71, 110]).

Based on the latter results, Chrobak et al. [41] stimulated the theoretical study of temperature-aware scheduling problems that aim to model the thermal and cooling management of processors. In their model they consider unit-length jobs, each one representing a unit portion of each job waiting to be processed by the system. Each job is assumed to have its own heat contribution, representing the increase in the processor's temperature, a release date and a deadline, and is going to be scheduled on a single processor operating under a strict thermal threshold. They assume that the heat contribution of each job is known in advance. This is not exactly the case in practice, but approximate heat contributions can be determined by well established prediction methods (see [110]). Moreover, the ambient temperature is assumed to be equal to zero and the thermal behavior of a processor (that depends on technical characteristics) is modeled by a constant $c > 0$, so called *cooling factor*. Finally, the processor runs at constant speed, and the scheduler can leave an idle time unit whenever the execution of any available job violates the thermal threshold.

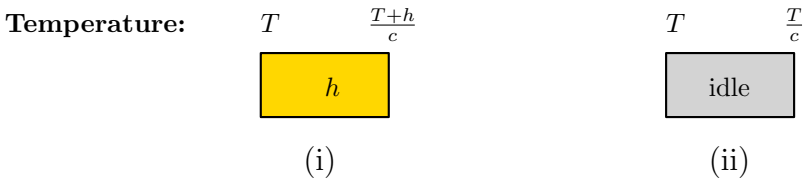


Figure 1.2: The thermal and cooling mechanism of a processor during (i) the execution of a unit-length job of heat contribution h and (ii) an idle unit-time slot.

The thermal and cooling mechanism of a processor is performed in a *geometric* manner (i.e., geometric increase or decrease of the temperature) as follows. Let Θ be the thermal threshold and T the current temperature of the processor. If a job, with heat contribution h, is allowed to be executed then, the processor's temperature after its execution becomes equal to $\frac{T+h}{c}$, while it holds that $\frac{T+h}{c} \leq \Theta$ (see Figure 1.2(i)). Moreover, when the processor is *idle*, i.e., no job is executed, the temperature becomes equal to $\frac{T}{c}$, as if a job of heat

contribution equal to zero is executed during a unit-time slot (see Figure 1.2.(ii)). The goal is to compute a schedule which maximizes the *throughput*, i.e., the number of tasks that meet their deadlines.

Other mechanisms. A common mechanism for saving energy is the Dynamic Power Management (DPM), also called *power-down*, where the device can always reside in one of several states, with individual power consumption rates. In addition to the active state there can be, for instance, standby, suspend, sleep, and full-off states. In practice, the BIOS of most computers includes the Advanced Configuration and Power Interface (ACPI) that provides five states, including standby and hibernation. However, altering a device between different states involve some delay and the expenditure of energy. The critical research issue in algorithmic power-down problems is to improve the delay and energy cost of these transitions with the energy savings. For an excellent survey on power-down scheduling problems the reader is referred to [5]. Usually in practice both power-down and speed scaling mechanisms are applied in order to reduce the energy consumption in computing devices (see e.g. [16]) thus, it seems realistic to study problems that combine both. From an algorithmic point of view this interaction raises questions that are even absent from both mechanisms. For instance, it is not always beneficial to run jobs with the smallest possible speed that allows to meet their deadlines, since a higher speed may create an idle period in which the system may transit into a lower-energy state. The theoretical study of such mechanisms was introduced by Irani et al. [65].

Finally, in terms of temperature management, a different approach was proposed by Bansal et al. [25], based on the *Newton's law of cooling* i.e., the assumption that the rate of heat loss of a processor is proportional to the difference between its temperature and the ambient temperature.

1.2 Algorithmic problems and tools

As already mentioned, the scope of this thesis concerns deterministic scheduling problems arising when we introduce energy and temperature constraints as an input to our problem. In fact, these are scheduling problems of two criteria, where the general goal is to determine a scheduling policy that optimizes some QoS criterion (e.g., lateness, throughput, total completion time) of cost $\mathcal{Q}$, while simultaneously minimizes the total energy consumption (resp. temperature), let $\mathcal{E}$ its cost. As these two criteria are in opposition, e.g., the more energy available the better QoS produced, there are four different optimization problems that arise.

O1: The first one, is to optimize $\mathcal{Q}$ while bounding the cost of $\mathcal{E}$. This bound corresponds to an available energy budget E or a thermal threshold Θ.

O2: Symmetrically with the first, the second one is to bound $\mathcal{Q}$ and then to optimize $\mathcal{E}$.

O3: The third one is to minimize their sum, $\mathcal{Q}+\mathcal{E}$, or a weighted combination of the two costs that expresses their relative value in the total cost i.e., $\mathcal{Q} + \lambda\mathcal{E}$, where $\lambda \in \Re^+$.

O4: The fourth one, is to identify the set of Pareto optimal schedules (points) ($\mathcal{Q}$,$\mathcal{E}$). A schedule S with $\mathcal{Q} = \mathcal{Q}(S)$ and $\mathcal{E} = \mathcal{E}(S)$ is called *Pareto optimal* if there is no schedule S' such that $\mathcal{Q}(S') \leq \mathcal{Q}(S)$, if $\mathcal{Q}$ is to be minimized (resp. $\mathcal{Q}(S') \geq \mathcal{Q}(S)$) and $\mathcal{E}(S') \leq \mathcal{E}(S)$, with at least one of the two inequalities being strict.

Motivated by applications where computing devices operating under a limited amount of available energy, we study problems with objectives of the form O1 and O3. More specifically, taking advantage of the wide variety of algorithmic techniques from scheduling theory, we propose offline and online algorithms with provably good performance guarantees on the quality of their solutions. It is clear that solving problems of the form O4 also solves the problems of forms O1-O3.

Before proceeding to the contributions of the thesis, we will give a brief description of the tools used for the evaluation of our algorithms' performance as well as the necessary terminology from scheduling theory.

1.2.1 Complexity and performance guarantees

The approach followed in this thesis is the design of *efficient algorithms* regarding the new field of energy (resp. temperature)-aware scheduling problems. The term efficient algorithm corresponds to an algorithm of polynomial number of steps compared to the size of the problem input, also called *polynomial time algorithm*. However, as in all algorithm oriented areas (e.g., combinatorial optimization, theoretical computer science) most of the problems of practical interest are computationally intractable.

The classification of computational problems in *easy* and *hard* is the object of study of computational complexity theory, initiated in the early 70's. A computational problem is easy to solve, if there is an efficient algorithm for it and the complexity class that contains all easy problems is the class P. The hardness of a computational problem is formalized in a class called $\mathcal{NP}$. Roughly speaking $\mathcal{NP}$ contains only decision problems (with an output either yes or no) such that each yes instance I has a polynomially bounded (on the size of I) certificate that can verify I in polynomial time. In order to provide a precise notion of what it means for a problem to be at least as hard as another, the concept of *reduction* has been proposed. We say that a problem A reduces to B, or equivalently B is at least as hard as A, if there exists a function f which, for every instance I of A, produces an equivalent instance $f(I)$ for B. It is important to note that reductions must be polynomial time algorithms, which means that the function f can be computed in polynomial number of steps.

A problem A is $\mathcal{NP}$-hard if, for each problem $A' \in \mathcal{NP}$, A' reduces to A. If moreover, $A \in \mathcal{NP}$, then we say that it is $\mathcal{NP}$-complete. The notion of completeness of a problem in $\mathcal{NP}$ means that, if there is a polynomial algorithm for it then, through a polynomial

reduction, we should have a polynomial time algorithm for any problem in $\mathcal{NP}$, which proves that $P = \mathcal{NP}$. We call strongly $\mathcal{NP}$-hard a problem that remains $\mathcal{NP}$-hard when its input values are bounded by a polynomial of the input. For more details concerning the class $\mathcal{NP}$ as well as the computational complexity theory, the reader is referred to the books of Garey and Johnson [51] and Papadimitriou [85].

Scheduling problems form a vast sub-area of optimization problems. An *optimization problem* is specified by a set of instances and a non-empty set of feasible solutions for them, as well as an objective function mapping every feasible solution to an objective cost. Optimization problems are distinguished to minimization problems, where the optimal solution is the feasible solution of the minimum cost, and maximization problems where the optimal solution is the feasible solution of maximum cost. We will define by $\mathrm{OPT}(I)$ the optimal objective cost of a feasible solution to a problem instance I. Since for most of the practical optimization problems it is $\mathcal{NP}$-hard to find an optimal solution, one way to overcome this difficulty, at the expense of reducing the quality of the solution, is to design approximation, instead of optimal algorithms.

Definition 1.1 *Consider a minimization (resp. maximization) problem* Π *and a positive value* $\rho \in \mathbb{R}^+$*, where* $\rho \geq 1$*. An algorithm* A *is called* ρ*-approximation for problem* Π*, if, for each problem instance* I*, it finds a feasible solution of cost* $c_A(I)$*, such that,*

- $c_A(I) \leq \rho \cdot OPT(I)$*, if* Π *is a minimization problem.*
- $OPT(I) \leq \rho \cdot c_A(I)$*, if* Π *is a maximization problem.*

The value ρ *is called approximation ratio (or factor) of algorithm* A*. Moreover,* A *is a polynomial time algorithm.*

The best that we can expect for an $\mathcal{NP}$-hard minimization (resp. maximization) problem is an *approximation scheme*, i.e., a family of algorithms, which, for each fixed constant $\epsilon > 0$, it computes a $(1+\epsilon)$-approximate (resp. $(1-\epsilon)$-approximate) solution to the problem. If the time complexity of this scheme is polynomial on the input size, then it is called polynomial time approximation scheme (PTAS) and if it is also polynomial on $\frac{1}{\epsilon}$, then it is called fully polynomial time approximation scheme (FPTAS). Moreover, if the time complexity is of the order of $O(n^{\mathrm{poly}\log(n)})$, the scheme is called quasi-polynomial time approximation scheme or QPTAS.

For an extensive study on approximation algorithms for many important combinatorial optimization problems, the reader is referred to the books [108, 105].

Online algorithms. Studying the computational complexity of algorithms in the offline setting, i.e., when the algorithm is aware of the whole input in advance, and more specifically designing polynomial time approximation algorithms for $\mathcal{NP}$-hard problems, it is useful to guarantee the quality of the algorithm's solution compared to the optimal one. However, in the online case, i.e., when not all relevant input data are available, but revealed as the computation progresses, we are interested in the ratio between our algorithm's performance

on a problem instance and the offline optimum for this instance. This concrete measure of quality of online algorithms is called *competitive ratio* [49] and formally is defined as follows.

Definition 1.2 *Consider a minimization problem (resp. maximization) problem* Π *and let* $c_A(I)$ *denote the cost of algorithm* A *on problem instance* I*. We define the competitive ratio of an algorithm* A *for problem* Π *to be the value* σ*, such that, for each problem instance* I*,*

- $c_A(I) \leq \sigma \cdot OPT(I)$*, if* Π *is a minimization problem.*
- $OPT(I) \leq \sigma \cdot c_A(I)$*, if* Π *is a maximization problem.*

1.2.2 Scheduling terminology

Since the early 50's [67] scheduling problems have formed a vast sub-area of combinatorial optimization. Many scheduling problems have been studied while new algorithmic techniques that have been devised to tackle them have influenced research on different fields e.g., Data Bases, Computer Networks, Operations Research. Given the magnitude of scheduling theory, next we describe only the concepts relevant to this thesis. An explicit overview of scheduling theory can be found in the books [75, 87].

A scheduling problem involves a set $\mathcal{J}$ of n jobs that are going to be executed on a set $\mathcal{P}$ of m processors. The execution of each job $j \in \mathcal{J}$ requires a certain processing volume. In the speed scaling setting this processing volume corresponds to the number of CPU cycles required by the job and is called *work volume* u_j, while in the classical scheduling setting, where processors run at unit speed, it corresponds to the actual *processing time* of the job in the schedule and is denoted by p_j.

Jobs in set $\mathcal{J}$ may also subject to *constraints* that should be taken into account for their execution. For instance, *precedence constraints* between jobs (i.e., a fixed partial order of the jobs) might be given as an input to the scheduling problem, represented by a directed acyclic graph $G = (V, A)$, where V corresponds to the jobs and $(j, k) \in A$ if and only if j must be completed before k can start its execution (see in Figure 1.4(i)). Moreover, *preemption* might be allowed, i.e., the execution of a job can be interrupted and resumed at later time. A job $j \in \mathcal{J}$ might be also associated with a *release date* r_j before which it cannot start its execution, a *weight* w_j that represents its importance with respect to other jobs and a *due date* (or deadline) d_j by which it should complete its execution.

In general, the *processor environment* of most scheduling problems can be either a single processor, in which all jobs have to be processed, or multiple processors, where each job can be processed on any of the processors (each processor can execute at most one job at a time). Two common multi-processor environments are (i) *identical processors*: each job $j \in \mathcal{J}$ has the same work volume u_j on every processor, and (ii) *unrelated processors*: each job j is given a vector of work volumes $(u_{i,j})$, $i \in \mathcal{P}$. Note that, in the classical scheduling setting, where processors run at unit speed, we have that, in (i) the work volume of each job is equal to its processing time, i.e., for each $j \in \mathcal{J}$, $u_j = p_j$, while in (ii) for each job

$j \in \mathcal{J}$ and each processor $i \in \mathcal{P}$, $u_{i,j} = p_{i,j}$.

In many practical scheduling problems each job consists of a fixed number of *tasks* (or operations). Each task is associated with a processing volume by which it contributes to the completion of the job. A common multi-task scheduling model is the *flow-shop*, where an ordered set of m processors, $\{1, 2, \ldots, m\}$, is given and each job j consists of m different tasks $T_{i,j}, i = 1, 2 \ldots, m$, that are subject to the precedence constraints, represented by a directed graph $G = (V, A)$, where V corresponds to the set of all tasks and $A = \{(T_{i,j}, T_{i+1,j}) \mid j \in \mathcal{J}, i = 1, 2, \ldots, m-1\}$. Each task $T_{i,j}$ is going to be scheduled non-preemptively on the corresponding processor i.

An interesting generalization of the flow shop problem is the *multi-stage flexible flow-shop* environment: There are k stages, each job consists of k tasks and each task can be scheduled on a set of parallel processors (e.g., identical, unrelated, etc.). The processors at each stage might be indistinguishable and the tasks of each job have to be scheduled in the order indicated by the stages (from 1 to k). A processor can execute at most one task at a time, while preemption of tasks is not allowed.

A flow-shop model in which there are no precedence constraints among the tasks of each job is defined as the *open-shop* model. Moreover, an important version of the open-shop model, where the tasks of the same job can be processed concurrently, i.e., different processors are allowed to execute operations of the same job at the same time, is called *concurrent open-shop*.

In this thesis, we are going to study scheduling problems on single and parallel processors that operate under the speed scaling or thermal and cooling mechanisms, as described in Section 1.1.

When scheduling a set of n jobs $\mathcal{J}$ to a set processors, the goal is either to minimize or maximize the value of an objective function. Some typical objective functions are the following:

- *Makespan*: represents the time when the last task of a schedule finishes its execution and is denoted by $C_{\max} = \max_{j \in \mathcal{J}}\{C_j\}$, where $C_j,\ j = 1, 2, \ldots, n$, is the completion time of job j in the schedule. In the schedule of Figure 1.3(ii) we have that $C_{\max} = 32$.
- *Total (or average) weighted completion time*: it is denoted by $\sum_{j \in \mathcal{J}} w_j C_j$. In the schedule of Figure 1.3(ii), if we assume that jobs have unit weights, then $\sum_{j \in \mathcal{J}} C_j = 90$.
- *Maximum lateness*: the lateness of a job j in a schedule is the difference between its completion time C_j minus its due date d_j and is denoted by $L_j = C_j - d_j$. In fact, if a job completes before its due date, its lateness can be negative. The maximum lateness over all jobs in a schedule is denoted by $L_{\max}$. In the schedule of Figure 1.3(ii), $L_{\max} = 12$, attained by job 2. In the schedule of Figure 1.3(iii) the maximum lateness is again attained by job 2, but is equal to $L_{\max} = 1$.
- *Throughput*: is the number of jobs that complete their execution before their deadline.

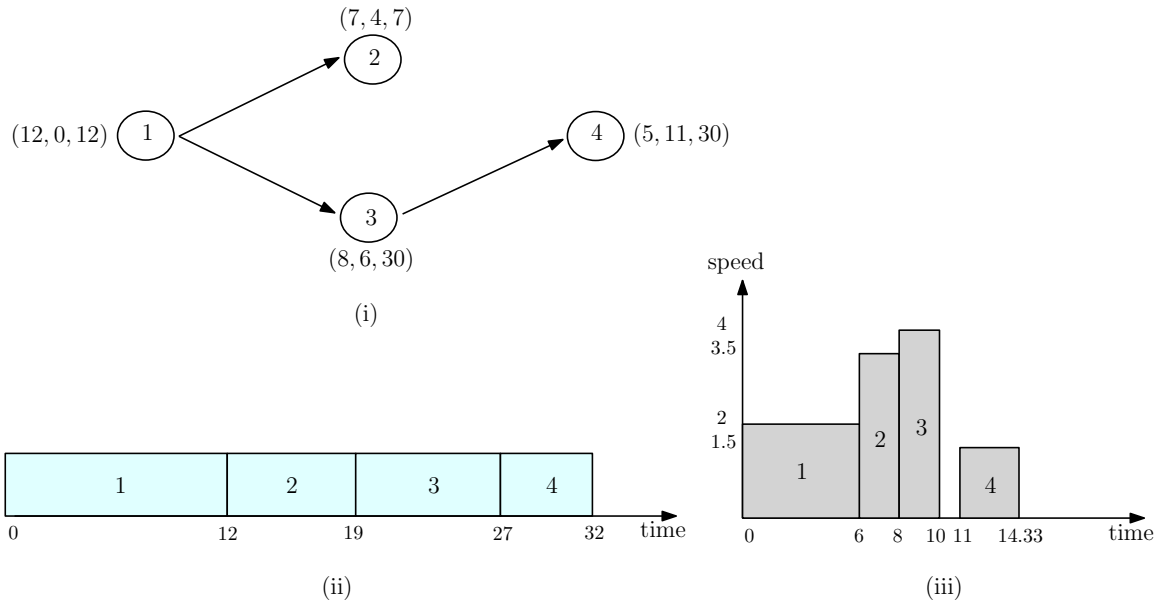


Figure 1.3: (i) The precedence graph of an instance of four jobs. Each job j is specified by an ordered triple, (v_j, r_j, d_j), where v_j is its work volume, r_j its release date and d_j its due date. (ii) A schedule of the jobs on a single processor. The jobs' work volumes v_j correspond to their actual processing times p_j and their completion times are displayed on the time axis. (iii) A schedule of the jobs on a single speed-scalable processor. Both the jobs' completion times and the speeds are displayed on the time and speed axes, respectively.

In the schedule of Figure 1.3(ii), jobs 1 and 3 are completed before their deadlines, so the throughput is equal to 2. In the schedule of Figure 1.3(iii), all jobs except job 2 are completed before their deadlines and the throughput equals 3.

Generally speaking, a schedule computed by an algorithm is *feasible* for an objective, if all the constraints and properties posed by the job and the processor environments are satisfied.

Concerning the speed scaling setting, an important note when designing a schedule is that except for which job to execute, the scheduler must also determine at what speed the job should be executed. Clearly, this additional requirement makes speed scaling scheduling more complex, compared with classical scheduling. For example, consider the two schedules illustrated in Figure 1.3(ii)-(iii) with respect to the total completion time QoS criterion. In the classical scheduling setting, where the processor runs at unit speed and energy is not our concern, Figure 1.3(ii) represents a feasible schedule of the instance in Figure 1.3(i) – respecting the release dates and the precedences among the jobs – that attains total completion time equal to 90. However, in the speed scaling setting, as we can see from the schedule depicted in Figure 1.3(iii), a job j is specified by both its processing time p_j ($p_1 = 6, p_2 = 2, p_3 = 2, p_4 = 10/3$) and its speed s_j ($s_1 = 2, s_2 = 3.5, s_3 = 4, s_4 = 1.5$). Actually, the volume $p_j \cdot s_j$ corresponds to the work volume of job j. To be more precise, assuming that each job runs at a single constant speed during its execution is completely

reasonable. As we will see in Chapters 2 and 3, this is actually a key-fact in many speed scaling scheduling problems and is based in the convexity of the power function. Thus, the total energy consumption E, although it is power integrated over time, can be computed as the following sum $E = \sum_{j \in \mathcal{J}} p_j s_j^{\beta}$, where $\beta > 1$ is the processor dependent constant. Let us assume that $\beta = 2$. Thus, as we can compute, the whole schedule consumes 88 units of energy and the total completion time is equal to $\sum_{j \in \mathcal{J}} C_j = 38.33$.

In terms of energy (resp.temperature)-aware scheduling, the above typical objective functions can be further extended with the total energy consumption and thermal threshold measures, and formulated as one of the objectives O1-O4 in Section 1.2. Consider again for example the instance of Figure 1.3(i), for the problem of minimizing the total completion time, in a speed-scalable processor, for a given energy budget $E = 88$. Then, the schedule in Figure 1.3(iii) is feasible, with total energy consumption equal to E and $\sum_{j \in \mathcal{J}} C_j = 38.33$.

Moreover, consider the problem of maximizing throughput, for a given thermal threshold Θ, in a processor equipped with the thermal and cooling mechanism. Clearly, the schedule depicted in Figure 1.4, for $\Theta = 1$, is a feasible schedule that attains throughput equal to 4. Note that, job 3 is not scheduled at its release date, since it would violate the thermal threshold Θ, so the processor remains idle while reducing its temperature by half and allowing job 3 to be executed at the next time instant.

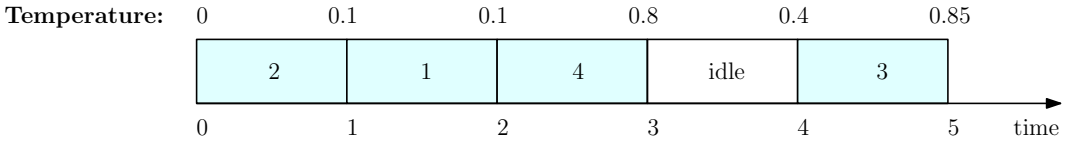


Figure 1.4: A feasible schedule of four jobs on a processor equipped with the thermal and cooling mechanism, for a thermal threshold $\Theta = 1$. Each job j is specified by an ordered triple (h_j, r_j, d_j), where h_j is its heat contribution, r_j its release date and d_j its deadline. The instance comprises of jobs: $1 \rightarrow (0.1, 0, 2)$, $2 \rightarrow (0.2, 0, 2)$, $3 \rightarrow (1.3, 3, 5)$, $4 \rightarrow (1.5, 2, 3)$. The initial temperature is zero.

1.2.3 An application: The MapReduce paradigm

Energy consumption and cooling of data centers dominate their operational cost while posing significant limitations in terms of efficiency. Moreover, an increasingly larger amount of processing in data centers is managed today by distributed platforms for parallel processing of large data sets. A standard programming model for implementing massive parallelism in large data centers is the MapReduce paradigm [43]. Applications of MapReduce such as search indexing, web analytics and data mining, involve the concurrent execution of several MapReduce jobs on a system like Google's MapReduce [43] or Apache Hadoop [89]. Several empirical works have been carried-out focusing on improving the energy-efficiency in MapReduce and especially for data processing in the Hadoop framework. Most of this

work is mainly based on the power-down mechanism [78, 47, 48, 53]. However, as Wirtz et al., [109] recently showed, for some computation intensive MapReduce applications the use of intelligent speed scaling may lead to significant energy savings. In this thesis, we focus on the theoretical study of energy-aware scheduling of MapReduce computations in the speed scaling setting.

When a MapReduce computation (or MapReduce job) is executed a number of Map and Reduce tasks are created. Each Map task operates on a portion of the input elements, translating them into a number of key-value pairs. After an intermediate process, all the key-value pairs having the same key are transmitted to the same Reduce task. The Reduce tasks operate on the key-value pairs, combine the values associated with a key, and generate the final result. Figure 1.5 illustrates the main parts during the execution of a MapReduce job. Note that, the time for transmitting the intermediate data from Map to Reduce tasks (communication cost) is a significant part of the processing cost in MapReduce applications, called *data shuffle*.

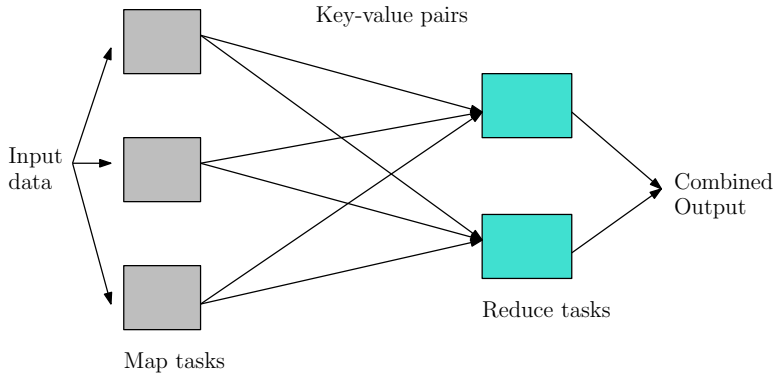


Figure 1.5: The structure of a MapReduce job

In addition to the many practical applications of MapReduce, there has been a significant interest in developing appropriate cost models and a computational complexity theory for MapReduce computation (see e.g., [4, 69]), in order to understand the basic principles underlying the design of efficient MapReduce algorithms (see e.g., [2, 72]), and to obtain upper and lower bounds on the performance of MapReduce algorithms for some fundamental computational problems (see e.g. [3] and the references therein).

Many important advantages of MapReduce are due to the fact that the Map tasks or the Reduce tasks can be executed in parallel and essentially independently from each other. However, to best exploit massive parallelism available in typical MapReduce systems, one has to carefully allocate and schedule Map and Reduce tasks to actual processors (or computational resources, in general). This important and delicate task is performed in a centralized manner, by a process running in the master node. A major concern of the scheduler, among others, is to satisfy task dependencies within the tasks of the same

MapReduce job; *all the Map tasks must finish before the execution of any Reduce task of the same job*. A MapReduce job is completed when the last of its reduce tasks finishes its execution. Moreover, during the assignment and scheduling process, a number of different needs must be taken into account, e.g, data shuffle, data locality (i.e., executing tasks on the node that stores the data), skew (i.e., highly variable task runtimes) which gives rise to the study of new challenging *MapReduce scheduling problems*.

Despite the importance and the challenging nature of scheduling in MapReduce environments, and despite the extensive investigation of a large variety of scheduling problems in parallel computing systems (see e.g., [87]), less attention has been paid to MapReduce scheduling problems. In fact, most of the previous work on scheduling in MapReduce systems concerns the experimental evaluation of scheduling heuristics, mostly from the viewpoint of finding good trade-offs between different objectives (see e.g., [112] and the references therein). From a theoretical viewpoint, only few results on MapReduce scheduling have appeared so far [83, 38, 40]. These are based on simplified abstractions of MapReduce scheduling, closely-related to the concurrent open-shop and the two-stage flexible flow-shop scheduling models, that capture issues such as task dependencies, shuffle, and task assignment, under the objective of minimizing the total weighted completion time of a set of MapReduce jobs. This is actually a natural objective since large MapReduce data clusters are usualy shared among several users. Under this context, we initiate the study of energy management in MapReduce scheduling on speed-scalable parallel processors, with the goal to minimize the total weighted completion time of a set of MapReduce jobs with respect to a given energy budget. Furthermore, when energy management is not a concern, we focus on generalizations of the theoretical framework proposed so far, incorporating important needs, such as data locality and data shuffle, in the scheduling process of the MapReduce jobs.

1.3 Outline of the thesis

This thesis is focused on energy-efficient algorithms for scheduling problems, under the speed scaling and the thermal and cooling mechanisms, where, for a given energy budget or thermal threshold, the goal is to find schedules, which maximize a QoS criterion. Three different QoS criteria are studied: maximum lateness, total weighted completion time and throughput. New challenging problems, motivated by the MapReduce paradigm, are also considered, in terms of both energy-bounded scheduling on parallel speed-scalable processors as well as classical scheduling on a set of unrelated processors. An organisation of the thesis results is described below.

Speed scaling for maximum lateness. In Chapter 2, we initiate the study of the problem of scheduling a set of jobs, each one associated with a release date, a due date and a work volume, to be executed non-preemptively on a single speed-scalable processor in

order to minimize the maximum lateness. In their seminal paper, Yao et al. [111], considered the problem of minimizing energy consumption on a single speed-scalable processor, while setting the maximum lateness to zero. As maximum lateness minimization and energy savings are conflicting objectives, we consider two variants: i) a *budget variant*, where we aim in minimizing maximum lateness for a given energy budget and ii) an *aggregated variant*, where our objective is the minimization of a linear combination of maximum lateness and energy, i.e., $L_{\max} + \lambda E,\ \lambda \in \Re^+$. We first study the case where all jobs are realeased at time zero and by applying the well-known KKT (Karush, Kuhn, Tucker) conditions on a convex programming formulation for each problem variant, we obtain a number of structural properties of an optimal solution and we propose optimal greedy algorithms for the *non-preemptive* single processor case with common release dates.

In the presence of arbitrary release dates, we prove that both problem variants become strongly $\mathcal{NP}$-hard. In addition, we turn our attention to the online case and for the budget variant we prove that it does not admit any $O(1)$-competitive deterministic algorithm, which is actually expected (see Theorem 10 in Bansal et al. [26]). Instead, for the aggregated variant we propose a 2-competitive online algorithm.

Speed scaling for MapReduce jobs. A known variant of the standard open-shop problem, called concurrent open-shop (see e.g. [80]), is closely-related to an abstract model of MapReduce scheduling proposed in [38, 40]. Indeed, a MapReduce job consists of a set of Map tasks and a set of Reduce tasks that can be executed simultaneously. However, this should be done provided that all Map tasks must finish before the execution of any Reduce task of the same job, while both the Map and Reduce tasks can be executed in parallel. In Chapter 3, we study the problem of scheduling a set of MapReduce jobs, where each task is associated with a positive work volume and each job has a single weight and a single release date. The tasks are preassigned on a set of speed-scalable parallel processors and the goal is to minimize the total weighted completion time of jobs. We propose a convex programming relaxation of our problem, when an order of the jobs is prespecified and we design greedy heuristics that combine the solution of this convex relaxation with two natural list scheduling policies: (i) First Come First Served and (ii) Highest Density First, and compare experimentally their effectiveness. As we prove, although these heuristics behave well for reasonable random instances, in terms of worst-case analysis their approximation ratio is proved to be arbitrarily high. As our goal is to design a good approximation algorithm, we propose a $O(1)$-energy $O(1)$-approximation algorithm, which uses constant energy augmentation. A schedule is called c-energy ρ-approximate if its objective function is at most ρ times far from the objective function of an optimal schedule and it exceeds the given energy budget by a factor of at most c (see e.g. [91]). Our result is based on an interval indexed LP-relaxation of the problem that incorporates a discretization of the possible speed values and transforms an optimal solution of this LP to a feasible solution for our problem, by list scheduling in the order of tasks' α-points (see e.g. [61, 98]). Further-

more, we introduce a trade-off between the approximation ratio and energy augmentation, as a function of α, where our result becomes a constant-factor approximation. In fact, our algorithm achieves a $\frac{(\alpha^{\beta-\frac{1}{\sqrt{\alpha}}})^2+3\alpha^{\beta-\frac{1}{\sqrt{\alpha}}}+1}{(\alpha^{\beta-\frac{1}{\sqrt{\alpha}}})^2(1-\alpha)}(1+\varepsilon)$-approximation, where $\varepsilon > 0$, $\alpha \in (0,1)$ and $\beta > 1$ is the exponent of speed in the power function; recall that $P(s) = s^\beta$. In the case where there are no precedence constraints between Map and Reduce tasks and all jobs have a common release date, the problem becomes equivalent with the speed scaling version of concurrent open-shop scheduling under an energy budget, and our algorithm achieves a $\frac{1}{\alpha^{\beta-\frac{1}{\sqrt{\alpha}}}(1-\alpha)}(1+\varepsilon)$-approximation.

In the same Chapter 3, under the context of classical scheduling we study the more general (and practically important) case of scheduling a set of MapReduce jobs on unrelated processors under the objective of minimizing their total weighted completion times. In [83], Moseley et al. considered MapReduce scheduling as a generalization of the two-stage flexible flow shop problem. They focused on the identical processors setting, proposing a 12-approximation algorithm. For unrelated processors, they dealt with the simple case in which each job has a single Map and a single Reduce task and proposed a 6-approximation algorithm. In our work, we consider the general case where each job can have any number of Map and Reduce tasks and propose a 54-approximation algorithm. Our main technical tool is the computation of a schedule of the Map (resp. Reduce) tasks to processors by combining a time indexed LP-relaxation of the problem with a well-known approximation algorithm for the makespan minimization problem on unrelated processors running on the time intervals specified by the optimal LP solution. Then, our result is derived by merging the two schedules produced for the Map and the Reduce tasks into a single schedule that respects task dependencies. Moreover, we integrate our model to capture data Shuffle, which forms a significant part of the processing cost in MapReduce applications. More specifically, we introduce a number of Shuffle tasks for each Map task that simulate the transmission of each key-value pair computed from a Map task, to every Reduce task. Each Shuffle task is associated with a transfer time, which is independent of the processor assignment, while its execution lies on some reasonable assumptions. For the Map-Shuffle-Reduce scheduling problem, we prove constant approximation algorithms in two different cases: i) a 54-approximation when the Shuffle tasks are scheduled on the same processors as the corresponding Reduce tasks and ii) a 81-approximation when the Shuffle tasks run on different processors of the Reduce tasks.

Temperature-aware scheduling. In Chapter 4, we study a temperature-aware scheduling problem where we are given a set of unit-length jobs, each one associated with a heat contribution and a common deadline, to be scheduled on single processor that operates under the thermal and cooling mechanism [41]. The goal is to maximize throughput, while not exceeding the given thermal threshold Θ. In their paper, Chrobak et al. [41] proved that the above problem is strongly $\mathcal{NP}$-hard. Moreover for the online version where each job has an arbitrary deadline and a release date they proposed a class of 2-competitive scheduling

policies, where each job is never scheduled if a job with a smaller deadline or heat contribution is available. In this thesis, we revisit this class of problems proposed in [41], and focus on the maximization of the throughput in the offline setting, under CoolestFirst scheduling: each time unit when the temperature is cool enough, the processor executes the job with the smallest heat contribution, otherwise remains idle. For the case of unit-length jobs with a common deadline we propose two lower bounds on the approximation factor of CoolestFirst, providing two different rounding schemes: a standard one, based on a partition of the produced schedule according to job's heat contributions and a more delicate one that uses a linear programming approach, yielding a lower bound on the approximation factor of at least 0.72.

Note

The results of this thesis are based on the following papers:

- Evripidis Bampis, Dimitrios Letsios, Ioannis Milis and Georgios Zois. *Speed Scalig for Maximum Lateness.* In Proceedings of the 18th Annual International Computing and Combinatorics Conference (COCOON), Springer LNCS 7846:187-200, 2012.
- Christoph Dürr, Ioannis Milis, Julien Robert and Georgios Zois. *Approximation the Throughput by Coolest First Scheduling.* In Proceedings of 10th Workshop on Approximation and Online Algorithms (WAOA), Springer LNCS 7434:25-36, 2012.
- Dimitris Fotakis, Ioannis Milis, Emmanouil Zampetakis and Georgios Zois. *Scheduling MapReduce jobs on Unrelated Processors.* In Workshop Proceedings of the EDBT/ICDT Joint Conference, CEUR-WS.org (ISSN 1613-0073), pp. 2-5, 2014. Final version in arxiv preprint, abs/1312.4203, 2014.
- Evripidis Bampis, Vincent Chau, Dimitrios Letsios, Giorgio Lucarreli, Ioannis Milis and Georgios Zois. *Energy-efficient Scheduling of MapReduce jobs.* In Proceedings of the 20th International European Conference on Parallel and Distributed Computing (EURO-PAR), Springer LNCS 8632:198-209, 2014.

Chapter 2

Speed Scaling to minimize maximum lateness

An instance of our problem consists of a set of n jobs $\mathcal{J} = \{1, 2, \ldots, n\}$, where every job j is associated to a release date r_j, a due date d_j, as well as to a work volume v_j. This set of jobs has to be executed non-preemptively on a single speed-scalable processor. For a given schedule the *lateness* of job j is defined as $L_j = C_j - d_j$, where C_j is the completion time of job j and the *maximum lateness* is defined as $L_{\max} = \max_{1 \leq j \leq n}\{L_j\}$. However, this setting is not amenable to obtaining near-optimal solutions, as an optimal $L_{\max}$ could be negative. An easy way to overcome this is to assume that all due dates are negative (by subtracting a sufficiently large constant from each one), which implies that the optimal $L_{\max}$ is always positive. Then, an equivalent model is that of scheduling with delivery times [58], where each job j needs a time $q_j \geq 0$ after its completion to be delivered and different jobs may be delivered simultaneously. By setting $q_j = -d_j$ we obtain an equivalent instance in the new model, since for any schedule $L_{\max} = \max_{1 \leq j \leq n}\{C_j - d_j\} = \max_{1 \leq j \leq n}\{C_j + q_j\}$. Due to this equivalence, in the sequel we are using the delivery times model and referring to the quantity $L_j = C_j + q_j$ as the lateness of job j. Jobs that attain the maximum lateness in a schedule are referred as *critical* jobs.

Adopting the speed scaling mechanism proposed by Yao et al. [111] (see Section 1.1) if a processor runs at speed s, at a given time, then its power consumption is $P(s) = s^\beta$, where $\beta > 1$ is a processor dependent constant, specifying the energy-inefficiency of the processor; speeding up by a factor c increases the power consumption by $c^{\beta-1}$ per unit of computation. The processor's energy consumption can be computed by integrating power over time. In this context, a schedule σ has to specify for every job the time interval during which it is executed as well as its speed over time. A key-property, which is actually a straightforward consequence of the convexity of speed-to-power function, is the following.

Lemma 2.1 *[65] Consider an instance I of the* Bud-Lateness *(resp.* Aggr-Lateness*) problem with a convex power function P. There is an optimal schedule where every job*

$j \in \mathcal{J}$ *is executed at constant speed* s_j.

According to Lemma 2.1, if a processor operates at a constant speed s_j during the execution of a job j, it will execute its whole amount of work volume v_j in v_j/s_j time units while consuming an amount of energy $E_j = v_j s_j^{\beta-1}$.

We consider the energy-aware problem of scheduling *non-preemptively* the set of jobs $\mathcal{J}$ on a single speed-scalable processor so as: (i) to minimize the maximum lateness, $L_{\max} = \max_{i \in \mathcal{J}}\{L_j\}$ under a given budget of energy E, so called *budget variant*, and (ii) to minimize a linear combination of maximum lateness and energy, $L_{\max} + \lambda E$, where $\lambda \geq 0$ is a given parameter, so called *aggregated variant*. The second objective specifies the relative importance of energy versus maximum lateness. We denote the former variant by Bud-Lateness and the latter by Aggr-Lateness.

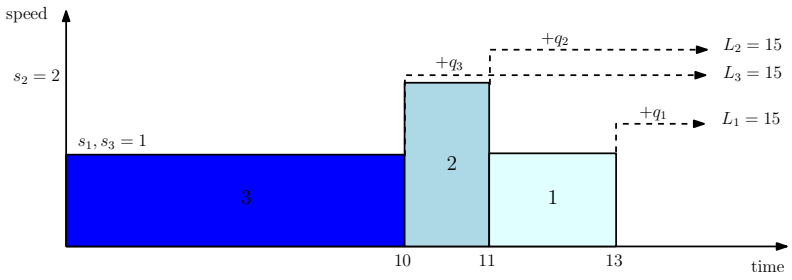


Figure 2.1: A feasible schedule of the Bud-Lateness problem for three jobs with zero release dates, work volumes 10, 2, 2, delivery times 5, 4, 2, $\beta = 2$ and $E = 20$. The total energy consumption is equal to 18, while the maximum lateness equals $L_{\max} = 15$, and it is attained by all jobs.

A feasible schedule of the Bud-Lateness problem for an instance of three jobs, with an energy budget equal to $E = 20$, is illustrated in Figure 2.1. Note that this is also feasible for the Aggr-Lateness problem.

2.1 Related work

Since the seminal paper by Yao et al. [111] speed scaling scheduling has been studied extensively (see e.g. [6] for a recent review). Here, we refer to the most significant results relevant to our work, focusing only on results for the single speed-scalable processor environment. The case of multiple speed-scalable processors will be examined in Section 3.1. Yao et al. [111] introduced the theoretical study of speed scaling by considering a scheduling problem, where the jobs are associated with release dates and deadlines, and the goal is to find a feasible *preemptive* schedule on a single speed-scalable processor with the minimum energy used while respecting the deadline feasibility QoS criterion. They studied both the offline and online versions. For the offline version, they proposed an elegant greedy algorithm, so called YDS, that runs in polynomial time $O(n^3)$. The optimality of YDS was

formally proved later (by applying the KKT conditions on a convex programming formulation) by Bansal et al. [25]. For the online version, they presented two constant-competitive algorithms, the Average Rate and the Optimal Available algorithm. More specifically, they proved that the competitive ratio of Average Rate is at most $2^{\beta-1}\beta^{\beta}$, which is almost tight, as it was later proved by Bansal et al. [27], since it cannot be smaller than $(2-\delta)^{\beta-1}\beta^{\beta}$, where δ tends to zero as β goes to infinity. Optimal Available was later analyzed by Bansal et al. [25], where they proved a tight competitive ratio of β^{β}. In [25], the authors also proposed a different online strategy with competitive ratio of $2(\frac{\beta}{\beta-1})^{\beta}e^{\beta}$. Recently, a general lower bound of $\frac{e^{\beta-1}}{\beta}$, for every deterministic online algorithm, has been proposed by Bansal et al. [29]. Finally, in the *non-preemptive* case the energy minimization problem was proved strongly $\mathcal{NP}$-hard [15] and to the best of our knowledge, the best result so far [22] attains an approximation ratio of $(1+\epsilon)^{\beta}\tilde{B}_{\beta}$, where $\tilde{B}_{\beta}$ is a generalization of the Bell number also valid for fractional values of β.

Most closely-related to our results are the following, concerning several QoS criteria under a budget or an aggregated variant on a single speed-scalable processor.

Pruhs et al. [90] studied the problem of minimizing the *total flow time* of jobs in a schedule, under a given energy budget, when jobs have arbitrary release dates. The flow time of a job is defined as the difference between its completion time and its release date. For the case where jobs have unit-work jobs, they proposed an optimal hill-climbing [1] polynomial time algorithm, of complexity $O(n^2 \log c)$, where c is the range of possible energy values whom each schedule is optimal, divided by the desired precision. For jobs with arbitrary work volumes, the authors proposed a $(1+\epsilon)^{\beta}$-energy $O(\frac{1}{\epsilon})$-approximate polynomial time algorithm, using $(1+\epsilon)^{\beta}$ energy augmentation, for $\epsilon \in (0,1)$. For the same problem, Bunde [35] proved a negative result, stating that there is no exact algorithm using exact real arithmetic, including the extraction of k-th roots, even when jobs have unit-work volumes. Albers and Fujiwara [7] were the first to consider an aggregated variant, with the objective of minimizing the total flow times of jobs plus energy. They studied both offline and online versions of the problem, in the case of unit-work jobs. In the offline case, they developed a polynomial time dynamic programming algorithm, with time complexity of $O(n^3 \log \rho)$, where ρ is the inverse of the desired precision. In the online case, they first proved that when jobs have arbitrary work volumes the competitive ratio of any deterministic *non-preemptive* online algorithm cannot be better than $\Omega(n^{1-1/\beta})$ and they proposed a $8.22(1+\Phi)^{\beta}(\frac{\beta}{1-\beta})^{\beta}$-competitive algorithm, where $\Phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. The latter result was improved by Bansal et al. [26], which proposed a 4-competitive algorithm. In [26] the authors also studied the *preemptive* version when jobs have arbitrary work volumes and presented a $(1+\epsilon)\max\left\{2, \frac{2(\beta-1)}{\beta-(\beta-1)^{\frac{\beta-2}{\beta-1}}}\right\}$, for $\epsilon \in (0,1)$, which was

1. The algorithm starts with an optimal schedule for a large energy budget and decreases energy while tracking the changes to the schedule.

improved to $\frac{2}{1-\left((\beta-1)/\beta^{\frac{\beta}{\beta-1}}\right)}$ by Lam et al. [74] and to 3-competitive by Bansal et al. [28]. Finally, Andrew et al. [10], based on the latter result proposed a 2-competitive algorithm. However, for the budget variant in the online setting, Bansal et al. [26] proved that there is no $O(1)$-competitive algorithm, even if all jobs have unit works.

Megow and Verschae [82] studied the problem of minimizing the total weighted completion time under a given energy budget and proposed a PTAS in the case where all jobs are released at time zero. It is interesting to note, that, for their result they applied the KKT conditions to a convex programming formulation of the problem, with respect to a given order of jobs, and proved that their problem is equivalent to the single (unit speed) processor scheduling problem with the objective of minimizing the $\sum_{j\in\mathcal{J}} w_j(C_j)^{\frac{\beta-1}{\beta}}$; actually, the same result has been also proved by Vásquez [104]. So, their PTAS is in fact applied for the latter problem. Megow and Verschae [81] also proposed a $(2+\epsilon)$-approximation algorithm, when jobs have arbitrary released dates and *preemption* is allowed. Carrasco et al. [36] studied the aggregated variant of minimizing the total completion time plus energy, when jobs are subject to precedence constraints. They proposed a $4(1+\epsilon)(1+\delta)$-approximation algorithm, for $\delta, \epsilon \in (0,1)$. They also proposed a similar $4^{\beta}(1+\epsilon)^{\beta-1}(1+\delta)^{\beta-1}$-approximation algorithm for the objective of minimizing the total weighted tardiness plus energy, where the tardiness T_j of a job j in a schedule is defined as the $T_j = \max\{0, C_j - d_j\}$.

Angel et al. [13] studied the problem of maximizing the throughput for a given energy budget and proposed an optimal polynomial time dynamic programming algorithm, of complexity $O(n^4 \log n \log P)$, where P is the sum of the jobs' work volumes, in the case where all jobs are released at time zero. Based on dynamic programming, they also gave an optimal polynomial time algorithm, of time complexity $O(n^6 \log n \log P)$, in the case of jobs with agreeable deadlines, i.e., for every pair of jobs i, j such that $r_i < r_j$ it holds that $d_i \leq d_j$. Moreover, they studied the weighted case, where the goal is to maximize the total weight of the jobs that finishing after their deadlines, and proved that the problem is $\mathcal{NP}$-hard in the ordinary sense, while also proposed an optimal pseudo-polynomial time algorithm, when jobs have agreeable deadlines. Angel et al. [12] also proved optimal pseudo-polynomial time algorithms, for the general problem when preemption is allowed, for both weighted and unit-weight cases.

Bunde [35] studied the problem of makespan minimization under an energy budget and proposed an optimal polynomial time algorithm, on the order of $O(n^2)$, for the *non-preemptive* case where each job has an arbitrary release date. Moreover, the author proposed a slight modification of the latter algorithm for finding all Pareto optimal schedules.

2.2 Contribution

Maximum lateness is a standard QoS criterion in scheduling theory, whose optimization has been extensively studied in the classical scheduling literature [66, 76, 88, 59]. Moreover,

it extends the deadline feasibility QoS criterion studied in the seminal paper by Yao et al. [111] in which the maximum lateness is fixed to zero. In this chapter, we propose to minimize maximum lateness in the context of energy management under the speed scaling setting.

In general, high processor speeds imply high performance with respect to the maximum lateness at the price of high energy consumption. As maximum lateness minimization and energy savings are conflicting objectives, we consider two variants of the problem: the budget variant (Bud-Lateness), where we aim in minimizing maximum lateness for a given budget of energy and the aggregated variant (Aggr-Lateness) where our objective is to minimize a linear combination of maximum lateness and energy. Note that the multiplication of either energy or lateness by a scalar, captures the relative value of these terms in the total cost. In both variants of the problem preemption is not allowed.

Our first result, in Section 2.3, is an optimal polynomial time algorithm for the Bud-Lateness problem when all jobs are released at time zero, running at time $O(n \log n)$. To this direction, we first note that in an optimal schedule jobs are executed in a non-increasing order of their delivery times. Moreover, due to the convexity of the speed-to-power function, we are able to formulate the problem as a convex program, which is actually already an optimal algorithm for the problem by applying the Elipsoid algorithm [84] which runs in polynomial time with arbitrary precision. As the Elipsoid algorithm is rather impractical, in order to derive a fast combinatorial algorithm, we apply the KKT conditions in our convex program and deduce a number of structural properties that, as we prove, are necessary and sufficient for a schedule to be optimal. Thus, it suffices to design an algorithm that creates such a schedule. Our algorithm is executed in two steps. In the first step it constructs groups of jobs satisfying the latter properties, apart from consuming an energy amount not greater than the energy budget E. In the second part, it manages the energy consumption with respect to E.

When the jobs have arbitrary release dates, in Section 2.4, we prove that the Bud-Lateness problem becomes strongly $\mathcal{NP}$-hard, by a reduction from 3-PARTITION. We also prove that, in the online case, there is no constant competitive deterministic algorithm, even when all jobs have unit works. This is actually a presumable result as, when we do not know the future jobs, it is difficult to decide how much energy to invest for the currently available ones. A similar result has been proved by Bansal et al. [26] for the total flow time QoS criterion. Next, in Section 2.5, we turn our attention to the Aggr-Lateness problem and in the case when all jobs are released at time zero, by using a similar analysis as for the Bud-Lateness problem (in Section 2.3) we derive an optimal polynomial time algorithm. In fact, the same result can be also derived by performing a binary search procedure over the interval of all possible energy levels, applying our optimal algorithm for the Bud-Lateness problem, for each candidate energy budget. For the general case of jobs having arbitrary release dates, we also prove strongly $\mathcal{NP}$-hardness, by a similar reduction

from 3-PARTITION, and then focus on the online case of our problem for which we propose a 2-competitive algorithm. Our algorithm processes jobs into phases and the jobs in each phase are scheduled according to our optimal offline algorithm, where jobs have a common release date. Finally, in Section 2.6, we conclude and propose directions for future work.

2.3 Budget variant with common release dates

In this section we present a polynomial algorithm for the BUD-LATENESS problem, when all jobs have a common release date r. For convenience, let r equal to zero. Our algorithm is based on a number of structural properties of an optimal schedule, deduced by formulating our problem as a convex program and applying the KKT conditions (see Appendix A for definition).

2.3.1 A convex programming formulation

A convex programming formulation of our problem stems from two basic properties of an optimal schedule. First, because of the convexity of the speed-to-power function, each job j runs at a constant speed s_j. Second, jobs are scheduled according to the EDD (Earliest Due Date First) rule, or equivalently in non-increasing order of their delivery times; this can be easily shown by a standard exchange argument. Hence, we propose the formulation (CP), where all jobs are considered to be released at time zero and numbered according to the EDD order.

$$
\begin{aligned}
&\textbf{(CP)} : \text{minimize } L \\
&\text{subject to :} \\
&C_j + q_j \leq L, && \forall j \in \mathcal{J} && (2.1)\\
&\frac{v_1}{s_1} \leq C_1, && && (2.2)\\
&C_{j-1} + \frac{v_j}{s_j} \leq C_j, && \text{for } j = 2, 3 \ldots, n && (2.3)\\
&\sum_{j=1}^{n} v_j s_j^{\beta-1} \leq E, && && (2.4)\\
&L, C_j, s_j \geq 0, && \forall j \in \mathcal{J} && (2.5)
\end{aligned}
$$

Our objective is to minimize the maximum lateness, L, among all feasible schedules. Constraints (2.1) ensure that the lateness of each job is at most L, constraints (2.2) and (2.3) enforce the jobs to be scheduled according to the EDD rule in non-overlapping time intervals, constraint (2.4) does not allow to exceed the given energy budget E and constraints (2.5) ensure that the maximum lateness, the completion times and the speeds of jobs are

non-negative. Constraint (2.4), for $\beta > 2$, and constraints (2.2) and (2.3) are convex, while constraints (2.1) and (2.5) and the objective function are linear. Thus, our mathematical program, (CP), is indeed convex.

Note that, (CP) already implies a polynomial algorithm for our problem, as convex programs can be solved to arbitrary precision by the Ellipsoid algorithm [84]. Since the Ellipsoid algorithm is rather impractical, we will exploit this convex program to derive a fast combinatorial algorithm.

2.3.2 Properties of an optimal schedule

In what follows we deduce a number of structural properties of an optimal schedule by applying the KKT conditions to the convex program (CP).

Lemma 2.2 *For the maximum lateness problem with an energy budget E, the following properties are necessary and sufficient for optimality of a feasible schedule.*
(i) Each job j runs at a constant speed s_j.
(ii) Jobs are scheduled according to the EDD rule.
(iii) There are no idle periods in the schedule.
(iv) The last job is critical, i.e., $L_n = L_{\max}$.
(v) Every non-critical job j has equal speed with the job $j+1$, i.e., $s_j = s_{j+1}$.
(vi) Jobs are executed in non-increasing speeds, i.e., $s_j \geq s_{j+1}$.
(vii) All the energy budget is consumed.

Proof. In order to apply the KKT conditions to (CP), we associate to each set of constraints from (2.1) up to (2.4), dual variables $\eta_j, \gamma_1, \gamma_j, \delta$, respectively. W.l.o.g. the variables L, C_j and s_j are positive and, by the complementary slackness conditions, the dual variables associated to the constraints (2.5) are equal to zero.

Stationarity conditions give that

$$\begin{aligned} \nabla L + \sum_{j=1}^{n} \eta_j \nabla (C_j + q_j - L) + \gamma_1 \nabla (\frac{v_1}{s_1} - C_1) \\ + \sum_{j=2}^{n} \gamma_j \nabla (C_{j-1} + \frac{v_j}{s_j} - C_j) + \delta \nabla (\sum_{j=1}^{n} v_j s_j^{a-1} - E) &= 0 \Rightarrow \\ (1 - \sum_{j=1}^{n} \eta_j) \nabla L + \sum_{j=1}^{n-1} (\eta_j - \gamma_j + \gamma_{j+1}) \nabla C_j \\ + (\eta_n - \gamma_n) \nabla C_n + \sum_{j=1}^{n} (-\gamma_j v_j s_j^{-2} + (a-1) \delta v_j s_j^{a-2}) \nabla s_j &= 0 \end{aligned}$$

Equivalently, we obtain the following equalities.

$$\sum_{j=1}^{n} \eta_j = 1 \tag{2.6}$$

$$\eta_j = \gamma_j - \gamma_{j+1} \quad 1 \leq j \leq n-1 \tag{2.7}$$

$$\eta_n = \gamma_n \tag{2.8}$$

$$(\beta - 1)\delta = \frac{\gamma_j}{s_j^{\beta}} \qquad 1 \leq j \leq n \tag{2.9}$$

The complementary slackness conditions give that

$$\eta_j(C_j + q_j - L) = 0 \quad 1 \leq j \leq n \tag{2.10}$$

$$\gamma_1(\frac{v_1}{s_1} - C_1) = 0 \tag{2.11}$$

$$\gamma_j(C_{j-1} + \frac{v_j}{s_j} - C_j) = 0 \quad 2 \leq j \leq n \tag{2.12}$$

$$\delta\left(\sum_{j=1}^{n} v_j s_j^{\beta-1} - E\right) = 0 \tag{2.13}$$

First, we show that the properties are necessary for optimality. That is, there is always an optimal schedule satisfying them.

(i)-(ii) They have been already discussed above.

(iii) First, note that $\delta \neq 0$. If $\delta = 0$ then by (2.9), we get that $\gamma_j = 0$ for each $1 \leq j \leq n$. This, combined with (2.7) and (2.8) yields that $\sum_{j=1}^{n} \eta_j = 0$, which is a contradiction because of (2.6). Since $\delta \neq 0$, we get by (2.9) that $\gamma_j \neq 0$ for each $1 \leq j \leq n$. Then, equations (2.11) and (2.12) give that there is no idle time in any optimal schedule since $C_1 = \frac{v_1}{s_1}$ and $C_j = C_{j-1} + \frac{v_j}{s_j}$, for $2 \leq j \leq n$, respectively.

(iv) Since $\delta \neq 0$, by (2.9), it follows that $\gamma_n \neq 0$ and finally, because of (2.8), $\eta_n \neq 0$. So, the last job to finish is always a critical job, by (2.10).

(v) Note that for every non-critical job j, it holds that $C_j + q_j < L$ and (2.10) implies that $\eta_j = 0$ for every such job. Hence, if a job j is non-critical $\eta_j = 0 \Rightarrow \gamma_j = \gamma_{j+1} \Rightarrow s_j = s_{j+1}$, by (2.7) and (2.9), respectively.

(vi) By the dual feasibility conditions and the equations (2.7) and (2.9) we get, respectively, that $\eta_j \geq 0 \Rightarrow \gamma_j \geq \gamma_{j+1} \Rightarrow s_j \geq s_{j+1}$. Thus, the jobs are executed with non-increasing speeds.

(vii) If the energy budget is not entirely consumed, then by (2.13), $\delta = 0$, which is a contradiction, since, as we have already proved, $\delta \neq 0$.

Next, we show that the properties are also sufficient for optimality. That is, any feasible schedule satisfying them must be optimal. In order to show this, it suffices to prove that, given any feasible schedule satisfying the properties, we can always give values to the dual variables such that the KKT conditions are satisfied.

Consider a feasible schedule and let s_j and C_j be the speed and the completion time of the job j, $1 \leq j \leq n$, respectively. Moreover, let L be the maximum lateness of the schedule. We give values to the dual variables as follows.

$$\delta = \frac{1}{(\beta-1)s_1^\beta}$$
$$\gamma_j = \frac{s_j^\beta}{s_1^\beta}, \qquad 1 \leq j \leq n$$
$$\eta_j = \frac{s_j^\beta - s_{j+1}^\beta}{s_1^\beta}, \quad 1 \leq j \leq n-1$$
$$\eta_n = \frac{s_n^\beta}{s_1^\beta}$$

We, now, observe that these values of the dual variables together with the values of the primal variables satisfy the KKT conditions.

Note that

$$\sum_{j=1}^{n} \eta_j = \sum_{j=1}^{n} \frac{s_j^\beta - s_{j+1}^\beta}{s_1^\beta} = \frac{s_1^\beta}{s_1^\beta} = 1$$
$$\eta_j = \frac{s_j^\beta - s_{j+1}^\beta}{s_1^\beta} = \frac{s_j^\beta}{s_1^\beta} - \frac{s_{j+1}^\beta}{s_1^\beta} = \gamma_j - \gamma_{j+1} \quad 1 \leq j \leq n-1$$
$$\eta_n = \frac{s_n^\beta}{s_1^\beta} = \gamma_n$$
$$(\beta-1)\delta = \frac{1}{s_1^\beta} = \frac{s_j^\beta}{s_1^\beta}\frac{1}{s_j^\beta} = \frac{\gamma_j}{s_j^\beta} \qquad 1 \leq j \leq n$$

So the stationarity conditions are satisfied.

Consider now a job j, $1 \leq j \leq n$. If i is critical, then $C_j + q_j = L$. Else, by property *(v)* we have that, for $1 \leq j \leq n-1$,

$$s_j = s_{j+1} \Leftrightarrow \frac{s_j^\beta}{s_1^\beta} = \frac{s_{j+1}^\beta}{s_1^\beta} \Leftrightarrow \eta_j = 0$$

Thus, equation (2.10) is satisfied. By property *(iii)*, we have that $C_1 = \frac{v_1}{s_1}$ and $C_j = C_{j-1} + \frac{v_j}{s_j}$, for $2 \leq j \leq n$. Therefore, equations (2.11) and (2.12) are also satisfied. Furthermore, by property *(vii)*, all the energy budget is consumed and the equation (2.13) holds. Hence, the complementary slackness conditions are satisfied.

Finally, in order to complete our proof, it remains to show that the values of all the dual variables are non-negative. The only case for which this is not straightforward, is for the values of variables η_j, for $1 \leq j \leq n-1$. But, it must be the case that $\eta_j \geq 0$ for all

$1 \leq j \leq n-1$, because of the property *(vi)* and the theorem follows. ∎

We refer to any schedule satisfying the properties of Lemma 2.2 as a *regular* schedule. By (i, j) we denote a sequence of consecutive jobs $i, i+1, \ldots, j$. Any regular schedule can be partitioned into *groups* of jobs, of the form (i, j), where the jobs $i-1$ and j are critical and the jobs $i, i+1, \ldots, j-1$ are not. By Lemma 2.2*(v)*, all jobs of such a group are executed at the same speed. We denote this common speed by s_j and the total amount of work volume of jobs in (i, j) by $v(i, j) = \sum_{k=i}^{j} v_k$. Then, the next proposition follows from Lemma 2.2.

Proposition 2.1 *Let i, j, be two consecutive critical jobs of a regular schedule. The speed of each job in the group $(i+1, j)$ equals to $s_j = \frac{v(i+1,j)}{q_i - q_j}$.*

Proof. Since i and j are both critical, they attain equal maximum latencies. Moreover, in any regular schedule, by Lemma 2.2*(iv)*, there is no idle period between jobs $i, i+1, \ldots, j$. Furthermore, all jobs $i+1, i+2, \ldots, j-1$ are non-critical and, by Lemma 2.2*(vi)*, they are all executed with speed equal to that of job j. Hence, we get, respectively, that

$$L_i = L_j \Rightarrow C_i + q_i = C_j + q_j \Rightarrow \sum_{k=1}^{i} \frac{v_k}{s_k} + q_i = \sum_{k=1}^{j} \frac{v_k}{s_k} + q_j \Rightarrow s_j = \frac{v(i, j)}{q_{i-1} - q_j}.$$

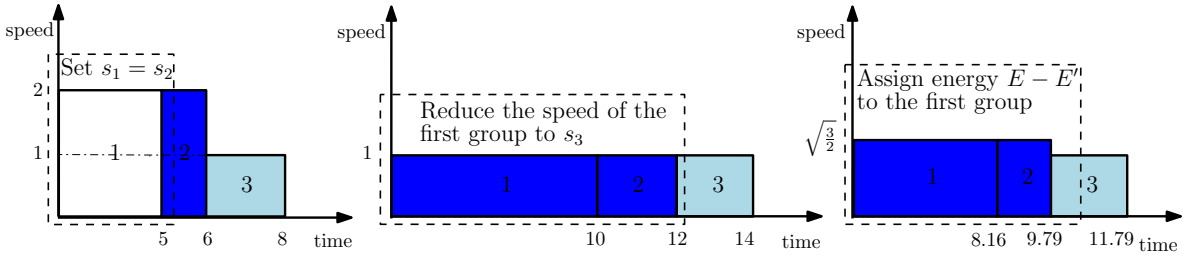
and the lemma follows. ∎

2.3.3 An optimal combinatorial algorithm

So far, by proving that the properties of a regular schedule are necessary and sufficient for optimality, we have derived a clear image of the structure of an optimal schedule for the Bud-Lateness problem, when all jobs are released at time zero. Next, we propose Algorithm BUD which constructs such a schedule in polynomial time. Note that a regular schedule is fully specified by the speeds of the jobs. The rough idea of our algorithm is the following: First, it constructs a preliminary schedule by finding groups of jobs running in non-increasing speeds without taking care of the energy consumption. Second, the algorithm manages the energy consumption w.r.t. the energy budget E and determines the final speeds of all jobs. Let E' be the energy consumption of the current schedule at any point of the execution of the algorithm.

Algorithm BUD starts from job n which is always a critical job and considers all jobs but the first, in reverse order (see lines 2-7). When a job j, $2 \leq j \leq n$, is considered for the first time, its speed s_j is set according to Proposition 2.1, assuming that jobs $j-1$ and j are critical. If $s_j \geq s_i$, for $j+1 \leq i \leq n$, then s_j is called *eligible* speed and it is assigned to job j. If this speed is not eligible, j is a non-critical job and it is merged with the $(j+1)$'s group. More specifically, if c is the last job of this group, then the speeds of jobs $j, j+1, \ldots, c$ are calculated by applying Proposition 2.1, assuming that $j-1$ and

c are critical while $j, j+1, \ldots, c-1$ are not. Next, the algorithm examines whether the new value of s_j is eligible. If this is the case, then it considers the job $j-1$. Otherwise, a further merging of the j's group with the $(c+1)$'s group, is performed, as before. That is, if c' is the last job of the $(c+1)$'s group, all jobs $j, j+1, \ldots, c'$ are assigned the same speed assuming that jobs $j-1$ and c' are critical, while $j, j+1, \ldots, c'-1$ are not. This speed, according to the Proposition 2.1, is equal to $s_{c'} = \frac{v(j,c')}{q_{j-1}-q_{c'}}$. Note that the job c is no longer critical in this case. This merging procedure is repeated until job i is assigned an eligible speed. In a degenerate case, jobs $j, j+1, \ldots, n$ are merged into one group. When the algorithm has assigned an eligible speed to all jobs $2, 3, \ldots, n$, in line 8, it sets $s_1 = s_2$ and its first part completes. An example of the first part of our algorithm is given in Figure 2.2(i).



(i) $L_{max} = 10,\ E' > E,\ E' = 50$ **(ii)** $L_{max} = 16,\ E' < E,\ E' = 14$ **(iii)** $L_{max} = 13.79,\ E' = 20$

Figure 2.2: The execution of Algorithm BUD for an instance of 3 jobs without release dates, work volumes 10, 2, 2, delivery times 5, 4, 2, $\beta = 3$ and $E = 20$.

Next, Algorithm BUD takes into account the available budget of energy E (see lines 9-21). If $E - E' \geq 0$, the current schedule's energy consumption does not exceed the budget of energy, and the surplus $E - E'$ is assigned to the first job. Otherwise, the current schedule is regular, except that it consumes an amount of energy greater than E. Then, the algorithm reduces the consumed energy until it becomes equal to E. In fact, it decreases the speed of the first group, by merging subsequent groups with it, if necessary. This merging procedure is different from the one of the first part of the algorithm and it is as follows: let j be the critical job of maximal index with $s_j = s_1$ in the current schedule. Observe that $s_j > s_{j+1}$. The algorithm sets the speed of jobs $1, 2, \ldots, i$ equal to s_{j+1}. This causes a reduction to E' and there are two cases to distinguish: either $E' \leq E$ or $E' > E$. In the first case, the algorithm adds an amount of energy $E - E'$ to jobs $1, 2, \ldots, j$ by increasing their speeds uniformly, i.e. so that they are all executed with the same speed. In the second case, at least one further merging step has to be performed. When the algorithm terminates, it is obvious that $E' = E$. For an example of the second part of our algorithm see Figures 2.2(ii) and 2.2(iii).

Theorem 2.1 *Algorithm BUD is optimal for the* Bud-Lateness *problem, when all jobs*

Algorithm BUD: an algorithm for the BUD-LATENESS problem, when jobs have common release dates.

```
1: Sort the jobs according to the EDD order.
2: for j = n to 2 do
3:     Set s_j assuming that j and j − 1 are critical.
4:     while s_j is not eligible do
5:         Merge the j's group with the next group.
6:     end while
7: end for
8: Set s_1 = s_2
9: Let E' be the current energy consumption.
10: if E > E' then
11:     Assign energy E − E' to job 1.
12: else
13:     while E < E' do
14:         Set the speed of the first group equal to the speed of the following group.
15:         Update E'.
16:         if E < E' then
17:             Merge the first group with the next one.
18:         end if
19:     end while
20:     Assign E − E' energy uniformly to the first group.
21: end if
```

are released at time zero.

Proof. We shall prove that the algorithm satisfies the properties of Lemma 2.2, i.e., it produces a regular schedule. For convenience, we distinguish two parts in the algorithm: *Part I*, corresponding to lines 1-8 and *Part II*, corresponding to lines 9-21, respectively.

Property (i)-(ii): The algorithm gives a single constant speed to each job and keeps their initial EDD order.

Property (iii): In Part I, the speeds of jobs are assigned according to Proposition 2.1. Specifically, the algorithm fixes two consecutive critical jobs i and j, $i < j$, with, potentially, some non-critical jobs between them. Then the speed of the non-critical jobs and the one of the critical job j is defined such that there is no idle period between the jobs. In Part II, no idle period is added between any jobs.

Property (iv) - (v): When the speed of job n is initialized, this is done by assuming that it is critical. Next, consider the current schedule just after the completion of Part I. This schedule can be partitioned into sequences of jobs, $a+1, a+2, \ldots, b$, with $a \geq 1$, such that the jobs of each sequence are executed with the same speed which has been assigned by applying Proposition 2.1, assuming that the jobs a and b are critical. In fact, jobs a and b attain equal lateness. In order for such a sequence to be a group, we should also prove that all but the last jobs are non-critical while the last job is critical.

Let $a+1, a+2, \ldots, b$ be a sequence of jobs. We claim that $L_j < L_b$, for $a+1 \leq j \leq b-1$.

Assume, by contradiction, that there exists a job i, where $a+1 \leq i \leq b-1$, such that $L_i \geq L_b$, or equivalently, $q_i - q_b \geq \sum_{j=i+1}^{b} \frac{v_j}{s_b}$. Since the last job of a sequence attains equal lateness with the last job of the sequence that follows, we have that $L_a = L_b$. This yields that $q_a - q_b = \sum_{j=a+1}^{b} \frac{v_j}{s_b}$. Therefore, $q_a - q_i \leq \sum_{j=a+1}^{i} \frac{v_j}{s_b}$.

Obviously, for any job j, $a+1 \leq j \leq b-1$, we must have a speed $s_j > \frac{v_j}{q_{j-1}-q_j}$, since otherwise, it wouldn't have been merged with another group. That is, $q_{j-1} - q_j > \frac{v_j}{s_j}$. If we sum the last inequalities for $a+1 \leq j \leq i$, we get that $q_a - q_i > \sum_{j=a+1}^{i} \frac{v_j}{s_b}$, a contradiction.

At this point, we have showed that when Part I completes, if a job j, $2 \leq j \leq n$, is critical, then it must be the right extremity of a sequence. Moreover, among all jobs $2, 3, \ldots, n$, the last jobs of all sequences, including job n, attain equal lateness and the remaining jobs attain smaller lateness. In addition, job 1 attains equal lateness with the last job of the sequence that follows. Recall that, at this point, we set $s_1 = s_2$. Job 1 would have equal lateness with the last job of the sequence that follows for any $s_1 > 0$ since the speed of the second group is set by applying Proposition 2.1, assuming that 1 is critical. So, at the end of Part I, job 1, job n and every last job of a sequence are critical. Therefore, after Part I finishes, Properties *(iv)* and *(v)* hold.

In Part II, if no merging step is performed, then the processing time of job 1 is decreased by some $t \geq 0$ and its lateness decreases by t, while the processing times and speeds of the other jobs are not modified. So, the lateness of every other job also decreases by t. Hence, the Properties *(iv)* and *(v)* hold.

If at least one merging step is performed, then the speed of the jobs in the first group decreases and their processing time increases. Then, in the first group, every non-critical job j has equal speed with the job $j+1$ that follows, while the speeds of the jobs in other groups remain unchanged. Now, let t_j be the total increase in the processing time of job j, $1 \leq j \leq n$. Note that this quantity is positive only for jobs belonging to the first group of the current schedule. Then, the lateness of any job j, $1 \leq j \leq n$, increases by $\sum_{i=1}^{j} t_i$; if c_1 is the critical job of the first group, it remains critical after the merging step since its lateness and the lateness of every other job that follows, increase by the same quantity, equal to $\sum_{i=1}^{c_1} t_i$. Note, that if a further merging step is performed, we consider the first two groups as one group. Moreover, the lateness of any job increases by no more than the increase of the lateness of job n, and thus, in the final schedule, job n remains critical and Property *(iv)* holds. Furthermore, each non-critical job has equal speed with the job that follows and Property *(v)* holds as well.

Property (vi): At the end of Part I, the speeds of jobs are non-increasing since otherwise, a merging step would be performed. Moreover, during Part II, no speed of a job becomes less than the speed of a subsequent job.

Property (vii): Recall that E' is the total energy consumed when Part I completes. If E' is less than the energy budget, then the energy of the first job increases until the schedule consumes exactly E units of energy, while if E' is greater than the energy budget E, then

the energy consumption of the schedule is gradually decreased until it becomes equal to E.

Let us now consider the complexity of the algorithm. Initially, jobs are sorted according to the EDD rule in $O(n \log n)$ time. The first part of the algorithm may take $O(n^2)$ time since each merging step takes $O(n)$ time and there can be $O(n)$ merging steps. Also, the algorithm's second part takes $O(n^2)$ time since the speed of each job may change at most $O(n)$ times. Therefore, the overall complexity of the algorithm is $O(n^2)$. Note that, using a more careful analysis, based on the use of a stack data structure, it can be shown that the algorithm may be implemented in $O(n \log n)$ time. ∎

2.4 Budget variant with arbitrary release dates

We now consider the general Bud-Lateness problem, where the jobs have arbitrary release dates and we show that it becomes strongly $\mathcal{NP}$-hard. Moreover, we show that there is no $O(1)$-competitive algorithm for its online version, even when all jobs have unit-work volumes.

2.4.1 NP-hardness

We reduce 3-PARTITION to the Bud-Lateness problem. 3-PARTITION problem is a well known $\mathcal{NP}$-hard [51] problem where, we are given a positive integer B and a set of $3n$ positive integers $A = \{a_1, a_2, \ldots, a_{3n}\}$, where $B/4 < a_j < B/2$ and $\sum_{a_j \in A} a_j = nB$, and we ask if there exists a partition of A into n disjoint sets $A_1, A_2 \ldots, A_n$ such that, for each $1 \leq k \leq n$, $\sum_{a_j \in A_k} a_j = B$.

Our reduction is inspired by the $\mathcal{NP}$-hardness proof for the classical maximum lateness minimization on a single processor problem [51], where we are given a set of jobs with each job j having a release date r_j, a due date d_j and a processing time p_j and we seek a schedule with maximum lateness at most Z, for some integer value Z. This problem can be viewed as a variant of our problem where the speed of each job is part of the instance. Specifically, we consider that each job j has an amount of work volume $v_j = p_j$ and it is executed at a constant speed $s_j = 1$. Based on this idea, we extend the existing $\mathcal{NP}$-hardness reduction by fixing an energy budget, so that all jobs have to be executed at the same speed $s_j = 1$ in order to get a feasible schedule.

Theorem 2.2 Bud-Lateness *problem is strongly $\mathcal{NP}$-hard.*

Proof. We construct an instance of the Bud-Lateness problem from an instance of 3-PARTITION as follows. The instance is depicted in Table 2.1.

- For each integer a_j, $1 \leq j \leq 3n$, we create a job j with $v_j = a_j$, $r_j = 0$ and $q_j = 0$.
- We introduce $n - 1$ gadget jobs, where the gadget job j, $3n + 1 \leq j \leq 4n - 1$, has $v_j = B, r_j = (2i - 6n - 1)B$ and $q_j = (8n - 2i - 1)B$.
- We set $E = (2n - 1)B$.

We shall prove that there is a feasible schedule σ with $L_{\max}$ at most Z and total energy consumption $E = (2n-1)B$ if and only if there exists a 3-PARTITION of A.

j	v_j	r_j	q_j
1	a_1	0	0
2	a_2	0	0
$\dots$	$\dots$	$\dots$	$\dots$
$3n$	a_{3n}	0	0
$3n+1$	B	B	$(2n-3)B$
$3n+2$	B	$3B$	$(2n-5)B$
$3n+3$	B	$5B$	$(2n-7)B$
$\dots$	$\dots$	$\dots$	$\dots$
$4n-2$	B	$(2n-5)B$	$3B$
$4n-1$	B	$(2n-3)B$	B

Table 2.1: An instance of BUD-LATENESS problem reduced from an instance of 3-Partition.

($\Leftarrow$) For the first direction, assume that $A_1, A_2 \dots, A_n$ is a partition of A, where $\sum_{a_j \in A_k} a_j = B$ for $1 \leq k \leq n$. Then, consider the schedule σ where: (i) each job j corresponding to an integer $a_j \in A_k$, $1 \leq k \leq n$, is scheduled during the time interval $[2(k-1)B, (2k-1)B]$, (ii) each gadget job j, $3n+1 \leq j \leq 4n-1$ is scheduled during the time interval $[(2j-6n-1)B, (2j-6n)B]$, and (iii) all jobs are executed at constant speed $s_j = 1$. The schedule σ (see Figure 2.3) is feasible and attains maximum lateness equal to $L_{\max} = (2n-1)B$. The total energy consumed is $E = \sum_{j=1}^{4n-1} v_j s_j^{\beta-1} = \sum_{j=1}^{4n-1} v_j = (2n-1)B$. Thus, by defining $Z = (2n-1)B$, the above schedule corresponding to the instance of 3-PARTITION, has $L_{\max} \leq Z$ and $E = (2n-1)B$.

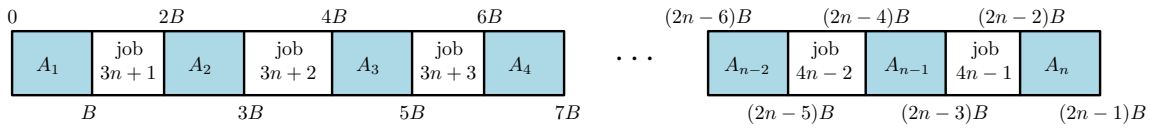


Figure 2.3: A feasible schedule σ for the BUD-LATENESS problem that attains maximum lateness equal to $L_{\max} = (2n-1)B$.

($\Rightarrow$) For the opposite direction, assume that σ is a feasible schedule with $L_{\max} \leq Z$, for $Z = (2n-1)B$, and total energy consumption $E = (2n-1)B$. In σ, each job j, $1 \leq j \leq 3n$, must have completion time $C_j \leq (2n-1)B$ and each gadget job j, $3n+1 \leq j \leq 4n-1$, must have completion time $C_j \leq (2j-6n)B$, since $L_j \leq (2n-1)B$ for every job j. For notational convenience, let $W = (2n-1)B$ be the sum of work volumes of all jobs. Let also p_j be the execution time of job j, $1 \leq j \leq 4n-1$.

It holds also that the completion time of (the last job of) schedule σ is $C_{\max} = (2n-1)B$. To see this, assume for the sake of contradiction that $C_{\max} < (2n-1)B$. Then, by the

convexity of speed-to-power function, it follows that the total energy consumption in σ will be

$$E(\sigma) = \sum_{j=1}^{4n-1} v_j s_j^{\beta-1} = \sum_{j=1}^{4n-1} v_j \left(\frac{v_j}{p_j}\right)^{\beta-1} \geq W \left(\frac{W}{C_{\max}}\right)^{\beta-1} > (2n-1)B$$

which is not possible because the energy budget is exceeded. With a similar argument, it can be shown that there will be no idle time during the interval $[0, (2n-1)B]$ in σ. Moreover, due to the convexity of the speed-to-power function, among the schedules with makespan $C_{\max} = (2n-1)B$ which have no idle period during $[0, (2n-1)B]$, only the ones in which all the jobs are executed with speed equal to $s_j = 1$ have energy consumption not greater than $E = (2n-1)$. Clearly, σ must be one of these schedules. Hence, every gadget job j, $3n+1 \leq j \leq 4n-1$, is executed within the whole time interval $[(2j-6n-1)B, (2j-6n)B]$ in σ.

So far we have shown that every gadget job j, $3n+1 \leq j \leq 4n-1$, spans in σ the time interval $[(2j-6n-1)B, (2j-6n)B]$, while the other jobs j, $1 \leq j \leq 3n$, span the time intervals $[2(k-1)B, (2k-1)B]$, $1 \leq k \leq n$. Therefore, during any interval $[2(k-1)B, (2k-1)B]$, $1 \leq k \leq n$, there will be executed a set of jobs with total amount of work volume B. This execution defines a 3-PARTITION for A. ∎

2.4.2 The on-line case

Let us now turn our attention to the online version of the Bud-Lateness problem. Bansal et al. [26] gave an adversarial strategy for proving that there is no $O(1)$-competitive algorithm for the problem of minimizing the total flow of a set of unit-work jobs on a single speed-scalable processor. This adversarial strategy consists of batches of jobs, $B_1, B_2, \ldots, B_k$, with all the jobs in batch B_i released after the online algorithm has finished all the jobs in B_{i-1}. Following a similar strategy it can be proved that the makespan minimization problem, for a given budget of energy, i.e. the Bud-Lateness problem, does not admit an $O(1)$-competitive algorithm. Note that the makespan minimization is a special case of our lateness problem (with $q_j = 0,\ 1 \leq j \leq n$).

Theorem 2.3 *There is no $O(1)$-competitive algorithm for the online version of the* Bud-Lateness *problem, even when jobs have unit-work volumes.*

Proof. In order to prove the theorem, we assume the existence of a ρ-competitive algorithm $\mathcal{A}$, where $\rho > 1$ is a constant. Then, we reach a contradiction by showing that there is an instance of the problem that cannot be feasibly solved by $\mathcal{A}$.

We consider a set of jobs consisting of batches $B_1, B_2, \ldots, B_\ell$, where the batch B_i, $1 \leq i \leq \ell$, contains $n_i = 2^{i-1}$ unit-work jobs which all arrive at the same time; the jobs of the batch B_1 are released at the time $r_1 = 0$ while the jobs of the batch B_i, $1 \leq i \leq \ell$, are released at time r_i. We assume that r_i is large enough so that the algorithm $\mathcal{A}$ has completed the jobs in the batches $B_1, \ldots, B_{i-1}$ by r_i.

We denote by $C^*_{\max,k}$, $1 \leq k \leq \ell$, the value of the makespan that the optimal offline algorithm achieves for the instance that consists exactly of the jobs in the batches $B_1, B_2, \ldots,$ B_k. The term $C^*_{\max,k}$ is upper bounded by the makespan of the schedule in which all the jobs in $B_1, B_2, \ldots, B_k$ are assigned equal speeds such that their energy consumption is equal to the energy budget E and they are executed continuously starting at time r_k. Therefore,

$$C^*_{\max,k} \leq r_k + \left(\frac{\left(\sum_{i=1}^{k} n_i\right)^{\beta}}{E} \right)^{\frac{1}{\beta-1}}. \tag{2.19}$$

As $\mathcal{A}$ is a ρ-competitive algorithm, it must complete all jobs of the batches $B_1, B_2,$ $\ldots, B_k$ not later than $\rho \cdot C^*_{\max,k}$ independently of the number of batches that our original instance contains. Otherwise, it wouldn't be ρ-competitive for the instance of the problem that consists only of the batches $B_1, B_2, \ldots, B_k$. Let $C_{\max,k}$ be the completion time of the jobs in batches $B_1, B_2, \ldots, B_k$ in $\mathcal{A}$'s schedule. Then, it must be the case that

$$C_{\max,k} \leq \rho \cdot C^*_{\max,k} \tag{2.20}$$

Let E_k be the energy consumption of the jobs in batch B_k in $\mathcal{A}$'s schedule. Due to the convexity of the speed-to-power function, we have that

$$E_k \geq \frac{n_k^{\beta}}{(C_{\max,k} - r_k)^{\beta-1}} \tag{2.21}$$

By combining inequalities (2.19), (2.20), (2.21) and the fact that $r_k \leq C^*_{\max,k}$, we obtain that

$$E_k \geq \frac{n_k^{\beta}}{\left(\sum_{i=1}^{k} n_i\right)^{\beta}} \frac{E}{(2\rho - 1)^{\beta-1}}$$

Since $n_i = 2^{i-1}$ for $1 \leq i \leq k$, we conclude that

$$E_k \geq \frac{E}{2^{\beta}(2\rho - 1)^{\beta-1}}$$

Thus, if the number of batches ℓ is large enough, i.e. $\ell \to \infty$, the algorithm will run out of energy after having completed $\lceil 2^{\beta}(2\rho - 1)^{\beta-1} \rceil$ batches, so it won't be able to finish the batches that follow. ∎

2.5 Aggregated variant

In this section we turn our attention to the Aggr-Lateness problem, the aggregated variant of the maximum lateness problem, where our objective is to minimize $L_{\max} + \lambda E$, for some $\lambda > 0$. For this variant, in the online case, we are able to overcome the impossibility

of obtaining constant-factor competitive algorithms (Theorem 2.3). Initially, we consider instances in which the jobs have a common release date and we describe how to obtain an optimal offline algorithm for the aggregated variant by slightly modifying our algorithm and its analysis for the budget variant in Section 2.3. For instances with arbitrary release dates, we prove that the problem is strongly $\mathcal{NP}$-hard, by using a similar reduction as for the budget variant. Last, we turn our attention to the online case of the aggregated problem in which the jobs arrive over time and we propose a 2-competitive algorithm which schedules the jobs into batches, by repeatedly applying our optimal offline algorithm for jobs with a common release date.

Common release dates. When all jobs are released at the same time, we are able to derive a polynomial algorithm, by using Algorithm BUD in the following way: suppose that we are given the energy consumption E^* of an optimal schedule minimizing $L_{\max} + \lambda E$. Then, in order to construct such an optimal schedule, it suffices to apply the optimal algorithm for the budget variant with an energy budget equal to E^*. This means that the optimal schedule for the aggregated variant is a regular schedule, satisfying the properties of Lemma 2.2 (with budget E^*). However, in order to construct the optimal schedule minimizing $L_{\max} + \lambda E$, we need to compute E^*. One approach, which has been already suggested in the literature for the total flow time criterion (see [7, 26]), would be to perform a binary search procedure in the interval of all possible energy levels. Here, we describe an alternative approach which resembles to the one we followed for the budget variant.

We first formulate the aggregated variant as a convex program similar to (CP) for the budget variant. Now, we do not introduce a constraint on the total energy consumption, since it is added in the objective function. By applying the KKT conditions, we obtain almost the same structure (properties) of an optimal solution with one single difference: the energy consumption is not specified by a given budget of energy, but it results from the fact that the speed of the first job should always be equal to a fixed value. Specifically, the Property *(vii)* of Lemma 2.2 is replaced by the fact that *"the job executed first runs at speed* $s_1 = \left(\frac{1}{(\beta-1)\lambda}\right)^{\beta}$*"*. Therefore, in order to obtain an optimal schedule for the aggregated variant, it suffices to do the following.
Run lines 1-7 of Algorithm BUD. Let σ *be the schedule produced. Find the highest-index critical job,* j*,* $j \neq 1$*, in* σ*, such that its corresponding sequence,* (k, j)*, has speed* $s_j < s_1$*. Modify* σ *such that all jobs* $1, 2, \ldots, k-1$ *are executed at speed* s_1*.*

Arbitrary release dates. When jobs have arbitrary release dates, then the problem becomes strongly $\mathcal{NP}$-hard. Similarly with the budget variant, our reduction is from the 3-PARTITION problem and uses the following lower bound on the objective of any optimal schedule.

Lemma 2.3 *For an optimal schedule of the* Aggr-Lateness *problem, it holds that*

$$OPT \geq \min_j\{r_j\} + \left(\frac{1}{s_1} + s_1^{\beta-1}\right)\sum_{j=1}^{n} v_j$$

where $s_1 = \left(\frac{1}{\lambda(\beta-1)}\right)^{\frac{1}{\beta}}$.

Proof. We denote the original problem by Π. Let Π' be the relaxed problem of minimizing the makespan plus λ times energy with release dates and Π'' the relaxed problem of minimizing the makespan plus λ times energy with a common release date equal to $\min_j\{r_j\}$. If $\mathrm{OPT}_\Pi, \mathrm{OPT}_{\Pi'}, \mathrm{OPT}_{\Pi''}$ is the cost of the optimal schedule for the problems Π, Π', Π'' respectively, it is easy to see that

$$\mathrm{OPT}_\Pi \geq \mathrm{OPT}_{\Pi'} \geq \mathrm{OPT}_{\Pi''}$$

Moreover, for an instance of the Π'' problem, note that in an optimal schedule all jobs are executed at a constant speed. Since the objective is to minimize the function

$$H(s) = \min_j\{r_j\} + \sum_{j=1}^{n}\frac{v_j}{s} + \lambda\sum_{j=1}^{n} v_j s^{\beta-1},$$

if we set the first derivative, with respect to s, equal to zero, we yield that the global minimum is achieved for $s^\beta = s_1^\beta = \frac{1}{\lambda(\beta-1)}$, and the lemma follows directly. ∎

Theorem 2.4 *The* Aggr-Lateness *problem is strongly $\mathcal{NP}$-hard.*

Proof. Given an instance of 3-PARTITION we will construct an instance of the Aggr-Lateness problem.

- For each integer a_j we create a job j with work volume $v_j = a_j(\frac{1}{\lambda(\beta-1)})^{\frac{1}{\beta}}$, $r_j = 0$ and $q_j = 0$, $1 \leq j \leq 3n$.
- We introduce $n-1$ gadget jobs, $3n+1, 3n+2, \ldots, 4n-1$, where the gadget job j has $v_j = (\frac{1}{\lambda(\beta-1)})^{\frac{1}{\beta}}B, r_j = (2j-6n-1)B$ and $q_j = (8n-2j-1)B$.

We claim that there is a feasible schedule σ with $L_{\max} + \lambda E$ at most Z if and only if there exists a 3-PARTITION of the set of integers A.

($\Leftarrow$) To the first direction, assume that $A_1, A_2 \ldots, A_n$ is a partition of A, where $\sum_{a_j \in A_k} a_j = B$, for $1 \leq k \leq n$. Then, consider the schedule σ according to which (i) each job j, corresponding to an integer $a_j \in A_k$, $1 \leq k \leq n$, is scheduled during the time interval $[2(k-1)B, (2k-1)B)$, (ii) each gadget job j, $3n+1 \leq j \leq 4n-1$ is scheduled during the time interval $[(2j-6n-1)B, (2j-6n)B)$, and (iii) all jobs are executed at constant speed $s_1 = (\frac{1}{\lambda(\beta-1)})^{\frac{1}{\beta}}$. The schedule σ is feasible and its value of objective function is $L_{\max} + \lambda E = (2n-1)B[1 + \lambda\frac{1}{\lambda(\beta-1)}] = (2n-1)B\frac{\beta}{\beta-1}$. By defining $Z = (2n-1)B\frac{\beta}{\beta-1}$, the above schedule corresponding to an instance of 3-PARTITION has objective value at most Z.

Algorithm ALE: an online algorithm for the AGGR-LATENESS problem.

Let J_0 be the set of jobs that arrive at time $t_0 = 0$. In phase 0, jobs in J_0 are scheduled according to $\sigma^*(J_0, 0)$. Let t_1 be the time at which the last job of J_0 is finished, i.e., the end of phase 0, and J_1 be the set of jobs released during $(t_0, t_1]$. In phase 1, jobs in J_1 are scheduled as in $\sigma^*(J_1, t_1)$ and so on. In general, if t_i is the end of phase $i-1$, we denote J_i to be the set of jobs released during $(t_{i-1}, t_i]$. Jobs in J_i are scheduled by computing $\sigma^*(J_i, t_i)$.

($\Rightarrow$) To the opposite direction, assume that σ is a feasible schedule with weighted maximum lateness plus energy equal to $L_{\max} + \lambda E \leq Z$, for $Z = (2n-1)B\frac{\beta}{\beta-1}$. By Lemma 2.3 the speed $s_1 = (\frac{1}{\lambda(\beta-1)})^{\frac{1}{\beta}}$ defines a unique global minimum for our objective. Moreover, by running each job in σ at a speed s_1, we attain maximum lateness plus energy equal to the assumed one. Thus, σ must run each job i at speed $s_i = s_1 = (\frac{1}{\lambda(\beta-1)})^{\frac{1}{\beta}}$. It is easy to see (as in the proof of Theorem 2.2) that there are no idle periods in σ and that (i) every gadget job j, $3n+1 \leq j \leq 4n-1$, will be executed during the whole time interval $[(2j-6n-1)B, (2j-6n)B)$ and (ii) during any interval $[2(k-1)B, (2k-1)B)$, $1 \leq k \leq n$, there will be executed a set of jobs with total amount of work volume $(\frac{1}{\lambda(\beta-1)})^{\frac{1}{\beta}}B$. This execution defines a 3-PARTITION for A. ∎

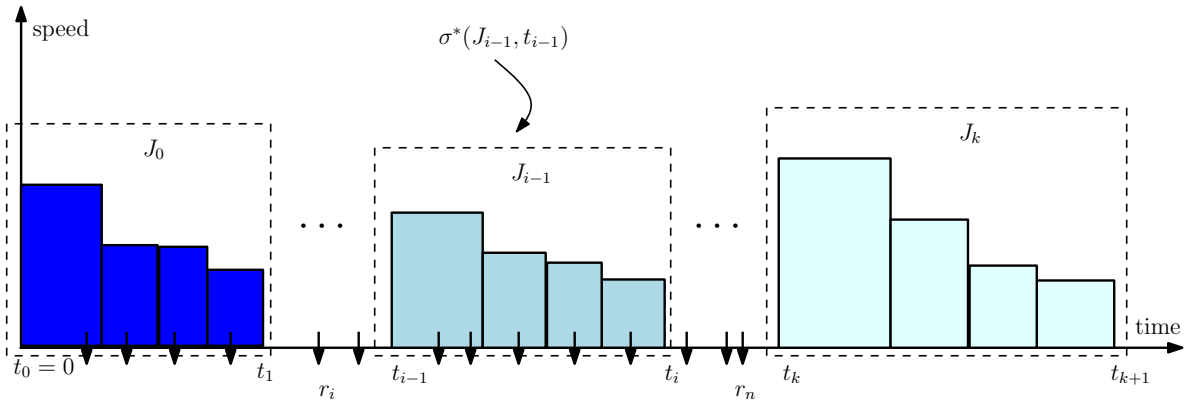


Figure 2.4: The structure of a schedule computed by Algorithm ALE.

Now, we turn our attention in the online version of the aggregated variant and we derive a 2-competitive online algorithm for the AGGR-LATENESS problem. The algorithm schedules the jobs in a number of phases by repeatedly applying the optimal offline algorithm for the AGGR-LATENESS problem, when all jobs have a common release date. This approach was introduced in [95]. We denote by $\sigma^*(J, t)$ the optimal schedule of a set of jobs J with a common release date t.

Next, we analyze the competitive ratio of the Algorithm ALE. The structure of a schedule produced by Algorithm ALE is illustrated in Figure 2.4.

Theorem 2.5 *Algorithm ALE is 2-competitive for the online version of the* AGGR-LATENESS

problem.

Proof. Assume that Algorithm ALE produces a schedule in $\ell+1$ phases. Recall that the jobs of the i-th phase, i.e., the jobs in J_i, are released during $(t_{i-1}, t_i]$ and scheduled as in $\sigma^*(J_i, t_i)$. Let $L_{\max,i} + \lambda E_i$ be the cost of $\sigma^*(J_i, t_i)$, where $L_{\max,i}$ is the maximum lateness among the jobs in J_i and E_i be the energy consumed by the jobs of J_i. The objective value of the algorithm's schedule is

$$\text{SOL} = \max_{0 \le i \le \ell} \{L_{\max,i}\} + \lambda \sum_{i=0}^{\ell} E_i \tag{2.22}$$

Now, we consider the optimal schedule. To lower bound the objective value OPT of an optimal schedule, we round down the release dates of the jobs; the release dates of the jobs in phase i, are rounded down to t_{i-1}. Let σ_d^* and OPT_d be the optimal schedule for the rounded instance and its cost, respectively. Clearly, any feasible schedule for the initial instance is also feasible for the rounded one. Thus, $\text{OPT} \geq OPT_d$.

To lower bound OPT_d we consider a restricted speed-scaling scheduling problem, i.e., a problem where each job can only be executed by a subset of the available processors. The instance of this problem consists of $\ell+1$ available speed-scalable processors $M_0, M_1, \ldots, M_\ell$ and the set of jobs J, with their release dates rounded down, as before. Jobs in J_0 can only be assigned to the processor M_0 and every job in J_i can only be executed by one of the processors M_0 or M_i, $1 \le i \le \ell$. Moreover, it is required that all jobs in J_i, $0 \le i \le \ell$, are executed by the same processor. Let σ_m^*, OPT_m be the optimal schedule and its cost, respectively, for this restricted problem. Obviously, $\text{OPT}_d \geq \text{OPT}_m$ since σ_d^* is feasible for the restricted scheduling problem.

Let us now describe an optimal schedule σ_m^*. Through a simple exchange argument, it can be shown that the jobs of J_i, $0 \le i \le \ell$, in an optimal schedule, are executed by the processor M_i. Moreover, jobs in J_i, for $1 \le i \le \ell$, are scheduled according to $\sigma^*(J_i, t_{i-1})$, while for $i = 0$, according to $\sigma^*(J_0, t_0)$. Assume that the maximum lateness of the above schedule, is attained by a job of the set J_k, $0 \le k \le \ell$, in the processor M_k. So, let $L_{\max}^* = L_{\max,k}^*$, where $L_{\max}^*$, $L_{\max,k}^*$ is the maximum lateness of the schedules $\sigma_m^*, \sigma^*(J_i, t_{i-1})$, respectively. Let E_i^* be the energy consumption of schedule $\sigma^*(J_i, t_{i-1})$. Then,

$$\text{OPT}_m = L_{\max,k}^* + \lambda \sum_{i=0}^{\ell} E_i^* \tag{2.23}$$

By considering the schedules $\sigma^*(J_i, t_{i-1})$ and $\sigma^*(J_i, t_i)$, it can be easily shown that $L_{\max,i}^* = L_{\max,i} - (t_i - t_{i-1})$ and $E_i^* = E_i$. Then, by (2.22) and (2.23) it yields that $\text{OPT}_m = \text{SOL} - (t_k - t_{k-1})$. Note that $t_k - t_{k-1}$ is the total processing time of the jobs in J_{k-1}, in the schedule produced by ALE, which is equal to the processing time of the jobs in J_{k-1} in σ_m^*. Recall also that the last job of each set J_i attains $L_{\max,i}$. Thus, $t_k - t_{k-1} \le L_{\max,k-1}^* \le \text{OPT}$.

Therefore, SOL $\leq$ 2OPT and Algorithm ALE is 2-competitive for the AGGR-LATENESS problem. ∎

Regarding the tightness of Algorithm ALE, we are able to construct an example of competitive ratio equal to $\frac{5}{3}$. Consider an instance with two jobs $1, 2$, with $v_1 = (K-1)s_1, v_2 = s_1$, $r_1 = 0, r_2 = \frac{1}{K}$, $q_1 = 1, q_2 = K$, where K is a big positive constant and $s_1 = \left(\frac{1}{\lambda(\beta-1)}\right)^{\frac{1}{\beta}}$. Moreover, let $\lambda \geq 0$ and $\beta = 3$. Algorithm ALE will schedule job 1 first, in speed s_1, and job 2 second, also in speed s_1 and the total energy consumption will be equal to $E = (v_1 + v_2)s_1^{\beta-1} = Ks_1^{\beta} = K\frac{1}{\lambda(\beta-1)}$. Thus, the total cost will be equal to SOL $= L_2 + \lambda E = 2K + K\frac{1}{\beta-1} = \frac{5}{2}K$. An optimal schedule must execute both jobs after r_2, in the EDD order, with job 1 scheduled last. In this case, the minimum cost is attained by executing both jobs in speed s_1, by applying the optimal offline algorithm for the common release date $\frac{1}{K}$. Hence, OPT $= r_2 + \frac{v_2}{s_1} + q_2 + \lambda E = \frac{1}{K} + K + 1 + \frac{K}{2} = \frac{1}{K} + 1 + \frac{3K}{2}$. As K becomes bigger, it holds that SOL $\rightarrow \frac{5}{3}$OPT.

2.6 Concluding remarks

We presented positive and negative results for the offline and online energy-aware versions of the classical maximum lateness scheduling problem. These results, along with the existing literature on energy-aware versions of other problems (e.g., makespan, total flow time) contribute on the direction of relating the computational complexity (polynomial or $\mathcal{NP}$-hard) of energy-aware scheduling problems with their classical versions. For polynomial algorithms, the most promising approach consists of deducing strong structural properties of optimal schedules by applying the KKT conditions on a convex programming formulation of the problem (see also in [25, 90, 23]). Towards this direction, an important note is to focus on proving first some basic optimality properties (such as an order of jobs' execution in the optimal schedule) that lead to a convex programming formulation of the problem. Actually, even if it is complicated to derive a fast combinatorial algorithm, this is already a proof of a polynomial algorithm with arbitrary precision. For $\mathcal{NP}$-hardness results, existing $\mathcal{NP}$-hardness reductions of the corresponding classical scheduling problems can be adapted by forcing all jobs to be executed with speed equal to one and considering the processing times as work volumes.

However, two interesting problems that are open concerning their computational complexity are the following: (i) find a *preemptive* schedule of a set of jobs, with arbitrary work volumes and arbitrary release dates on a single speed-scalable processor, in order to minimize their total completion time under a given energy budget, and (ii) find a *non-preemptive* schedule of a set of jobs, with arbitrary work volumes and arbitrary weights, released at time zero, on a single speed-scalable processor, in order to minimize their total weighted completion time under a given energy budget.

For problem (i), when all jobs are released at time zero, it is easy to derive an optimal

polynomial time algorithm as stated in the following theorem, by a simple exchange argument and by applying the KKT conditions to a convex programming formulation of the problem.

Theorem 2.6 *Consider the problem of scheduling a set of jobs, with arbitrary work volumes, released at time zero, on a single speed-scalable processor, to minimize their total completion time, for a given energy budget. There is a unique optimal schedule such that,*
(i) The jobs are scheduled in non-increasing order of work volumes, i.e., $v_j \leq v_{j+1},\ 1 \leq j \leq n$.
(ii) The speed of the j*-th job is equal to* $s_j = \left(\frac{n-j+1}{\rho(\beta-1)}\right)^{\frac{1}{\beta}}$, *and*
(iii) ρ *is a positive variable, equal to* $\rho = \frac{1}{\beta-1}\left(\frac{\sum_j v_j (n-j+1)^{\frac{\beta-1}{\beta}}}{E}\right)^{\frac{\beta}{\beta-1}}$.

For arbitrary release dates, Pruhs et al. [90] proposed an optimal polynomial time algorithm in the case of unit-work jobs, scheduled according to the First Come First Served (FCFS) rule, i.e., in non-decreasing order of release dates. In fact, the authors studied the problem under the total flow time objective, which, in terms of optimality, is equivalent to the total completion time. When jobs have arbitrary release dates, although in the classical scheduling setting the problem can be solved optimally by applying the Shortest Remaining Processing Time First (SRPT) rule [18]. However, in the energy-aware context this is not the case, mainly due to the fact that the job with the smallest remaining work depends on the previous scheduling decisions (see the discussion in Section 4 of [90] and Section 2.6 of [64]). The best result that has been proposed so far is a $(1+\epsilon)^{\beta}$-energy $O(\frac{1}{\epsilon})$-approximate polynomial algorithm that uses $(1+\epsilon)^{\beta}$ energy augmentation, for $\epsilon \in (0,1)$.

For problem (ii), as mentioned in Section 2.1, Megow and Verschae [82] proposed a PTAS, while also proving that the problem is equivalent to the single processor scheduling problem with the objective of minimizing the $\sum_{j\in\mathcal{J}} w_j (C_j)^{\frac{\beta-1}{\beta}}$. However, the complexity of the latter problem remains open (see in [63, 101] for approximation results based on the Smith's rule [100]).

Finally, another direction for future work concerns the use of energy augmentation for the online case of the budget variant of the problem, in order to overcome the fact that there is no $O(1)$-competitive deterministic algorithm (Theorem 2.3). Moreover, as the proposed Algorithm ALE for the aggregated variant (in Section 2.5) is not tight, it would be nice to improve its analysis or find a tight counter-example.

Chapter 3

Energy-efficient scheduling of MapReduce jobs

We consider a set $\mathcal{J} = \{1, 2, \ldots, n\}$ of n MapReduce jobs to be executed on a set $\mathcal{P} = \{1, 2, \ldots, m\}$ of m speed-scalable processors. Each job is associated with a positive weight w_j and a release date r_j and consists of a set of Map tasks and a set of Reduce tasks that are preassigned to the m processors. We denote by $\mathcal{T}$ the set of all tasks of all jobs, and by $\mathcal{M}$ and $\mathcal{R}$ the sets of all Map and Reduce tasks, respectively. Each task $T_{i,j} \in \mathcal{T}$ is associated with a non-negative work volume $v_{i,j}$.

We consider each job having at least one Map and one Reduce task and that each job has at most one task, either Map or Reduce, assigned to each processor. Map or Reduce tasks can run simultaneously on different processors, while the following precedence constraints hold for each job: every Reduce task cannot start its execution before the completion of all Map tasks of the same job. In other words, as shown in Figure 3.1, there is a complete bipartite precedence graph for the tasks of each MapReduce job.

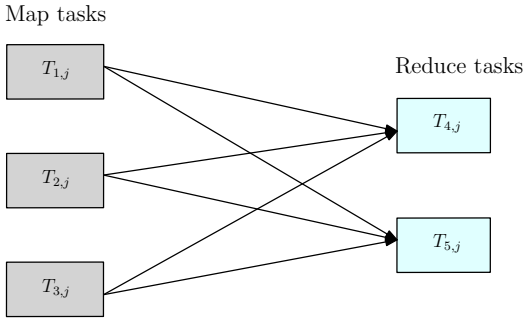


Figure 3.1: The precedence graph among tasks of a MapReduce job j consisting of 3 Map tasks and 2 Reduce tasks.

For a given schedule we denote by C_j and $C_{i,j}$ the completion times of each job $j \in \mathcal{J}$

and each task $T_{i,j} \in \mathcal{T}$, respectively. Note that, due to the precedence constraints of Map and Reduce tasks, $C_j = \max_{T_{i,j} \in \mathcal{R}}\{C_{i,j}\}$. By $C_{\max} = \max_{j \in \mathcal{J}}\{C_j\}$ we denote the makespan of the schedule, i.e., the completion time of the job which finishes last. Let also, $w_{\min} = \min_{j \in \mathcal{J}}\{w_j\}$, $v_{\min} = \min_{T_{i,j} \in \mathcal{T}}\{v_{i,j} : v_{i,j} > 0\}$, $w_{\max} = \max_{j \in \mathcal{J}}\{w_j\}$, $r_{\max} = \max_{j \in \mathcal{J}}\{r_j\}$ and $v_{\max} = \max_{T_{i,j} \in \mathcal{T}}\{v_{i,j}\}$.

We combine this abstract model for MapReduce scheduling with the speed scaling mechanism for energy saving described in Section 1.1. Recall that due to the convexity of the speed-to-power function, a key property of our problem is that each task runs at a constant speed during its whole execution (see Lemma 2.1). So, if a task $T_{i,j}$ is executed at a speed $s_{i,j}$, the time needed for its execution (processing time) is equal to $p_{i,j} = \frac{v_{i,j}}{s_{i,j}}$ and its energy consumption is $E_{i,j} = \frac{v_{i,j}}{s_{i,j}} s_{i,j}^{\beta} = v_{i,j} s_{i,j}^{\beta-1}$.

Moreover, we are given an energy budget E and the goal is to schedule *non-preemptively* all the tasks to the m processors, so as to minimize their total weighted completion time, i.e., $\sum_{j \in \mathcal{J}} w_j C_j$, without exceeding the energy budget E. We refer to this problem as NRG-MapReduce problem.

3.1 Related work

As our work generalizes on existing models proposed for classical MapReduce scheduling, we present first the results concerning these models.

Over the last decade there has been a great deal of empirical work in MapReduce scheduling concerning the experimental evaluation of scheduling heuristics in finding good trade-offs among different practical needs (see e.g., [112] for a review). On the other hand, in terms of the commonly studied metrics in scheduling theory, only three theoretical works are known [38, 40, 83]. These represent abstract models for MapReduce scheduling that are closely-related to the well-known open-shop and flow-shop scheduling models, under the natural objective of minimizing the total weighted completion time of a set of MapReduce jobs in a schedule.

A first primitive theoretical framework for MapReduce scheduling was proposed by Chang et al. [38]. According to their model, the jobs' tasks are preassigned to processors, while there is no distinction and no dependencies between Map and Reduce tasks of each job. The goal is to schedule the tasks *non-preemptively* into processors so as to minimize the total weighted completion time of the jobs. However, this model falls into an extensively studied version of the open-shop model, referred as *concurrent open-shop* (or *order scheduling*) problem in scheduling literature, where the tasks of the same jobs can be processed concurrently (see Subsection 1.2.2). Concurrent open-shop is known to be strongly $\mathcal{NP}$-hard [92], even when all jobs have common release dates, unit-weights and there are only two processors available, while recently it was proved to be inapproximable within a factor better than 2, assuming the Unique Games Conjecture [24, 70]. Mastrolilli et

al. [80] proposed an efficient (primal-dual) combinatorial algorithm that achieves the factor of 2. However, some standard LP-relaxations for the classical single processor problem of minimizing the total weighted completion time (see, e.g., [39]) can be also applied to the concurrent open-shop problem. These extensions lead to approximation algorithms of ratio 3, for arbitrary release dates and 2, for common release dates (see [52, 77]). Similar results were also proposed in [38] under the MapReduce scheduling context leading to approximation algorithms of ratios 3 and 2 for arbitrary and common release dates, respectively.

Extending the above model, Chen et al. [40], proposed a more realistic approach which takes into account the dependencies among Map and Reduce tasks, and derived an LP-based 8-approximation algorithm in the case where all jobs are released at time zero and the goal is to minimize their total weighted completion time. The algorithm schedules the tasks *non-preemptively*, in non-decreasing order of their completion times in an optimal solution to an LP-relaxation, with respect to the precedences among tasks. The LP-relaxation is similar to the one proposed in Section 2 of [60], for the problem of minimizing the total weighted completion time of a set of jobs on a single processor. Moreover, under some simplified abstractions, the authors managed to model the data shuffle (i.e., the transmission of intermediate data of a job from Map tasks to Reduce tasks) in MapReduce computations, and presented a 58-approximation algorithm for this generalization.

In a third model, Moseley et al. [83] introduced the relation of MapReduce scheduling with the two-stage flexible flow-shop (FFS) problem. In the FFS problem, we are given a set of jobs, each consisting of a number of tasks (each task corresponds to a stage), to be scheduled on a set of multiple identical processors for each stage. The jobs should be executed in the same fixed order of stages, without overlaps between tasks (stages) of the same job. Known results for the FFS problem concern the two-stage case on parallel identical processors. More specifically, for the makepsan objective a PTAS is known [94], while for the total completion time objective, a simple 2-approximation algorithm was proposed in [55], for the special case where each stage has to be executed on a single processor. For the latter case, which is known to be strongly $\mathcal{NP}$-hard [50], Moseley et al. [83] proposed a QPTAS which becomes a PTAS for a fixed number of tasks' processing times. For the MapReduce setting, the authors [83] extended the two-stage FFS problem so that, at the first stage the set of Map tasks are going to be executed concurrently on a set of parallel identical processors (say Map processors) while in the second stage, the set of Reduce tasks are going to be executed concurrently on a set of parallel identical processors (say Reduce processors). The two sets of processors might be indistinguishable while, all tasks are available at time zero and jobs have unit-weights. They presented a greedy 12-approximation algorithm which constructs a *non-preemptive* schedule by merging two individual schedules for the Map and the Reduce tasks, with respect to the precedence constraints among them. They also studied the online version of the problem, when preemption is allowed, and proposed a $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive online algorithm, for any $\epsilon \in (0,1)$, under $(1+\epsilon)$

speed augmentation. Moreover, they studied the more general environment of unrelated processors and focused on the special case of the problem, where each job has a single Map and a single Reduce task. Using similar ideas, as in the identical processors' case, they presented a 6-approximation algorithm. Finally, for the online version of the latter case, when *preemption* is allowed, they proposed a $(1+\epsilon)$-speed $O(1/\epsilon^5)$-competitive online algorithm, for any $\epsilon \in (0,1)$, under $(1+\epsilon)$ speed augmentation.

As already discussed in Section 1.2.3, our work focuses on the theoretical study of energy-aware MapReduce scheduling, with the objective of minimizing the total weighted completion time of a set of MapReduce jobs, for a given energy budget. Actually, the processor environment is closely-related to the concurrent open-shop model (with precedences among Map and Reduce tasks) combined with multiple speed-scalable processors. Although we are not aware of any theoretical work in this setting, a huge body of research work has been proposed for scheduling on multiple speed-scalable processors under different processor environments. Next, we present some of the main results relevant to our objectives.

Megow and Verschae [81], proposed a $(2+\epsilon)$-approximation algorithm for minimizing the total weighted completion time, for a given energy budget, on a set of multiple identical speed-scalable processors, where jobs have arbitrary release dates and *preemption* is allowed. Bampis et al. [20] proposed an optimal polynomial time algorithm for the minimization of a linear combination of the total weighted completion time of the jobs and the total energy consumption, on multiple identical speed-scalable processors, without preemptions and release dates. Angel et al. [11], proposed a randomized $2(1+\epsilon)$-approximation algorithm, for $\epsilon \in (0,1)$, for minimizing the total weighted completion time, for a given energy budget, on unrelated parallel processors operating under a model where the processing time and the energy consumption of each job depend on both the processor on which the job is executed and the speed that is used by the processor. In fact, every processor can change its speed dynamically, choosing among a finite set of speeds.

Pruhs et al. [91] studied the *non-preemptive* makespan minimization problem, for a given energy budget, on multiple identical speed-scalable processors, in the presence of precedence constraints. They proposed a $O(\log^{1+2/\beta} m)$-approximation algorithm for the problem. They also gave a PTAS for the case with no precedence constraints. Recently, Bampis et al. [21] significantly improved the latter result (where jobs have precedence constraints) proposing a $(2-1/m)$-approximation algorithm for the problem. They moreover, proposed a general framework for designing approximation algorithms for makespan minimization variants on (single or multiple) speed-scalable processors, for a given energy budget, and presented a 2-approximation algorithm for the open-shop environment on multiple speed-scalable processors.

Finally, Angel et al. [14] proposed a $2(\beta+1)$-approximation algorithm for the maximization of the weighted throughput, for a given energy budget, on a set of multiple unrelated

speed-scalable processors, when jobs have arbitrary release dates, the preemption of the jobs is allowed but not their migration [1]. In fact, their algorithm is polynomial on the input size and on $1/\epsilon$, while also violates the energy budget by $(1+\epsilon)$, for $\epsilon \in (0,1)$. They also proposed optimal polynomial and pseudopolynomial algorithms for several special cases of the problem.

3.2 Contribution

In this chapter we adopt the model proposed by Chen et al. [40] while extending it to multiple speed-scalable processors environment, with a given budget on energy consumption. We study two different algorithmic approaches for the NRG-MapReduce problem. The first, in Section 3.3, is a convex programming approach where the main idea is the following: *if we are aware of the execution order of the jobs in a schedule, then we are able to compute their processing times.* In fact, by considering an order of execution for the jobs we are able to formulate a convex programming formulation of the problem. To maintain a reasonable (polynomial on the input size) number of constraints, we introduce a rough lower bound concerning the tasks completion times which is based on their precedence constraints. As a result, our convex program might produce infeasible schedules. So, in order to ensure feasibility, we apply a greedy algorithm that uses the processing times computed by the convex program, while respecting both the given order of execution and the precedences between Map and Reduce tasks. We test the above algorithm for two standard scheduling policies, the First Come First Served (FCFS) and the Highest Density First (HDF), and we compare their solutions for different random instances. Moreover, by using the solution of the convex relaxation as lower bound on the optimal solution of the NRG-MapReduce problem, with respect to either FCFS or HDF orders, we extract fairly good approximation ratios for our algorithm, for both scheduling policies. However, as we prove, there are instances for which the optimal solutions, with respect to the FCFS or HDF orders, are very far (more than n) from the optimal solution to the NRG-MapReduce problem.

In order to derive a good approximation ratio for the NRG-MapReduce problem, in Section 3.4, we propose a second approach based on a linear programming formulation of our problem, which results in a constant approximation ratio. To obtain our result we use a combination of ingredients. We start by discretizing both the time horizon as well as the range of possible processors' speed values, imposing only a small loss in the objective value of the schedule, compared to the optimal one. This is done by computing an upper bound on the makespan while also, upper and lower bounds on the speed values of each task in an optimal schedule. These discretizations allow us to setup an interval-indexed LP-relaxation of our problem, using as parameters the completion times of jobs and the speeds of the tasks. Our LP is inspired by ideas proposed for the classical single processor

1. In a migratory schedule each job may be executed by more than one processors, with no parallel execution.

problem (see [39]) and extends them to our problem, for multiple speed-scalable processors. In fact, having computed an optimal solution to LP, we extend the idea of list scheduling in the order of α-points, so as to find a trade-off between the energy and the total weighted completion time, as function of α. We prove that this idea leads to a $O(1)$-energy $O(1)$-approximation algorithm for the NRG-MapReduce problem. Note that, α-points have been also used in the context of speed scaling in [36].

In Section 3.5, we deal with the classical MapReduce scheduling and generalize the models proposed so far [38, 40, 83]. The basic idea behind a MapReduce job is that each job is split into a large number of Map and Reduce tasks that can be executed in parallel (see e.g., [4, 69, 3]). In addition, a significant cost when running a MapReduce job is that of data shuffle, i.e., the time for transmitting the intermediate data from Map to Reduce tasks (communication cost). This cost affects crucially the performance of MapReduce systems (e.g., bandwidth bottleneck [40], high wall-clock time [103]) and usually dominates the computation cost of Map and Reduce tasks (see e.g. [4, 3]). In terms of scheduling, this makes the problem more intricate and important for system performance. So, we consider a general model taking into account the real constraints of MapReduce systems: (a) each job has multiple tasks in each stage; (b) the assignment of tasks to processors is flexible; (c) there are dependencies between Map and Reduce tasks; (d) the processors are unrelated to capture data locality; and (e) there is a significant communication cost for the data shuffle. Our goal is to find a *non-preemptive* schedule minimizing the objective of total weighted completion time for a set of MapReduce jobs.

More importantly, we present constant approximation algorithms which generalize the model proposed by Moseley et al. [83] on unrelated processors, towards two directions: we deal with jobs consisting of multiple Map and Reduce tasks and also incorporate the shuffle phase into our setting. As it has been observed in [83], new ideas and techniques are required for both these directions.

In Subsection 3.5.1, we present a 54-approximation algorithm for the Map-Reduce scheduling problem when jobs consist of multiple Map and Reduce tasks. The main idea of our algorithm is similar to the one proposed in [83] for single task jobs: first, we compute a schedule for only the Map (resp. Reduce) tasks and then, we merge the two schedules into a single one with respect to the task dependencies. However, in [83], since each job consists of a single Map and a single Reduce task, a schedule of only Map (resp. Reduce) tasks can be computed by applying the well known 3/2-approximation algorithm by Skutella [97], for the problem of minimizing the total weighted completion time of a set of jobs on unrelated processors. Instead, we formulate an interval-indexed LP-relaxation for the problem of minimizing the total weighted completion times separately for Map and Reduce tasks on unrelated processors. Our LP formulation is inspired by the one proposed by Hall et al. [60] for scheduling a set of single task jobs on unrelated processors under the same objective. However, in our problem, not all the tasks of each job contribute to the objective value, but

only the one that finishes last and this makes the analysis of such an LP more difficult. Recently, Correa et al. [42] proposed and analyzed a similar LP-relaxation for a more general problem, where, instead of jobs consisting of tasks, they are given a set of orders of jobs and the completion time of each order is specified by the completion of the job that finishes last. Since scheduling multi-task Map and Reduce jobs separately is quite similar to the setting considered in [42], we can use their approximation result for scheduling separately the Map and Reduce tasks. Next, we concatenate the two schedules into a single one respecting the task dependencies, extending the ideas in [83] so as to ensure that preemption is not allowed and our schedule respects the precedences between Map and Reduce tasks.

In Subsection 3.5.2, we incorporate the data shuffle phase into our model by introducing an additional set of *Shuffle tasks*, each one associated with a communication cost (transfer time). When the Shuffle tasks are scheduled on the same processors as the corresponding Reduce tasks, we are able to keep the same 54-approximation ratio for the Map-Shuffle-Reduce scheduling problem. Moreover, we also prove a 81-approximation ratio when the Shuffle tasks are allowed to be executed on different processors than their corresponding Reduce tasks. To the best of our knowledge, this is the most general setting of the FFS problem (with a special third stage) for which a constant approximation guarantee is known.

3.3 A convex programming approach

We are interested in natural list scheduling policies such as First Come First Served (FCFS) and Highest Density First (HDF). However, in our context we need to determine the speeds of every task in order to respect the energy budget. Therefore, we propose a convex programming relaxation of our problem, when an order of execution of the jobs is given as input.

3.3.1 The convex program

$$\textbf{(CP}_\sigma\textbf{)} : \text{minimize} \sum_{j \in \mathcal{J}} w_j C_j$$

subject to :

$$\sum_{T_{i,j} \in \mathcal{T}} \frac{v_{i,j}^{\beta}}{p_{i,j}^{\beta-1}} \leq E \tag{3.1}$$

$$r_{j'} + \sum_{k=j'}^{j} p_{i,k} \leq C_{i,j}, \qquad \forall T_{i,j}, T_{i,j'} \in \mathcal{T}, j' \prec j \tag{3.2}$$

$$C_{i',j} + p_{i,j} \leq C_{i,j}, \qquad \forall T_{i,j} \in \mathcal{R}, T_{i',j} \in \mathcal{M} \tag{3.3}$$

$$C_{i,j} \leq C_j, \qquad \forall T_{i,j} \in \mathcal{T} \tag{3.4}$$

$$s_{i,j}, C_{i,j}, C_j \geq 0, \qquad \forall T_{i,j} \in \mathcal{T}, j \in \mathcal{J}$$

Let $\sigma = \langle 1, 2, \ldots, n \rangle$ be a given order of the jobs. Consider now the restricted version of the NRG-MapReduce problem where, for each processor $i \in \mathcal{P}$, the tasks are forced to be executed according to this order. We shall refer to this problem as the NRG-MapReduce(σ) problem. Note that, the order is the same for all processors. We write $j \prec j'$ if job $j \in \mathcal{J}$ precedes job $j' \in \mathcal{J}$ in σ. We propose a convex program that considers the order σ as input and returns a solution that is a lower bound to the optimal solution for the NRG-MapReduce(σ) problem.

In order to formulate our problem as a convex program, for each task $T_{i,j} \in \mathcal{T}$, let $p_{i,j}$ be a variable that corresponds to its processing time and $C_{i,j}$ a variable that determines its completion time. Let also C_j, $j \in \mathcal{J}$, be the variable that corresponds to the completion time of job j. Then, (CP$_\sigma$) is a convex programming relaxation of the NRG-MapReduce(σ) problem.

The objective function of (CP$_\sigma$) is to minimize the weighted completion time of all jobs. Constraint (3.1) guarantees that the energy budget is not exceeded; note that we have substituted the energy consumption $E_{i,j}$ of each task $T_{i,j}$ by its equivalent $E_{i,j} = p_{i,j} s_{i,j}^{\beta} = p_{i,j} (\frac{v_{i,j}}{p_{i,j}})^{\beta}$, where $s_{i,j} = \frac{v_{i,j}}{p_{i,j}}$ is the speed of task $T_{i,j}$. Constraints (3.2) and (3.3) give lower bounds on the completion time of each task $T_{i,j} \in \mathcal{T}$, based on the release dates and the precedence constraints, respectively. Note that, if we do not consider precedences between the tasks, then (CP$_\sigma$) will return the optimal value of the objective function, instead of a lower bound on it, as constraints (3.2) describe in a complete way the completion times of the tasks. However, this is not true for constraints (3.3) which are responsible for the precedence constraints. Finally, constraints (3.4) ensure that the completion time of each job is the maximum over the completion times among all of its tasks.

3.3.2 Experimental evaluation of scheduling policies

As the optimal solution to (CP_σ) does not necessarily describe a feasible schedule, we need to apply an algorithm that uses the processing times found by (CP_σ) and the order σ so as to create a feasible schedule for the NRG-MapReduce(σ) problem, and hence for the NRG-MapReduce problem. This can be achieved by Algorithm EMR_σ, which at every time instant where a processor becomes available, schedules a task that is been released but not yet executed, while respecting the precedences among Map and Reduce tasks.

Algorithm EMR_σ: a heuristic for the NRG-MapReduce problem.

1: Compute an optimal solution to $(\bar{p}_{i,j}, \bar{C}_{i,j}, \bar{C}_j)$ to (CP_σ).
2: **for** each time where a processor $i \in \mathcal{P}$ becomes available **do**
3: Select a task, say $T_{i,j}$, of highest priority such that: $T_{i,j}$ is already released and has not yet been executed and if $T_{i,j}$ is a Reduce task, then all Map tasks of the same job must have been already completed at t.
4: Schedule $T_{i,j}$ non-preemptively, with processing time $p_{i,j}$.
5: Let $C_{i,j}$ be the completion time of task $T_{i,j}$.
6: **end for**
7: **for** each job $j \in \mathcal{J}$ **do**
8: Compute its completion time $C_j = \max_{i \in \mathcal{P}} C_{i,j}$.
9: **end for**

Now, we consider the following standard scheduling policies and we test Algorithm EMR_σ, with respect to the order indicated by each one of them.

First Come First Served (FCFS): for each pair of jobs $j, j' \in \mathcal{J}$, if $r_j < r_{j'}$ then $j \prec j'$ in σ. Highest Density First (HDF): for each pair of jobs $j, j' \in \mathcal{J}$, if $\frac{w_j}{\sum_{T_{i,j} \in j} v_{i,j}} > \frac{w_{j'}}{\sum_{T_{i,j'} \in j'} v_{i,j'}}$ then $j \prec j'$ in σ.

We compare the FCFS and HDF policies with respect to the quality of the solution they produce for the NRG-MapReduce(FCFS) and NRG-MapReduce(HDF) problems, respectively. Our simulations have been performed on a machine with a CPU Intel Xeon X5650 with 8 cores, running at 2.67GHz. The operating system of the machine is a Linux Debian 6.0. We used Matlab with cvx toolbox. The solver used for the convex program is SeDuMi.

The instance of the problem consists of a matrix $m \times n$ that corresponds to the work of the tasks, two vectors of size n that correspond to the weights and the release dates of jobs, a precedence graph for the tasks of the same job, the energy budget and the value of β. Similarly with [40], the instance consists of $m = 50$ processors and up to $n = 25$ jobs. Each job has 20 Map and 10 Reduce tasks, which are preassigned at random to a different processor. The work of each Map task is selected uniformly at random from $[1, 10]$, while the work of each Reduce task $v_{i,j} \in \mathcal{R}$ is set equal to a random number in

$[1, 10]$ plus $\frac{3 \cdot \sum_{T_{i',j} \in \mathcal{M}} v_{i',j}}{|\{T_{i',j} \in \mathcal{M}\}|}$, taking into account the fact that Reduce tasks have more work to execute than Map tasks. The weight of each job is selected uniformly at random from $[1, 10]$ and the release date of a job, is given as a Bernoulli random variable with probability $1/2$ for every interval $(t, t+1]$. The energy budget that is used equals $E = 1000$, while β is set $\beta = 2$. We have also set the desired accuracy of the returned solution of the convex program to be equal to 10^{-7}. For each number of jobs, we have repeated the experiments with 10 different matrices. The results we present below, concern the average of these 10 instances. The benchmark and the code used in our experiments are freely available at `http://www.ibisc.univ-evry.fr/~vchau/research/mapreduce/`.

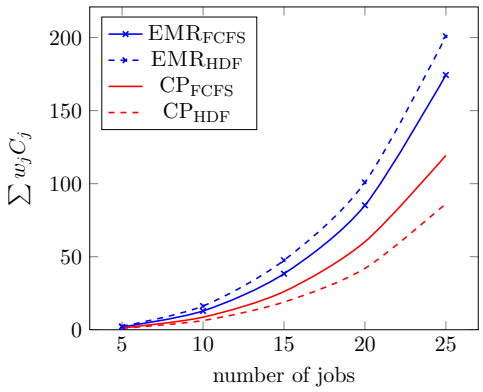


Figure 3.2: Comparing solutions for FCFS and HDF (scaled down by a factor of 10^3).

As shown in Figure 3.2 the heuristic based on FCFS outperforms the heuristic based on HDF. In fact, the first heuristic gives up to $16 - 21\%$ better solutions than the second one for different values of n. Surprisingly, the situation is completely inverse if we consider the corresponding solutions of the convex programs. More precisely, the convex programming relaxation using HDF leads to $26\% - 43\%$ smaller values of the objective function compared to the convex programming relaxation using FCFS. Moreover, we can observe that the ratio between the final solution of each heuristic with respect to the lower bound for the NRG-MapReduce(σ) problem given by the convex program is equal to 1.46 for FCFS and 2.43 for HDF; the variance is less than 0.1 in both cases.

Negative results: Concerning how close is an optimal solution for the NRG-MapReduce(σ) problem, if we use the FCFS or the HDF order, with respect to an optimal solution for the NRG-MapReduce problem, the following proposition gives negative results.

Proposition 3.1 *There are instances of the* NRG-MapReduce(FCFS) *and the* NRG-MapReduce(HDF) *problems, for which the optimal solutions are within a factor of* $\Omega(n)$ *from the optimal solution to the* NRG-MapReduce *problem.*

Proof. First, consider an instance of the NRG-MapReduce(FCFS) problem, consisting

of m processors and n jobs, where $m = n$. The release date of each job $j \in \mathcal{J}$ is $(j-1)\epsilon$, for a very small $\epsilon > 0$, and its weight $w_j = 1$. Each job $j \in \mathcal{J}$ consists of m tasks, one per processor. Moreover, the task $T_{i,j} \in \mathcal{T}$ is a Map task only if $i = j$; otherwise $T_{i,j}$ is a Reduce task. For each task $T_{i,i} \in \mathcal{M}$, let $v_{i,i} = 1$. For each task $T_{i,j} \in \mathcal{R}$, let $v_{i,j} = \epsilon$. Let also $E = 1$ and $\beta = 2$.

Note that, if $\epsilon \ll 1$ then the processing time of each Reduce task can be considered to be very small in both the optimal schedules for the NRG-MapReduce and the NRG-MapReduce(FCFS) problems. So, we can ignore the execution time and the energy consumption of the Reduce tasks. We only consider the precedence constraints that they imply.

In an optimal solution for the NRG-MapReduce problem, the Map task of job j starts at time $(j-1)\epsilon$. Due to the convexity and the fact that $w_j = 1$ for each $j \in \mathcal{J}$, we can assume that all Map tasks will be executed with the same speed; hence the processing time of each Map task is approximately equal to $\sqrt[\beta-1]{\frac{m}{E}} = m$, as $E = 1$ and $\beta = 2$. Thus, the completion time of each job is approximately equal to m, and hence $\mathrm{OPT} = O(m^2)$.

On the other hand, in an optimal solution for the NRG-MapReduce(FCFS) problem the Map tasks are not executed in parallel, as we are forced to respect the order and the precedence constraints. Ignoring again the processing times of the Reduce tasks, we can assume that the Map task of job j starts at the completion time of job $j-1$. In order to find the speed s_j of each Map task $T_{j,j} \in \mathcal{T}$ into an optimal solution for the NRG-MapReduce(FCFS) problem, we have to solve the following convex program.

$$\text{minimize} \quad \sum_{j=1}^{n} \frac{n-j+1}{s_j} \quad \text{subject to} \quad \sum_{j=1}^{n} s_j \leq E$$

The objective of this convex program corresponds to the one of the NRG-MapReduce(FCFS) problem for the given instance, while the constraint ensures that the selected speeds respect the energy budget. By applying the Karush-Kuhn-Tucker conditions to this program we get that $s_j = \frac{E \cdot (n-j+1)^{1/2}}{\sum_{i=1}^{n} (n-i+1)^{1/2}}$. By replacing this to the objective we get

$$\begin{aligned}
\mathrm{OPT}_{\mathrm{FCFS}} &= \sum_{j=1}^{n} \frac{n-j+1}{\frac{E \cdot (n-j+1)^{1/2}}{\sum_{i=1}^{n} (n-i+1)^{1/2}}} \\
&= \frac{1}{E} \left(\sum_{i=1}^{n} (n-i+1)^{1/2} \right)^2 \\
&= \frac{1}{E} \left(\sum_{i=1}^{n} i^{1/2} \right)^2 = O\left(\frac{n^3}{E} \right)
\end{aligned}$$

As $n = m$ and $E = 1$, the proposition follows.

Now, we consider a simplified instance for the NRG-MapReduce(HDF) problem,

which consists of only one processor and does not take into account Map and Reduce tasks and hence precedences. In this instance the critical issue is the release dates. For each job j, $1 \leq j \leq n-1$, we have $v_j = 1$, $w_j = 1$ and $r_j = 0$, while for the job n we have $v_n = 1-\epsilon$, $w_n = 1$ and $r_n = r$, where $r \in \mathbb{R}$ is a big number. Let $E = 1$ and $\beta = 2$.

In an optimal schedule for the NRG-MapReduce problem, the jobs $1, 2, \ldots, n-1$ are scheduled consecutively starting from time 0, while the job n is scheduled starting from time r. Let E_1 and E_2 be parts of the energy budget used for the execution of the jobs $1, 2, \ldots, n-1$ and n, respectively. Clearly, it holds that $E_1+E_2 = 1$. Hence, following similar analysis as for the NRG-MapReduce(FCFS) problem, the total weighted completion time of the jobs $1, 2, \ldots, n-1$ will be

$$\sum_{j=1}^{n-1} w_j C_j = O\left(\frac{n^3}{E_1}\right)$$

The processing time of job n is E_2, and hence its completion time is $C_j = r+E_2$. Therefore, for the optimal solution for the NRG-MapReduce problem we have that,

$$\begin{aligned} \text{OPT} &= O\left(\frac{n^3}{E_1}\right) + r + \frac{1}{E_2} \\ &= O\left(\frac{n^3}{E_1}\right) + r + \frac{1}{1-E_1} = r + O(n^3) \end{aligned}$$

as this function is minimized for $E_1 \simeq 1/2$.

On the other hand, in an optimal schedule for the NRG-MapReduce(HDF) problem, the jobs are scheduled starting from r according to the HDF order, i.e., $\langle n, 1, 2, \ldots, n-1 \rangle$. As we can choose an ϵ such that $\epsilon \ll 1$, we can assume that all jobs have the same work to execute. Then, following similar analysis as for the NRG-MapReduce(FCFS) problem, we have that $\text{OPT}_{SR} = nr + O(n^3)$.

As r can be arbitrary large, the proposition follows. ∎

3.4 A linear programming approach

Since our goal is to derive a provably good performance guarantee for the NRG-MapReduce problem, in this section we propose a $O(1)$-energy $O(1)$-approximation algorithm for the NRG-MapReduce problem. Our algorithm is based on a linear programming relaxation of the problem and it transforms the solution obtained by the linear program to a feasible schedule for the NRG-MapReduce problem using the technique of α-points. Note that, by allowing energy augmentation we are able to describe a trade-off between energy and performance. Moreover, we can derive a constant-factor approximation ratio (without energy augmentation) for the NRG-MapReduce problem by appropriately choosing some

parameters.

3.4.1 Discretization of Speeds

To give a linear programming formulation of our problem, we first discretize the possible speed values. In order to do this, we need to compute an upper and a lower bound on the speed of each task given by the following propositions which bound the length of an optimal schedule and the possible speed values.

Proposition 3.2 *The makespan of any optimal schedule for the* NRG-MapReduce *problem is at most*

$$t_{\max} = \frac{w_{\max}}{w_{\min}} \left(n r_{\max} + n(n+1) \left(\frac{|\mathcal{T}| \cdot v_{\max}^{\beta}}{E} \right)^{\frac{1}{\beta-1}} \right)$$

Proof. Consider an optimal schedule for the NRG-MapReduce problem. By definition, we have that $C_{\max} = \max_{j \in \mathcal{J}} \{C_j\}$. Hence, it holds that $w_{\min} C_{\max} \leq \sum_{j \in \mathcal{J}} w_j C_j$.

In order to give an upper bound to $\sum_{j \in \mathcal{J}} w_j C_j$, consider an instance of our problem where the weight w_j and the release date r_j of each job $j \in \mathcal{J}$ are rounded up to $w_{\max}$ and $r_{\max}$, respectively. Moreover, assume that in this instance all tasks have work equal to $v_{\max}$.

Consider now an arbitrary order $\{1, 2, \ldots, n\}$ of the jobs. We create a feasible schedule S for the modified instance as follows. All tasks run with the same speed $s = \left(\frac{E}{|\mathcal{T}| \cdot v_{\max}} \right)^{1/(\beta-1)}$, hence each task has a processing time $p = \frac{v_{\max}}{s}$. Note that this speed allows us to execute all tasks without exceeding the energy budget. As all tasks have the same processing time, we can consider the time horizon partitioned into time slots of length p starting from $r_{\max}$. For each job j, $1 \leq j \leq n$, we execute its Map tasks at time $r_{\max} + (2j-2)p$ and its Reduce tasks at time $r_{\max} + (2j-1)p$. Then, for the objective value $\sum_{j \in \mathcal{J}} w_{\max} C_j^S$ of this schedule it holds that

$$\begin{aligned} \sum_{j \in \mathcal{J}} w_{\max} C_j^S &= w_{\max} \sum_{j=1}^{n} (r_{\max} + 2jp) \\ &= w_{\max} \left(n r_{\max} + n(n+1) \frac{v_{\max}}{s} \right) \end{aligned}$$

The objective value of schedule S is clearly an upper bound on the objective value $\sum_{j \in \mathcal{J}} w_j C_j$ of an optimal schedule for the initial instance and the proposition follows. ∎

Proposition 3.3 *For the speed $s_{i,j}$ of any task $T_{i,j} \in \mathcal{T}$ in the optimal schedule it holds that*

$$\frac{v_{i,j}}{t_{\max}} \leq s_{i,j} \leq \left(\frac{E}{v_{i,j}} \right)^{\frac{1}{\beta-1}}$$

Proof. The processing time $p_{i,j}$ of a task $T_{i,j} \in \mathcal{T}$ in an optimal schedule cannot exceed

the maximum completion time, that is $p_{i,j} = \frac{v_{i,j}}{s_{i,j}} \leq C_{\max}$ and, since by Proposition 3.2 it holds that $C_{\max} \leq t_{\max}$, the lower bound follows.

The energy consumption of any task cannot exceed the energy budget, that is $E_{i,j} = v_{i,j} s_{i,j}^{\beta-1} \leq E$ and the upper bound follows. ∎

Let $s_L = \frac{v_{\min}}{t_{\max}}$ and $s_U = \left(\frac{E}{v_{\min}}\right)^{1/(\beta-1)}$ be an upper and a lower bound, respectively, on the speed of any task. Given these bounds, we discretize the interval $[s_L, s_U]$ geometrically. In other words, we assume that the processors can only run according to one of the following speeds: $s_L, s_L(1+\epsilon), s_L(1+\epsilon)^2, \ldots, s_L(1+\epsilon)^k$, where k is the smallest integer such that $s_L(1+\epsilon)^k \geq s_U$. Note that $k = \lceil \log_{1+\epsilon} \frac{s_U}{s_L} \rceil$ and hence the number of possible speeds is polynomial to the size of the instance and to $1/\epsilon$. We denote by $\mathcal{V} = \{s_L(1+\epsilon)^\ell | \epsilon > 0, 0 \leq \ell \leq k\}$ the set of all possible discrete speed values. Let also $s_{\max} = s_L(1+\epsilon)^k$. Then, by loosing a factor of $(1+\epsilon)$ with respect to an optimal solution, we can prove the following.

Lemma 3.1 *There is a feasible $(1+\epsilon)$-approximate schedule for the* NRG-MapReduce *problem in which each task $T_{i,j} \in \mathcal{T}$ runs at a speed $s \in \mathcal{V}$.*

Proof. Let an optimal schedule for our problem and consider the speed of each task $T_{i,j} \in \mathcal{T}$ rounded down to the closest $s_L(1+\epsilon)^\ell$ value. As the speeds are decreased, the energy consumption of $\mathcal{S}$ does not exceed E. Moreover, the execution time of all tasks, and hence the completion time of every job and the optimal objective value increase by a factor at most $(1+\epsilon)$. ∎

Henceforth we will consider the NRG-MapReduce problem in which each task $T_{i,j} \in \mathcal{T}$ runs at a single speed $s \in \mathcal{V}$. We call this version of the problem DS-NRG-MapReduce.

3.4.2 Linear Programming Relaxation

In what follows we give an interval-indexed linear programming relaxation of the DS-NRG-MapReduce problem. In order to do this, we discretize the time horizon of an optimal schedule as follows. By Proposition 3.2, in any optimal schedule, all jobs are executed during the interval $(0, t_{\max}]$. We partition $(0, t_{\max}]$ into the intervals $(0, \lambda], (\lambda, \lambda(1+\delta)], (\lambda(1+\delta), \lambda(1+\delta)^2], \ldots, (\lambda(1+\delta)^{u-1}, \lambda(1+\delta)^u]$, where $\delta > 0$ is a small constant, $\lambda > 0$ is a constant that we will define later, and u is the smallest integer such that $\lambda(1+\delta)^{u-1} \geq t_{\max}$. Let $\tau_0 = 0$ and $\tau_t = \lambda(1+\delta)^{t-1}$, for $1 \leq t \leq u+1$. Moreover, let $I_t = (\tau_t, \tau_{t+1}]$, for $0 \leq t \leq u$, and $|I_t|$ be the length of the interval I_t, i.e., $|I_0| = \lambda$ and $|I_t| = \lambda\delta(1+\delta)^{t-1}$, $1 \leq t \leq u$. Note that, the number of intervals is polynomial to the size of the instance and to $1/\delta$, as $u = \lceil \log_{1+\delta} \frac{t_{\max}}{\lambda} \rceil + 1$.

Let $p_{i,j,s} = \frac{v_{i,j}}{s}$ be the potential processing time for each task $T_{i,j} \in \mathcal{T}$ if it is executed entirely with speed $s \in \mathcal{V}$. For each $T_{i,j} \in \mathcal{T}$, $t \in \{0, 1, \ldots, u\}$ and $s \in \mathcal{V}$, we introduce a variable $y_{i,j,s,t}$ that corresponds to the portion of the interval I_t during which the task $T_{i,j}$

$$\textbf{(LP)} : \text{minimize} \sum_{j \in \mathcal{J}} w_j C_j$$

subject to :

$$\sum_{s \in \mathcal{V}} \sum_{t=0}^{u} \frac{y_{i,j,s,t}|I_t|}{p_{i,j,s}} = 1, \qquad \forall T_{i,j} \in \mathcal{T} \tag{3.5}$$

$$\sum_{j : T_{i,j} \in \mathcal{T}} \sum_{s \in \mathcal{V}} y_{i,j,s,t} \leq 1, \qquad \forall i \in \mathcal{P}, 0 \leq t \leq u \tag{3.6}$$

$$C_{i,j} \geq \frac{1}{2} \sum_{s \in \mathcal{V}} y_{i,j,s,0}|I_0| \left(\frac{1}{p_{i,j,s}} + 1 \right) + \sum_{t=1}^{u} \sum_{s \in \mathcal{V}} \left(\frac{y_{i,j,s,t}|I_t|}{p_{i,j,s}} \tau_t + \frac{1}{2} y_{i,j,s,t}|I_t| \right), \quad \forall T_{i,j} \in \mathcal{T} \tag{3.7}$$

$$C_j \geq C_{i,j}, \qquad \forall T_{i,j} \in \mathcal{T} \tag{3.8}$$

$$\sum_{T_{i,j} \in \mathcal{T}} \sum_{s \in \mathcal{V}} \sum_{t=0}^{u} y_{i,j,s,t}|I_t| s^{\beta} \leq E \tag{3.9}$$

$$\sum_{t=0}^{\ell} \sum_{s \in \mathcal{V}} \frac{y_{i,j,s,t}|I_t|}{p_{i,j,s}} \geq \sum_{t=0}^{\ell} \sum_{s \in \mathcal{V}} \frac{y_{i',j,s,t}|I_t|}{p_{i',j,s}}, \qquad \forall T_{i,j} \in \mathcal{M}, T_{i',j} \in \mathcal{R}, 0 \leq \ell \leq u \tag{3.10}$$

$$y_{i,j,s,t} = 0, \qquad \forall T_{i,j} \in \mathcal{T}, s \in \mathcal{V}, t : \tau_t < r_j \tag{3.11}$$

$$y_{i,j,s,t}, C_{i,j}, C_j \geq 0, \qquad \forall T_{i,j} \in \mathcal{T}, s \in \mathcal{V}, 0 \leq t \leq u$$

is executed with speed s. In other words, $y_{i,j,s,t}|I_t|$ is the time that task $T_{i,j}$ is executed within the interval I_t at speed s, or equivalently, $\frac{y_{i,j,s,t}|I_t|}{p_{i,j,s}}$ is the fraction of the task $T_{i,j}$ that is executed within I_t at speed s. Note that the number of $y_{i,j,s,t}$ variables is polynomial to the size of the instance, to $1/\epsilon$ and to $1/\delta$. Furthermore, for each task $T_{i,j} \in \mathcal{T}$, we introduce a variable $C_{i,j}$, which corresponds to the completion time of $T_{i,j}$. Finally, let C_j, $j \in \mathcal{J}$, be the variable that corresponds to the completion time of job j. (LP) is a linear programming relaxation of the DS-NRG-MapReduce problem.

Our objective is to minimize the sum of weighted completion times of all jobs. For each task $T_{i,j} \in \mathcal{T}$, the corresponding constraint (3.5) ensures that $T_{i,j}$ is entirely executed. Constraints (3.6) enforce that the total amount of processing time that is executed within an interval I_t cannot exceed its length. In [93], the authors proposed a lower bound for the completion time of a job. This lower bound can be adapted to our problem and for the completion time of a task $T_{i,j} \in \mathcal{T}$ leads to a corresponding constraint (3.7). Constraints (3.8) ensure that the completion time of each job is the maximum over the completion times of all its tasks. Constraint (3.9) ensures that the given energy budget is not exceeded. Note that the value s^{β} for each $s \in \mathcal{V}$ is a fixed number. Constraints (3.10) imply the precedence constraints between the Map and the Reduce tasks of the same job, as they enforce that

the fraction of a Map task that is executed up to each time point should be at least the fraction of a Reduce task of the same job executed up to the same time point; hence, each Map task completes before all Reduce tasks of the same job. Constraints (3.11) do not allow tasks of a job to be executed before their release date.

In what follows, we denote an optimal solution to (LP) by $(\bar{y}_{i,j,s,t}, \bar{C}_{i,j}, \bar{C}_j)$.

3.4.3 The algorithm

In this section we use (LP) to derive a feasible schedule for the NRG-MapReduce problem. Our algorithm is based on the idea of list scheduling in order of α-points [61, 86]. In general, an α-point of a job is the first point in time where an α-fraction of the job has been completed, where $\alpha \in (0, 1)$ is a constant that depends on the analysis. In this paper, we will define the α-point $t^{\alpha}_{i,j}$ of a task $T_{i,j} \in \mathcal{T}$ as the minimum ℓ, $0 \leq \ell \leq u$, such that at least an α-fraction of $v_{i,j}$ is accomplished up to the interval I_ℓ to (LP), i.e.,

$$t^{\alpha}_{i,j} = \min \left\{ \ell : \sum_{t=0}^{\ell} \sum_{s \in \mathcal{S}} \frac{\bar{y}_{i,j,s,t} |I_t|}{p_{i,j,s}} \geq \alpha \right\}.$$

Algorithm EMR(α, γ): an algorithm for the NRG-MapReduce problem.

1: Compute an optimal solution $(\bar{y}_{i,j,s,t}, \bar{C}_{i,j}, \bar{C}_j)$ to (LP).
2: **for** each task $T_{i,j} \in \mathcal{T}$ **do**
3: Compute the α-point $t^{\alpha}_{i,j}$, the processing time $p_{i,j}$ and the speed $s_{i,j}$.
4: **end for**
5: **for** each processor $i \in \mathcal{P}$ **do**
6: Compute the priority list σ_i.
7: **end for**
8: **for** each time where a processor $i \in \mathcal{P}$ becomes available **do**
9: Select the first available task, let $T_{i,j}$, in σ_i which has not been yet executed.
10: Schedule $T_{i,j}$, non-preemptively, with processing time $p_{i,j}$.
11: Let $C_{i,j}$ be the completion time of task $T_{i,j}$.
12: **end for**
13: **for** each job $j \in \mathcal{J}$ **do**
14: Compute its completion time $C_j = \max_{i \in \mathcal{P}} C_{i,j}$.
15: **end for**

Thus, once our algorithm has computed an optimal solution $(\bar{y}_{i,j,s,t}, \bar{C}_{i,j}, \bar{C}_j)$ to (LP), it calculates the corresponding α-point, $t^{\alpha}_{i,j}$, for each task $T_{i,j} \in \mathcal{T}$. Then we create a feasible schedule as follows: For each processor $i \in \mathcal{P}$, we consider a priority list σ_i of its tasks such that the tasks with smaller α-point have higher priority. A crucial point in our analysis is that we consider that a task $T_{i,j} \in \mathcal{T}$ becomes *available* for the algorithm after the time $\tau_{t^{\alpha}_{i,j}+1} > r_j$. Moreover, if $T_{i,j} \in \mathcal{R}$ then we need also all tasks $T_{i',j} \in \mathcal{M}$ to be completed in order $T_{i,j}$ to be considered as available. For each task $T_{i,j} \in \mathcal{T}$, we use a constant speed

$s_{i,j} = \frac{v_{i,j}}{p_{i,j}}$, where

$$p_{i,j} = \gamma \sum_{t=0}^{t_{i,j}^{\alpha}} \sum_{s \in \mathcal{V}} \bar{y}_{i,j,s,t} |I_t|$$

is the processing time of $T_{i,j}$ used by our algorithm, and $\gamma > 0$ is a constant that we define later and describes the trade-off between the energy consumption and the weighted completion time of jobs. In fact, speed $s_{i,j}$ is determined by the needs of the analysis and it serves as a tool in order to upper bound the energy augmentation used for the execution of $T_{i,j}$ and also the completion time of $T_{i,j}$ in a schedule produced by the algorithm. At each time point where a processor $i \in \mathcal{P}$ is available, our algorithm selects the highest priority available task in σ_i which has not been yet executed. Note that our algorithm always create a feasible solution as we do not insist on selecting the highest priority task if this is not available. Algorithm EMR(a, γ) gives a formal description of our method.

Note that the processing time of a task $T_{i,j} \in \mathcal{T}$ to an optimal solution to (LP) is $\bar{p}_{i,j} = \sum_{t=0}^{u} \sum_{s \in \mathcal{V}} \bar{y}_{i,j,s,t} |I_t|$. Hence, the energy consumption $\bar{E}_{i,j} = \sum_{t=0}^{u} \sum_{s \in \mathcal{V}} \bar{y}_{i,j,s,t} |I_t| s^{\beta}$ for the execution of $T_{i,j}$ to an optimal solution to (LP) may be smaller or bigger than the energy consumption $E_{i,j}$ for the execution of $T_{i,j}$ by the algorithm. In order to relate these two quantities we need the following technical lemma.

Lemma 3.2 *Let $s_1, s_2, \ldots, s_k$ and $q_1, q_2, \ldots, q_k$ be positive values and $\beta > 2$. Then, it holds that*

$$\left(\frac{1}{\sum_{i=1}^{k} q_i \frac{1}{s_i}} \right)^{\beta-1} \leq \frac{\sum_{i=1}^{k} q_i s_i^{\beta-1}}{\left(\sum_{i=1}^{k} q_i \right)^{\beta}}$$

Proof. The expression of the statement can be written equivalently as follows.

$$\left(\frac{\sum_{i=1}^{k} q_i}{\sum_{i=1}^{k} q_i \frac{1}{s_i}} \right)^{\beta-1} \leq \frac{\sum_{i=1}^{k} q_i s_i^{\beta-1}}{\sum_{i=1}^{k} q_i} \tag{3.12}$$

Note that the function $f(x) = x^{\beta-1}$ is convex for $\beta > 2$. Thus, by the Jensen's inequality we have that

$$f\left(\frac{\sum_{i=1}^{k} q_i s_i}{\sum_{i=1}^{k} q_i} \right) \leq \frac{\sum_{i=1}^{k} q_i f(s_i)}{\sum_{i=1}^{k} q_i}$$

which is translated as

$$\left(\frac{\sum_{i=1}^{k} q_i s_i}{\sum_{i=1}^{k} q_i} \right)^{\beta-1} \leq \frac{\sum_{i=1}^{k} q_i s_i^{\beta-1}}{\sum_{i=1}^{k} q_i}$$

Therefore, in order to show inequality (3.12), it suffices to show that

$$\left(\frac{\sum_{i=1}^{k} q_i}{\sum_{i=1}^{k} q_i \frac{1}{s_i}} \right)^{\beta-1} \leq \left(\frac{\sum_{i=1}^{k} q_i s_i}{\sum_{i=1}^{k} q_i} \right)^{\beta-1}$$

Thus, it suffices to prove that

$$\frac{\sum_{i=1}^{k} q_i}{\sum_{i=1}^{k} q_i \frac{1}{s_i}} \leq \frac{\sum_{i=1}^{k} q_i s_i}{\sum_{i=1}^{k} q_i}$$

An equivalent representation of the above expression is

$$\left(\sum_{i=1}^{k} q_i\right)^2 \leq \left(\sum_{i=1}^{k} q_i s_i\right)\left(\sum_{i=1}^{k} q_i \frac{1}{s_i}\right) \Leftrightarrow$$

$$\sum_{i=1}^{k} q_i^2 + \sum_{i,j=1,\, i\neq j}^{k} 2q_i q_j \leq \sum_{i=1}^{k} q_i^2 \sum_{i,j=1,\, i\neq j}^{k} q_i q_j \left(\frac{s_i}{s_j} + \frac{s_j}{s_i}\right)$$

The last inequality is always true, as

$$2 \leq \frac{s_i}{s_j} + \frac{s_j}{s_i} \Leftrightarrow 2 \leq \frac{s_i^2 + s_j^2}{s_i s_j} \Leftrightarrow 0 \leq (s_i - s_j)^2$$

and hence the lemma follows. ∎

The next lemma gives an upper bound on the energy augmentation used by Algorithm EMR(a, γ) for the execution of $T_{i,j}$.

Lemma 3.3 *Let $\bar{E}_{i,j}$ and $E_{i,j}$ be the energy consumption of the task $T_{i,j} \in \mathcal{T}$ in an optimal solution to (LP) and in the solution of Algorithm* EMR(a, γ)*, respectively. It holds that $E_{i,j} \leq \frac{1}{\gamma^{\beta-1}\alpha^{\beta}} \bar{E}_{i,j}$.*

Proof. By the definition of $E_{i,j}$ we have that

$$\begin{aligned} E_{i,j} &= v_{i,j} s_{i,j}^{\beta-1} = v_{i,j} \left(\frac{v_{i,j}}{\gamma p_{i,j}}\right)^{\beta-1} \\ &= \frac{v_{i,j}}{\gamma^{\beta-1}} \left(\frac{v_{i,j}}{\sum_{s\in\mathcal{V}} \sum_{t=0}^{t_{i,j}^{\alpha}} \bar{y}_{i,j,s,t} |I_t|}\right)^{\beta-1} \end{aligned}$$

Since for each speed $s \in \mathcal{V}$, $p_{i,j,s} = \frac{v_{i,j}}{s}$, the above equality can be written as

$$E_{i,j} = \frac{v_{i,j}}{\gamma^{\beta-1}} \left(\frac{1}{\sum_{s\in\mathcal{V}} \frac{1}{s} \sum_{t=0}^{t_{i,j}^{\alpha}} \frac{\bar{y}_{i,j,s,t}|I_t|}{p_{i,j,s}}}\right)^{\beta-1}$$

Hence, by using Lemma 3.2 we get

$$E_{i,j} \leq \frac{v_{i,j}}{\gamma^{\beta-1}} \frac{\sum_{s\in\mathcal{V}} s^{\beta-1} \sum_{t=0}^{t_{i,j}^{\alpha}} \frac{\bar{y}_{i,j,s,t}|I_t|}{p_{i,j,s}}}{\left(\sum_{s\in\mathcal{V}} \sum_{t=0}^{t_{i,j}^{\alpha}} \frac{\bar{y}_{i,j,s,t}|I_t|}{p_{i,j,s}}\right)^{\beta}}$$

By the definition of α-points we have that $\sum_{t=0}^{t_{i,j}^{\alpha}} \sum_{s\in\mathcal{V}} \frac{\bar{y}_{i,j,s,t}|I_t|}{p_{i,j,s}} \geq \alpha$, and thus

$$\begin{aligned}
E_{i,j} &\leq \frac{1}{\gamma^{\beta-1}\alpha^{\beta}} \sum_{s\in\mathcal{V}} s^{\beta-1} \sum_{t=0}^{t_{i,j}^{\alpha}} v_{i,j} \frac{\bar{y}_{i,j,s,t}|I_t|}{p_{i,j,s}} \\
&= \frac{1}{\gamma^{\beta-1}\alpha^{\beta}} \sum_{s\in\mathcal{V}} s^{\beta-1} \sum_{t=0}^{t_{i,j}^{\alpha}} v_{i,j} \frac{\bar{y}_{i,j,s,t}|I_t|}{v_{i,j}/s} \\
&= \frac{1}{\gamma^{\beta-1}\alpha^{\beta}} \sum_{s\in\mathcal{V}} \sum_{t=0}^{t_{i,j}^{\alpha}} \bar{y}_{i,j,s,t}|I_t|s^{\beta} \\
&\leq \frac{1}{\gamma^{\beta-1}\alpha^{\beta}} \sum_{s\in\mathcal{V}} \sum_{t=0}^{u} \bar{y}_{i,j,s,t}|I_t|s^{\beta} = \frac{1}{\gamma^{\beta-1}\alpha^{\beta}} \bar{E}_{i,j}
\end{aligned}$$

and the lemma follows. ∎

We also need to lower bound the completion time $\bar{C}_{i,j}$ of the task $T_{i,j} \in \mathcal{T}$ given by the (LP). This is done by the following lemma.

Lemma 3.4 *If $\lambda < \alpha\frac{v_{\min}}{s_{\max}}$, then for each task $T_{i,j} \in \mathcal{T}$ it holds that $\bar{C}_{i,j} \geq (1-\alpha)\cdot\tau_{t_{i,j}^{\alpha}}$.*

Proof. Recall that $t_{i,j}^{\alpha}$ corresponds to the interval $I_{t_{i,j}^{\alpha}} = (\tau_{t_{i,j}^{\alpha}}, \tau_{t_{i,j}^{\alpha}+1}]$. If we select $\lambda < \alpha\frac{v_{\min}}{s_{\max}}$, then there is no task with α-point to the interval I_0. Hence, we can consider that the α-point of each task $T_{i,j} \in \mathcal{T}$ corresponds to an interval of the form $(\lambda(1+\delta)^{t_{i,j}^{\alpha}-1}, \lambda(1+\delta)^{t_{i,j}^{\alpha}}]$.

Starting from constraint (3.7) we have that

$$\begin{aligned}
\bar{C}_{i,j} &\geq \frac{1}{2}\sum_{s\in\mathcal{V}} \bar{y}_{i,j,s,0}|I_0| \left(\frac{1}{p_{i,j,s}} + 1\right) \\
&\quad + \sum_{t=1}^{u} \sum_{s\in\mathcal{V}} \left(\frac{\bar{y}_{i,j,s,t}|I_t|}{p_{i,j,s}}\tau_t + \frac{1}{2}\bar{y}_{i,j,s,t}|I_t|\right) \\
&\geq \sum_{t=t_{i,j}^{\alpha}}^{u} \sum_{s\in\mathcal{V}} \left(\frac{\bar{y}_{i,j,s,t}|I_t|}{p_{i,j,s}}\tau_t + \frac{1}{2}\bar{y}_{i,j,s,t}|I_t|\right) \\
&\geq \sum_{t=t_{i,j}^{\alpha}}^{u} \sum_{s\in\mathcal{V}} \frac{\bar{y}_{i,j,s,t}|I_t|}{p_{i,j,s}}\tau_t \\
&\geq \tau_{t_{i,j}^{\alpha}} \sum_{t=t_{i,j}^{\alpha}}^{u} \sum_{s\in\mathcal{V}} \frac{\bar{y}_{i,j,s,t}|I_t|}{p_{i,j,s}} \geq (1-\alpha)\cdot\tau_{t_{i,j}^{\alpha}}
\end{aligned}$$

where the last inequality holds by constraint (3.5) and as by the definition of α-point we know that $\sum_{t=0}^{t_{i,j}^{\alpha}-1} \sum_{s\in\mathcal{V}} \frac{\bar{y}_{i,j,s,t}|I_t|}{p_{i,j,s}} < \alpha$. ∎

Using Lemmas 3.3 and 3.4 as well as Lemma 3.1, the following approximation ratio of Algorithm EMR($\boldsymbol{a}, \gamma$) can be proved.

Theorem 3.1 *Algorithm* EMR(a,γ) *is a* $\frac{1}{\gamma^{\beta-1}\alpha^{\beta}}$*-energy* $\frac{\gamma^2+3\gamma+1}{1-\alpha}(1+\varepsilon)$*-approximation algorithm for the* NRG-MapReduce *problem, where* $\gamma > 0$ *and* $\alpha, \varepsilon \in (0,1)$.

Proof. Consider the schedule $\mathcal{S}$ produced by Algorithm EMR(a,γ) and let $T_{i,j} \in \mathcal{M}$ be any Map task. Recall that σ_i is the priority list of processor i. Let $\sigma_i(j) \subseteq \sigma_i$ be the list including the tasks with priority higher than the priority of $T_{i,j}$ in σ_i, including $T_{i,j}$. Then, for $C_{i,j}$ it holds that

$$C_{i,j} \leq \tau_{t^{\alpha}_{i,j}+1} + \sum_{k \in \sigma_i(j)} p_{i,k} \tag{3.13}$$

as $T_{i,j}$ is always available after $\tau_{t^{\alpha}_{i,j}+1}$, as a Map task. For the total processing time of jobs in $\sigma_i(j)$ we have that

$$\begin{aligned}
\sum_{k \in \sigma_i(j)} p_{i,k} &= \gamma \sum_{k \in \sigma_i(j)} \sum_{t=0}^{t^{\alpha}_{i,k}} \sum_{s \in \mathcal{V}} \bar{y}_{i,k,s,t} |I_t| \\
&\leq \gamma \sum_{k \in \sigma_i(j)} \sum_{t=0}^{t^{\alpha}_{i,j}} \sum_{s \in \mathcal{V}} \bar{y}_{i,k,s,t} |I_t| \\
&\leq \gamma \sum_{k \in \sigma_i} \sum_{t=0}^{t^{\alpha}_{i,j}} \sum_{s \in \mathcal{V}} \bar{y}_{i,k,s,t} |I_t| \\
&= \gamma \sum_{t=0}^{t^{\alpha}_{i,j}} |I_t| \sum_{k \in \sigma_i} \sum_{s \in \mathcal{V}} \bar{y}_{i,k,s,t} \leq \gamma \sum_{t=0}^{t^{\alpha}_{i,j}} |I_t| = \gamma \tau_{t^{\alpha}_{i,j}+1}
\end{aligned}$$

where the last inequality holds by applying constraint (3.6) of the (LP). Thus, from inequality (3.13) we have

$$C_{i,j} \leq (\gamma+1)\tau_{t^{\alpha}_{i,j}+1} \tag{3.14}$$

for each Map task $T_{i,j} \in \mathcal{T}$.

Consider now a job $j \in \mathcal{J}$ and let $T_{i,j} \in \mathcal{R}$ be a task of j. Moreover, let $T_{i',j} \in \mathcal{M}$ be the Map task of j that completes last in $\mathcal{S}$, i.e., $C_{i',j} = \max\{C_{i,j} : T_{i,j} \in \mathcal{M}, i \in \mathcal{P}\}$. By definition, $T_{i,j}$ becomes available at time $t = \max\{\tau_{t^{\alpha}_{i,j}+1}, C_{i',j}\}$. Note that

$$t \leq \max\{\tau_{t^{\alpha}_{i,j}+1}, (\gamma+1)\tau_{t^{\alpha}_{i',j}+1}\} \leq (\gamma+1)\tau_{t^{\alpha}_{i,j}+1}$$

where the first inequality holds by inequality (3.14) and the second by the constraint (3.10) of (LP).

Let again $\sigma_i(j)$ be the list of tasks with higher priority than $T_{i,j}$ in σ_i, including $T_{i,j}$. If in the schedule $\mathcal{S}$ the processor i at time t executes a task $T_{i,j'} \notin \sigma_i(j)$, then for the completion time of $T_{i,j}$ it holds that

$$C_{i,j} \leq t + p_{i,j'} + \sum_{k \in \sigma_i(j)} p_{i,k} \tag{3.15}$$

because $T_{i,j}$ is available after time t and it has higher priority than any task $T_{i,j''} \notin \sigma_i(j)$. As before, we have that

$$\sum_{k \in \sigma_i(j)} p_{i,k} \leq \gamma \tau_{t^{\alpha}_{i,j}+1}$$

Moreover, for the processing time of $T_{i,j'}$ it holds that

$$p_{i,j'} = \gamma \sum_{t=0}^{t^{\alpha}_{i,j'}} \sum_{s \in \mathcal{V}} \bar{y}_{i,j',s,t} |I_t| \leq \gamma \tau_{t^{\alpha}_{i,j'}+1} < \gamma t$$

as $T_{i,j'}$ is available at time t. Then, by equation (3.15) we have

$$C_{i,j} \leq (\gamma + 1)t + \gamma \tau_{t^{\alpha}_{i,j}+1} \leq (\gamma^2 + 3\gamma + 1)\tau_{t^{\alpha}_{i,j}+1}$$

As $\tau_{t^{\alpha}_{i,j}+1} = (1+\delta)\tau_{t^{\alpha}_{i,j}}$, using Lemma 3.4 we get

$$C_{i,j} \leq \frac{\gamma^2 + 3\gamma + 1}{1 - \alpha}(1+\delta)\bar{C}_{i,j},$$

and by using constraint (3.8) of (LP)

$$C_{i,j} \leq \frac{\gamma^2 + 3\gamma + 1}{1 - \alpha}(1+\delta)\bar{C}_j$$

Since the above inequality holds for each processor $i \in \mathcal{P}$, it must also hold for $C_j = \max_{i \in \mathcal{P}}\{C_{i,j}\}$ and thus

$$C_j \leq \frac{\gamma^2 + 3\gamma + 1}{1 - \alpha}(1+\delta)\bar{C}_j$$

If we sum up all weighted completion times in $\mathcal{S}$ we yield

$$\sum_{j \in \mathcal{J}} w_j C_j \leq \frac{\gamma^2 + 3\gamma + 1}{1 - \alpha}(1+\delta) \sum_{j \in \mathcal{J}} w_j \bar{C}_j$$

and as $\sum_{j \in \mathcal{J}} w_j \bar{C}_j$ is a lower bound to the objective value of an optimal solution for the DS-NRG-MapReduce problem, the theorem follows. ∎

By choosing $\gamma = \frac{1}{\alpha^{\beta - \sqrt[1]{\alpha}}}$, no energy augmentation is used and Algorithm EMR(a, γ) becomes a constant-factor approximation for the NRG-MapReduce problem, and the following theorem holds.

Theorem 3.2 *There is a* $\frac{(\alpha^{\beta - \sqrt[1]{\alpha}})^2 + 3\alpha^{\beta - \sqrt[1]{\alpha}} + 1}{(\alpha^{\beta - \sqrt[1]{\alpha}})^2(1-\alpha)}(1+\varepsilon)$*-approximation algorithm for the* NRG-MapReduce *problem, where* $\alpha, \varepsilon \in (0,1)$.

In Figure 3.3 we depict the trade-off between energy augmentation and approximation ratio for some practical values of β.

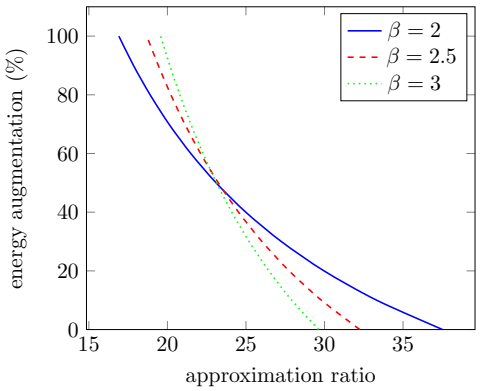


Figure 3.3: Trade-off between energy augmentation and approximation ratio when $\beta = \{2, 2.5, 3\}$.

For special instances of our problem where there are no precedence constraints between Map and Reduce tasks or even all jobs have a common release date (as in [38]) our results are improved as follows.

Corollary 3.1 *There is a* $\frac{\alpha^{\beta-\sqrt[\beta]{\alpha}+1}}{\alpha^{\beta-\sqrt[\beta]{\alpha}(1-\alpha)}}(1+\varepsilon)$*-approximation algorithm for the* NRG-MapReduce *problem without precedence constraints between Map and Reduce tasks, and a* $\frac{1}{\alpha^{\beta-\sqrt[\beta]{\alpha}(1-\alpha)}}(1+\varepsilon)$*-approximation algorithm for the* NRG-MapReduce *problem without precedence constraints between Map and Reduce tasks and jobs with common release dates, where* $\alpha, \varepsilon \in (0, 1)$.

Our ratios are optimized by selecting the appropriate value of α for each β. Table 3.1 gives the achieved ratios for practical values of β.

β	general	no precedence	no precedence & no release dates
2	37.52	9.44	6.75
2.2	34.89	8.84	6.29
2.4	33.01	8.41	5.97
2.6	31.59	8.09	5.72
2.8	30.50	7.84	5.53
3	29.62	7.64	5.38

Table 3.1: Approximation ratios for the NRG-MapReduce problem for different values of β.

3.5 Classical MapReduce scheduling

In this section, we turn our attention to classical MapReduce scheduling, where energy management is not a concern while processors run at (unit) constant speed and present constant approximation algorithms, which substantially generalize the results of [83] for

MapReduce scheduling on unrelated processors towards two directions, motivated by real MapReduce systems: (i) the jobs consist of multiple Map and Reduce tasks and (ii) the shuffle phase is incorporated into the scheduling process.

Unlike the previous model, now we consider a set $\mathcal{J}$ of n MapReduce jobs to be executed on a set $\mathcal{P}$ of m *unrelated processors*. Each job is associated with a positive weight and consists of a set of Map tasks $\mathcal{M}$ and a set of Reduce tasks $\mathcal{R}$, which are all *available at time zero*. Each task is denoted by $T_{k,j} \in \mathcal{M} \cup \mathcal{R}$, where $k \in N$ is the task index of job $j \in \mathcal{J}$ and is associated with a vector of non-negative *processing times* $\{p_{i,k,j}\}$, one for each processor $i \in \mathcal{P}_b$, where $b \in \{\mathcal{M}, \mathcal{R}\}$. Let $\mathcal{P}_{\mathcal{M}}$ and $\mathcal{P}_{\mathcal{R}}$ be the set of Map and the set of Reduce processors respectively. For convenience, we assume that $\mathcal{P}_{\mathcal{M}} \cap \mathcal{P}_{\mathcal{R}} = \emptyset$, however we are able to extend our results to the case where the two sets of processors are not necessarily disjoint (or even are identical). As before, each job has at least one Map and one Reduce task and every Reduce task cannot start its execution before the completion of all Map tasks of the same job.

For a given schedule we denote by C_j and $C_{k,j}$ the completion times of each job $j \in \mathcal{J}$ and each task $T_{k,j} \in \mathcal{M} \cup \mathcal{R}$ respectively. Note that, due to the precedence constraints between Map and Reduce tasks, $C_j = \max_{T_{k,j} \in \mathcal{R}}\{C_{k,j}\}$. Our goal is to schedule *non-preemptively* all Map tasks on processors of $\mathcal{P}_{\mathcal{M}}$ and all Reduce tasks on processors of $\mathcal{P}_{\mathcal{R}}$, with respect to their precedence constraints, so as to minimize the total weighted completion time of the schedule, i.e., $\sum_{j \in \mathcal{J}} w_j C_j$. We refer to this problem as Map-Reduce scheduling problem.

Concerning the complexity of the Map-Reduce scheduling problem, it generalizes the FFS problem which is is known to be strongly $\mathcal{NP}$-hard [50], even when there is a single Map and a single Reduce task that has to be assigned only to one Map and one Reduce processor respectively.

3.5.1 Map-Reduce scheduling problem

In this subsection, we present a 54-approximation algorithm for the Map-Reduce scheduling problem. Our algorithm is executed in the following two steps: (i) it computes a 27/2-approximate schedule for assigning and scheduling all Map tasks (resp. Reduce tasks) on processors of the set $\mathcal{P}_{\mathcal{M}}$ (resp. $\mathcal{P}_{\mathcal{R}}$) and (ii) it merges the two schedules in one, with respect to the precedence constraints between Map and Reduce tasks of each job. Step (ii) is performed by increasing the approximation ratio by a factor of 4.

Scheduling Map tasks and Reduce tasks. To schedule separately the Map and Reduce tasks on the processors $\mathcal{P}_{\mathcal{M}}$ and $\mathcal{P}_{\mathcal{R}}$, respectively, we start by formulating an interval-indexed LP-relaxation for the minimization of the total weighted completion time. Our LP-relaxation LP(b) is an adaptation to our problem of the standard LP-relaxation proposed by Hall et al. [60] for the problem of minimizing the total weighted completion time on unrelated processors.

For notational convenience, we use an argument $b \in \{\mathcal{M}, \mathcal{R}\}$ to refer either to Map or to Reduce sets of tasks. We define $(0, t_{\max} = \sum_{T_{k,j} \in b} \max_{i \in \mathcal{P}_b} p_{i,k,j}]$ to be the time horizon of potential completion times, where $t_{\max}$ is an upper bound on the makespan of a feasible schedule. We discretize the time horizon into intervals $[1,1], (1,(1+\delta)], ((1+\epsilon), (1+\delta)^2], \ldots, ((1+\delta)^{L-1}, (1+\delta)^L]$, where $\delta \in (0,1)$ is a small constant, and L is the smallest integer such that $(1+\delta)^{L-1} \geq t_{\max}$. Let $I_\ell = ((1+\delta)^{\ell-1}, (1+\delta)^\ell]$, for $1 \leq \ell \leq L$, and $\mathcal{L} = \{1, 2, \ldots, L\}$. Note that, interval $[1,1]$ implies that no job finishes its execution before time 1; in fact, we can assume, without loss of generality, that all processing times are positive integers. Note also that, the number of intervals is polynomial in the size of the instance and in $1/\delta$. For each processor $i \in \mathcal{P}_b$, task $T_{k,j} \in b$ and $\ell \in \mathcal{L}$, we introduce a variable $y_{i,k,j,\ell}$ that indicates if task $T_{k,j}$ is completed on processor i within the time interval I_ℓ. Furthermore, for each task $T_{k,j} \in T$, we introduce a variable $C_{k,j}$ corresponding to its completion time. For every job $j \in \mathcal{J}$, we also introduce a dummy task D_j with zero processing time on every processor, which has to be processed after the completion of every other task $T_{k,j} \in b$. Note that, the corresponding integer program is a $(1+\delta)$-relaxation of the original problem.

$$\textbf{LP(b)} : \text{minimize} \sum_{j \in \mathcal{J}} w_j C_{D_j}$$

subject to :

$$\sum_{i \in \mathcal{P}_b, \ell \in \mathcal{L}} y_{i,k,j,\ell} \geq 1, \qquad \forall T_{k,j} \in b \tag{3.16}$$

$$C_{D_j} \geq C_{k,j}, \qquad \forall j \in \mathcal{J}, T_{k,j} \in b \tag{3.17}$$

$$\sum_{i \in \mathcal{P}_b} \sum_{\ell \in \mathcal{L}} (1+\delta)^{\ell-1} y_{i,k,j,\ell} \leq C_{k,j}, \qquad \forall T_{k,j} \in b \tag{3.18}$$

$$\sum_{T_{k,j} \in b} p_{i,k,j} \sum_{t \leq \ell} y_{i,k,j,t} \leq (1+\delta)^\ell, \qquad \forall i \in \mathcal{P}_b, \ell \in \mathcal{L} \tag{3.19}$$

$$p_{i,k,j} > (1+\delta)^\ell \Rightarrow y_{i,k,j,\ell} = 0, \ \forall i \in \mathcal{P}_b, T_{k,j} \in b, \ell \in \mathcal{L} \tag{3.20}$$

$$y_{i,k,j,\ell} \geq 0, \qquad \forall i \in \mathcal{P}_b, T_{k,j} \in b, \ell \in \mathcal{L}$$

Our objective is to minimize the sum of weighted completion times of all jobs. Constraints (3.16) ensure that each task is completed on a processor of the set $\mathcal{P}_b$ in some time interval. Constraints (3.17) assure that for each job $j \in \mathcal{J}$, the completion of each task $T_{k,j}$ precedes the completion of task D_j. Constraints (3.18) impose a lower bound on the completion time of each task. For each $\ell \in \mathcal{L}$, constraints (3.19) and (3.20) are validity constraints which state that the total processing time of jobs that are executed up to an interval I_ℓ on a processor $i \in \mathcal{P}_b$ is at most $(1+\delta)^\ell$, and that if it takes time more than $(1+\delta)^\ell$ to process a task $T_{j,k}$ on a processor $i \in \mathcal{P}_b$, then $T_{k,j}$ should not be scheduled on i, respectively.

Our algorithm, called Algorithm TaskScheduling(b), starts from an optimal fractional solution $(\bar{y}_{i,k,j,\ell}, \bar{C}_{k,j}, \bar{C}_{D_j})$ to LP(b) and, working along the lines of Section 5 in [42], rounds it to an integral solution corresponding to a feasible 27/2-approximate schedule of the job set $\mathcal{J}$ on processors $\mathcal{P}_b$. The idea of Algorithm TaskScheduling(b) is to partition the set of tasks $T_{k,j}$ into classes $S(\ell) = \{\mathcal{T}_{k,j} \in b \mid (1+\delta)^{\ell-1} \leq a\bar{C}_{k,j} \leq (1+\delta)^{\ell}\}$, where $\ell \in \{1, \ldots, L\}$ and $a > 1$ is a parameter, according to their (fractional) completion time in the optimal solution of LP(b), and to use Theorem 2.1 in [96] for scheduling the tasks in each class $S(\ell)$ independently. In fact, Algorithm TaskScheduling(b) can be regarded as a generalization of the approximation algorithm of Section 4 in [60], where the objective is to minimize weighted completion time, but each job consists of a single task (see also the discussion of Section 5 in [42]).

More specifically, we first observe that by the definition of $S(\ell)$ and due to constraints (3.5) and (3.8), for each task $T_{k,j} \in S(\ell)$, $\sum_{i \in \mathcal{P}_b} \sum_{t \leq \ell} y_{i,k,j,t} \geq \frac{a-1}{a}$. Otherwise, it would be $\sum_{i \in \mathcal{P}_b} \sum_{t \geq \ell+1} y_{i,k,j,t} > \frac{1}{a}$, which implies $a\bar{C}_{k,j} > (1+\delta)^{\ell}$. Therefore, if we set $y^*_{i,j,k,t} = 0$, for all $t \geq \ell + 1$, and $y^*_{i,j,k,t} = \frac{a}{a-1}\bar{y}_{i,j,k,t}$, for all $t \leq \ell$, we obtain a solution $y^*_{i,j,k,t}$ that satisfies the constraints (3.5), (3.9), and (3.11) of LP(b), if the right-hand side of (3.9) is multiplied by $a/(a-1)$. Therefore, for each $\ell = 1, \ldots, L$, the tasks in $S(\ell)$ alone can be (fractionally) scheduled on processors $\mathcal{P}_b$ with makespan at most $\frac{a}{a-1}(1+\delta)^{\ell}$. Now, using Theorem 2.1 in [96], we obtain an integral schedule for the tasks in $S(\ell)$ alone with makespan at most $(\frac{a}{a-1}+1)(1+\delta)^{\ell}$. By the definition of $S(\ell)$, in this integral schedule, each task $T_{k,j} \in S(\ell)$ has a completion time of at most $a(\frac{a}{a-1}+1)(1+\delta)\bar{C}_{k,j}$. Therefore, if we take the union of these schedules, one after another, in increasing order of $\ell = 1, \ldots, L$, the completion time of each job j is at most $a(\frac{a}{a-1}+1+\frac{1}{\delta})(1+\delta)\bar{C}_{D_j}$. Choosing $a = 3/2$ and $\delta = 1/2$, we obtain that:

Theorem 3.3 *[42] Algorithm* TaskScheduling*(b) is a* 27/2*-approximation for scheduling a set of Map tasks (resp. Reduce tasks) on a set of unrelated processors $\mathcal{P}_{\mathcal{M}}$ (resp. $\mathcal{P}_{\mathcal{R}}$), in order to minimize their total weighted completion time.*

Merging task schedules. Let $\sigma_{\mathcal{M}}, \sigma_{\mathcal{R}}$ be two schedules computed by two runs of Algorithm TaskScheduling(b), for $b = \mathcal{M}$ and $b = \mathcal{R}$, respectively. Let also $C_j^{\sigma_{\mathcal{M}}} = \max_{T_{j,k} \in \mathcal{M}}\{C_{k,j}\}, C_j^{\sigma_{\mathcal{R}}} = \max_{T_{j,k} \in \mathcal{R}}\{C_{k,j}\}$ be the completion times of all the Map and all the Reduce tasks of a job $j \in \mathcal{J}$ within these schedules, respectively. Depending on these completion time values, we assign each job $j \in \mathcal{J}$ a *width* equal to $\omega_j = \max\{C_j^{\sigma_{\mathcal{M}}}, C_j^{\sigma_{\mathcal{R}}}\}$.

Algorithm MR computes a feasible schedule by processing, in each time instant where a processor $i \in \mathcal{P}_b$ becomes available, either the Map task, assigned to $i \in \mathcal{P}_{\mathcal{M}}$ in $\sigma_{\mathcal{M}}$, with the minimum width, or the available (w.r.t. its release time ω_j) Reduce task, assigned to $i \in \mathcal{P}_{\mathcal{R}}$ in $\sigma_{\mathcal{R}}$, with the minimum width.

Extending the analysis in [83], we are able to prove that:

Algorithm MR: an algorithm for the Map-Reduce scheduling problem.

1: Assign the tasks in $\mathcal{M} \cup \mathcal{R}$ on the same processors as in schedules $\sigma_{\mathcal{M}}$ and $\sigma_{\mathcal{R}}$ respectively.
2: **for** each job $j \in \mathcal{J}$ **do**
3: Fix $\omega_j = \max\{C_j^{\sigma_{\mathcal{M}}}, C_j^{\sigma_{\mathcal{R}}}\}$ to be the width job j
4: **end for**
5: **for** each time t where a processor $i \in \mathcal{P}$ becomes available **do**
6: **if** $i = \mathcal{P}_{\mathcal{M}}$ **then**
7: Among the unscheduled Map tasks in i, schedule task $T_{k,j} \in \mathcal{M}$ with the smallest ω_j, with processing time $p_{i,k,j}$.
8: **else**
9: Among the unscheduled Reduce tasks, which have $\omega_j > t$, schedule task $T_{k,j} \in \mathcal{R}$ with the smallest ω_j, with processing time $p_{i,k,j}$.
10: **end if**
11: Let $C_{k,j}$ be the completion time of task $T_{k,j}$.
12: **end for**
13: **for** each job $j \in \mathcal{J}$ **do**
14: Compute the completion time $C_j = \max_{T_{k,j} \in \mathcal{R}} C_{k,j}$.
15: **end for**

Theorem 3.4 *Algorithm* MR *is a* 54*-approximation for the Map-Reduce scheduling problem.*

Proof. First, we have to prove that the schedule computed by the Algorithm MR algorithm is a non-preemptive one. This is obvious for the Map tasks, while in the case of Reduce tasks the only way to have preemption is to have a task $T_{r_1,j}$ that is not scheduled by the time a task $T_{r_2,j}$ with higher width is executed. But this cannot happen because if $T_{r_2,j}$ has higher width, then it will be available after $T_{r_1,j}$ and our algorithm will schedule first $T_{r_1,j}$, thus, a contradiction. Therefore, by execution of Algorithm MR it is clear that all tasks are executed non-preemptively, while all Map tasks are scheduled only on the Map processors $\mathcal{P}_{\mathcal{M}}$ and all Reduce tasks only on the Reduce processors $\mathcal{P}_{\mathcal{R}}$.

Now, we have to prove that the resulting schedule respects the precedence constraints between Map and Reduce tasks. Therefore, we have to prove that a Map task with width ω_j finishes before time ω_j. This means that the corresponding Reduce tasks will be executed afterwards since their release time is ω_j. For the sake of contradiction we assume that there is a Map task $T_{m_1,j}$ with width ω_j finishing by time $t > \omega_j$. It is obvious that the schedule has no idle time and therefore in the time interval $[0, t]$ the processor i of task $T_{m_1,j}$ processes tasks with width at most ω_j. However, by definition of width this means that in schedule $\sigma_{\mathcal{M}}$ the processor i processes more than ω_j volume of work in less than ω_j time which gives us a contradiction.

Using the same argument as in the case of Map tasks, we can prove that the completion time of each Reduce task is upper bounded from $r + \omega_j$, where r is the release time of the task in σ. Moreover, as we note, $r \leq \omega_j$ and thus $C_j^{\sigma} \leq 2\omega_j = 2\max\{C_j^{\sigma_{\mathcal{M}}}, C_j^{\sigma_{\mathcal{R}}}\}$.

Now, let C_j^{OPT} be the completion time of job j in the overall optimal schedule and let $C_j^{\mathrm{OPT}_{\mathcal{M}}}, C_j^{\mathrm{OPT}_{\mathcal{R}}}$ be its completion time in the optimal schedules of only the Map or the Reduce tasks. Applying Theorem 3.3 and using the fact that $\sum_j C_j^{\mathrm{OPT}} \geq \sum_j C_j^{\mathrm{OPT}_{\mathcal{M}}}$ and $\sum_j C_j^{\mathrm{OPT}} \geq \sum_j C_j^{\mathrm{OPT}_{\mathcal{R}}}$, the theorem follows. ∎

Remark. If the two sets of processors, $\mathcal{P}_{\mathcal{M}}, \mathcal{P}_{\mathcal{R}}$, are not necessarily disjoint (or even if they coincide with each other), then by setting $\omega_j = C_j^{\sigma_{\mathcal{M}}} + C_j^{\sigma_{\mathcal{R}}}$ and applying a similar analysis, we can yield the same result as in Theorem 3.4.

3.5.2 Map-Shuffle-Reduce scheduling problem

In the Map-Reduce scheduling problem, the Reduce phase of each job can start executed once its Map phase is finished. However, in real systems there is a significant cost for the key-value pairs with the same key to be transmitted to the corresponding single Reduce task. In this subsection, we incorporate the data shuffle phase in our model. To this end, inspired by [40] we introduce a number of *Shuffle tasks* for each Map task that simulate this transmission of the key-value pairs from a Map to the corresponding Reduce tasks. In contrast to [40], where the assignment of Shuffle tasks to processors is fixed, we consider a flexible model and study two different variants. In the first variant, each Shuffle task is executed on the same processor with its corresponding Reduce task, while in the second one, we consider a different set of processors executing the Shuffle tasks. For both variants, we present $O(1)$-approximation algorithms.

Note that the number of different keys is in general greater than the number of the Reduce processors available, and in this case a Reduce task receives all key-value pairs of some different keys. Although not all Reduce tasks receive key-value pairs from each Map task, we may assume without loss of generality that this is the case by simply setting the transmission time of the corresponding Shuffle tasks equal to zero. We also assume that only a single key-value pair can be transferred to a Reduce processor at any time and moreover, the transmission process cannot be interrupted. Thus, since the key-value pairs allocated to the same Reduce task cannot be transmitted in parallel, we can assume that all key-value pairs from a Map task that have been assigned to the same Reduce task can be considered as a single Shuffle task. Hence, the number of Shuffle tasks per Map task equals the number of the Reduce tasks.

The following properties summarize the above discussion for the Map-Shuffle-Reduce scheduling problem:

Properties

(i) Each Shuffle task cannot start its execution before the completion of its corresponding Map task.

(ii) For every Map task of a job, there are as many Shuffle tasks as the job's Reduce tasks. Some of them may have zero processing time, indicating that no key-value pairs are trans-

mitted from the corresponding Map task to the corresponding Reduce task).
(iii) Each Shuffle task is executed non-preemptively.
(iv) Shuffle tasks that are transmitting to the same Reduce processor must not overlap with each other.

To present our results for the Map-Shuffle-Reduce scheduling problem we introduce some additional notation. For each Map task $T_{k,j} \in \mathcal{M}$ of a job $j \in \mathcal{J}$, we introduce a set of Shuffle tasks $T_{r,k,j}$, $1 \leq r \leq \tau_j = |\{T_{k,j} \in \mathcal{R}\}|$, where τ_j is the number of Reduce tasks of job j. We denote by $\mathcal{H}$ the set of Shuffle tasks; note that for each Map task of a job there is a bijection between its Shuffle tasks and the job's Reduce tasks. Each Shuffle task $T_{r,k,j} \in \mathcal{H}$ is associated with a *transfer time* $t_{r,k,j}$, which is independent of the processor assignment. In Figure 3.4(i) we depict a MapReduce job j, as formed after the introduction of the Shuffle tasks.

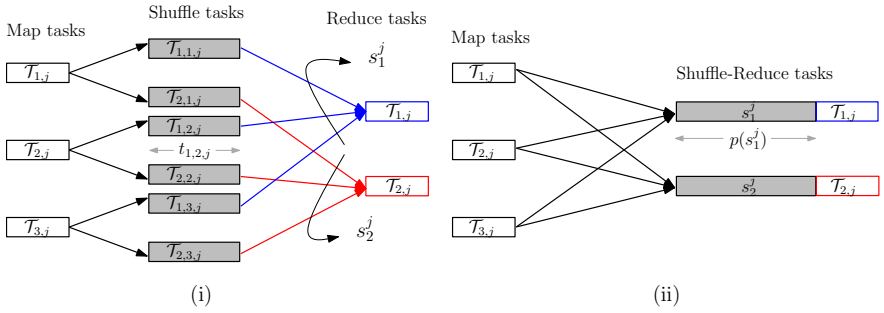


Figure 3.4: (i) Shuffle tasks and their precedence constraints with the Map tasks and Reduce tasks of a job j that comprises three Map tasks and two Reduce tasks and (ii) Precedence constraints among Map tasks and Shuffle-Reduce tasks.

The Shuffle tasks are executed on the Reduce processors. When the Shuffle tasks are executed on the same processors with its corresponding Reduce tasks, our algorithm proceeds into steps as for the Map-Reduce scheduling problem: a) It computes a 27/2-approximate schedule for the Map Tasks and a 27/2-approximate schedule for the Shuffle-Reduce tasks, with respect to the task Properties *(iii)-(iv)* and b) it merges the two schedules in a 54-approximate schedule for the Map-Shuffle-Reduce problem, with respect to the precedence between Map, Shuffle and Reduce tasks.

The key element of our algorithm is the integration of the Shuffle phase into the Reduce phase. In this direction, we consider a Reduce task $T_{r,j}$ of a job j and let $s_j^r = \{T_{r,k,j} \mid T_{k,j} \in \mathcal{M}\}$ be the set of Shuffle tasks that must complete before task $T_{r,j}$ starts its execution. As the tasks in s_j^r will be executed in the same processor as Reduce task $T_{r,j}$. Then, we are able to prove the following.

Lemma 3.5 *There is an optimal schedule of Shuffle tasks and Reduce tasks on processors*

of the set $\mathcal{P}_{\mathcal{R}}$ such that:
(i) There are no idle periods and
(ii) All Shuffle tasks in s_j^r are executed together and complete exactly before the Reduce task $T_{r,j}$ starts its execution.

Proof. (i) Consider a feasible schedule σ, then there are three cases in which an idle time can occur: either between the execution of two Shuffle tasks or two Reduce tasks or between a Shuffle and a Reduce task. Since all Shuffle tasks and Reduce tasks are assumed to be available from time zero and there are no precedence constraints among only Shuffle tasks or only Reduce tasks, skipping the idle times in the first two cases only decreases the objective value of σ. For the third case, it suffices to notice that since Shuffle tasks precede their corresponding Reduce tasks, by skipping the idles we decrease the completion time of the Reduce tasks and thus the objective value of σ. Hence, σ can be transformed into a schedule of less or equal total weighted completion time.
(ii) Again we consider a schedule σ that violates the claim and has the last Reduce task $T_{k,j}$ of a job j completed on some processor $i \in \mathcal{P}_{\mathcal{R}}$. If we fix the completion time of $T_{k,j}$ and shift all Shuffle tasks in s_j^r to execute just before $T_{k,j}$, consecutively and in arbitrary order, then, the completion time of j remains unchanged, while that of every task preceding $T_{k,j}$ in σ may decrease. Thus, after a finite number of shifts, σ can be transformed into a schedule of less or equal objective value. ∎

By Lemma 3.5 we are able to reformulate our input so as to incorporate the execution of Shuffle tasks of each job into the execution of its Reduce tasks. More specifically, for each Reduce task $T_{r,j}$ of a job j, for $1 \leq r \leq \tau_j$, we increase its processing time $p_{i,r,j}$, on each processor $i \in \mathcal{P}_{\mathcal{R}}$, by a quantity equal to the total processing time of the Shuffle tasks in s_j^r, i.e., $p(s_j^r) = \sum_{T_{r,k,j} \in s_j^r} t_{r,k,j}$. Let $p'_{i,r,j} = p_{i,r,j} + p(s_j^r)$ be the increased processing time for each task $T_{r,j} \in \mathcal{R}$ on processor $i \in \mathcal{P}_{\mathcal{R}}$, referred as Shuffle-Reduce task. Let $\mathcal{R}_{\mathcal{H}}$ be the new set of Shuffle-Reduce tasks. Then, by running Algorithm TaskScheduling($\mathcal{R}_{\mathcal{H}}$) and applying Theorem 3.3 we compute a 27/2-approximate schedule for scheduling the Shuffle-Reduce tasks of $\mathcal{R}_{\mathcal{H}}$. It is not difficult to prove that a schedule produced by TaskScheduling($\mathcal{R}_{\mathcal{H}}$), satisfies Properties *(iii)-(v)* and thus it is feasible for scheduling Shuffle-Reduce tasks.

In order to merge the two obtained schedules (the one for the Map tasks with the one for Shuffle-Reduce tasks) it suffices to consider the same precedence constraints, for Map tasks and Shuffle-Reduce tasks, as the ones among Map tasks and Reduce tasks (see Figure 3.4(ii)). The latter dependencies are clearly more general than the precedence constraints among Map tasks and Shuffle tasks of each job (each Shuffle task $T_{r,k,j}$ cannot start executing before the completion of Map task $T_{k,j}$) since, in order to start the execution of all Shuffle tasks in s_j^r, we have to wait for all Map tasks $T_{k,j}$ of job j to complete. However,

it satisfies Property *(i)*, and as we note, $C_j^{\mathrm{OPT}(b)}$ is a lower bound on C_j^{OPT} [2] for any kind of precedences among Map tasks and Shuffle-Reduce tasks. Thus, by running Algorithm MR we yield that:

Theorem 3.5 *Algorithm* MR *is a* 54*-approximation for the Map-Shuffle-Reduce scheduling problem.*

The Shuffle Tasks may be executed on different processors. When the Shuffle tasks are executed on different processors, we prove that we lose only a factor of 2 in the approximation ratio of the Shuffle-Reduce schedule. We assume that for any Reduce processor $i \in \mathcal{P}_{\mathcal{R}}$, there exits an *input* processor which receives data from the Map processors. Therefore, the input processor executes the Shuffle tasks that correspond to the Reduce tasks which have been assigned to i. We call the set of input processors $\mathcal{P}_{\mathcal{S}}$. Then, we can prove the following.

Lemma 3.6 *Consider two optimal schedules σ and σ' of Shuffle tasks and Reduce tasks on processors of the set $\mathcal{P}_{\mathcal{R}} \cup \mathcal{P}_{\mathcal{S}}$ and on processors of the set $\mathcal{P}_{\mathcal{R}}$ respectively. Let also $C_{k,j}^{\sigma}, C_{k,j}^{\sigma'}$ be the completion times of any Reduce task $T_{k,j}$ in σ and σ' respectively. Then, it holds that $C_{k,j}^{\sigma'} \leq 2C_{k,j}^{\sigma}$.*

Proof. We start with optimal schedule σ on the $\mathcal{P}_{\mathcal{R}} \cup \mathcal{P}_{\mathcal{S}}$ processors. We fix a Reduce processor i^r, the corresponding input processor i^s and a Reduce task $T_{k,j} \in \mathcal{R}$ of a job $j \in \mathcal{J}$. We build the schedule σ' on the i^r processor by executing the Reduce tasks in the same order as in σ and just before a Reduce task, we execute the corresponding Shuffle tasks. Let $B(k)$ be the set of Reduce tasks executed on processor i^r, before $T_{k,j}$ and $Sh(k)$ the set of the shuffle tasks that correspond to the Reduce tasks $B(k) \cup \{T_{k,j}\}$. Then, we have that

$$C_{k,j}^{\sigma'} = \sum_{T_{l,j} \in B(k)} p_{i^r,l,j} + \sum_{\substack{T_{q,l,j} \in Sh(k) \\ 1 \leq q \leq \tau_j}} t_{q,l,j},$$

which holds since in σ' there is no idle time, as already shown in Lemma 3.5. Moreover, since both $B(k)$ and $Sh(k)$ have to complete before $T_{k,j}$ in σ, we have that

$$C_{k,j}^{\sigma} \geq \max \left\{ \sum_{T_{l,j} \in B(k)} p_{i^r,l,j}, \sum_{\substack{T_{q,l,j} \in Sh(k) \\ 1 \leq q \leq \tau_j}} t_{q,l,j} \right\}$$

and therefore $C_{k,j}^{\sigma'} \leq 2C_{k,j}^{\sigma}$. ∎

2. Where C_j^{OPT} is the completion time of job j in the overall optimal schedule and $C_j^{\mathrm{OPT}(b)}$ the completion time in optimal schedules of either the Map tasks or the Shuffle-Reduce tasks separately.

Therefore, if we assume the existence of the $\mathcal{P}_{\mathcal{S}}$ processors then, combining Lemma 3.6 with Theorem 3.3 we yield a 27-approximation algorithm for scheduling the Shuffle-Reduce tasks.

Then, by running Algorithm MR in order to combine this schedule with the schedule of the Map tasks, using the same analysis as before, we get the next corollary. Note that the Shuffle tasks here form a special third stage in the FFS problem.

Corollary 3.2 *Algorithm* MR *is a* 81*-approximation for the Map-Shuffle-Reduce scheduling problem, when the Shuffle tasks run on different processors of the Reduce tasks.*

3.6 Concluding remarks

We presented a constant-factor approximation algorithm based on a linear programming formulation of the problem of scheduling a set of MapReduce jobs in order to minimize their total weighted completion time under a given budget of energy. Moreover, in the direction of exploring the efficiency of standard scheduling policies, we experimentally evaluated their performance, using a convex programming relaxation of the problem, when a prespecified order of jobs is given. It has to be noticed that our results can be applied also in the case where multiple Map or Reduce tasks of a job are executed on the same processor. Furthermore, they can also be simplified, to apply for an aggregated objective where the goal is to minimize a linear combination of energy plus weighted completion times.

We also presented constant-approximation algorithms, when energy is not our concern, for scheduling a set of MapReduce jobs on unrelated processors in order to minimize their total weighted completion time. These are the first constant-approximation algorithms for a general setting of the FFS problem while also, according to our knowledge, this is the most general theoretical model for MapReduce scheduling that have been studied so far.

For the energy-aware setting of MapReduce scheduling an intersting direction for future work concerns the online case of the problem. In fact, it can be proved that there is no an $O(1)$-competitive deterministic algorithm (see Theorem 13 in [26]). However, a possible way to overcome this is to consider energy augmentation, or to study the closely-related objective of a linear combination of the sum of weighted completion times of the jobs and of the total consumed energy.

As already mentioned, a special case of the MapReduce scheduling model proposed in [38] is the concurrent open-shop problem (see [80]). Actually, our $\frac{1}{\alpha^{\beta-\frac{1}{\sqrt[3]{\alpha}}}(1-\alpha)}(1+\varepsilon)$-approximation algorithm for the NRG-MapReduce problem, with common release dates and without precedence constraints between Map and Reduce tasks, where $\alpha, \varepsilon \in (0,1)$ (see in Corollary 3.1) applies also for the concurrent open-shop problem in the speed-scaling setting, for a given budget of energy. As shown in Table 3.1, for practical values of $\beta \in [2,3]$ this ratio ranges from 6.75 to 5.38. So, another direction for further study is to improve the latter ratio for the energy-aware concurrent open-shop problem; recall that

in the classical setting there is an efficient (primal-dual) 2-approximation algorithm [80]. An idea to this direction is to investigate the structure of an optimal schedule, by applying the KKT conditions in a convex programming formulation of the problem. As we noted, a number of useful properties can be deduced, by using the convex program proposed in Subection 3.3.1 (ignoring constraints (3.3)). However, the difficulty to find a good ratio is due to the fact that these properties depend on the jobs' order of execution.

In terms of classical MapReduce scheduling, a promising direction for future work is the online case of the MapReduce scheduling problem under resource (speed) augmentation, when preemption is allowed. As noticed in [83], even when preemption is allowed, resource augmentation is essential for a reasonable competitive ratio. An idea towards this goal is to extend the techniques used in Subsections 3.2 and 5.2 of Moseley et al. [83] for jobs having multiple Map and Reduce tasks.

Moreover, an obvious question that arises is whether our analysis can be improved to provide a better approximation ratio. An interesting idea is to try to improve the 27/2-approximation ratio of Algorithm TaskScheduling(b), by extending the analysis of Section 4 in [60] for jobs consisting of multiple tasks. Recall that the authors of [60] proposed a 16/3-approximation algorithm for the minimization of the total weighted completion time. Another question concerns the extension of our model in the case where the jobs have arbitrary release dates. However, it seems difficult to incorporate the release dates into our analysis, since the merging procedure becomes complicated.

Chapter 4

Temperature-aware scheduling for throughput maximization

We consider a set $J = \{1, 2, \ldots, n\}$ of n unit-length jobs to be scheduled on a single processor, each one having a heat contribution $h_j \in \mathbb{Q}^+$. All jobs are considered to be released at time zero and have a common deadline D. Jobs are executed in some time interval of the form $[t-1, t)$, which we call the *time slot* t, for some positive integer t.

Based on the thermal and cooling mechanism [41], described in Section 1.1, we assume that the processor's thermal behavior obeys the following rule: At time 0 its temperature is T_0; when a job j is executed in time slot t, the processor's temperature at time t is equal to $T_t = \frac{T_{t-1}+h_j}{2}$, where 2 is the processor's *cooling factor* and T_{t-1} its temperature at time $t-1$. The processor's temperature is not allowed to exceed a given thermal threshold, which we assume to be $\Theta = 1$ by normalization. Therefore, w.l.o.g. we assume that the heat contribution of each job belongs to the interval $[0, 2]$. This means that at some time slot t, we can schedule only jobs of heat contribution h such that $(T_{t-1} + h)/2 \leq 1$. Idle slots can be treated as executing jobs of heat contribution 0, that is, after an idle slot the temperature is divided by 2. For the sake of simplicity, we refer to a job of heat contribution $0 \leq h \leq 2$ as an *h-job*. Moreover we say that this job is *hot* if $h > 1$ and *cool* if $h \leq 1$.

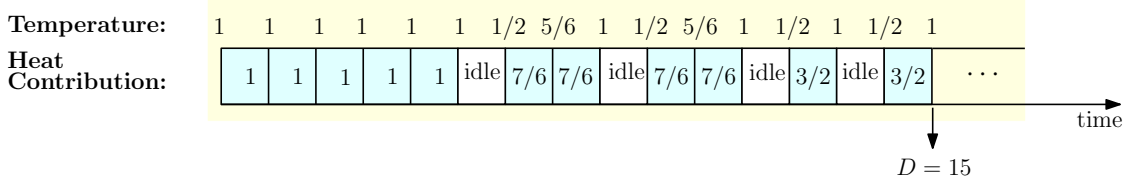


Figure 4.1: A schedule of unit-length jobs, with a common deadline $D = 15$ and heat contributions of the set $\{1, 7/6, 3/2\}$, to a processor for thermal and cooling management. The temperature at time zero as well as the thermal threshold are equal to 1. The throughput of the schedule is equal to 11.

Our goal is to maximize the throughput, i.e., the number of jobs that complete their execution before the common deadline D. The problem has been already proved to be strongly $\mathcal{NP}$-hard [41] and we focus on analyzing the approximation factor of a natural list scheduling algorithm, proposed for it, so called CoolestFirst: *at any time slot, if the current temperature is cool enough to allow a job to be scheduled, then it schedules the one with the smallest heat contribution — the coolest job — otherwise the processor remains idle in that slot.* A feasible schedule according to Algorithm CoolestFirst is presented in Figure 4.1, where five 1-jobs, four 7/6-jobs and two 3/2-jobs are executed before the common deadline $D = 15$, resulting to a throughput equal to 11.

4.1 Related work

The first theoretical approach that models the temperature and cooling management of processors was presented by Bansal et al. [25]. The authors proposed a model which is based on the Newton's law of cooling and uses speed scaling to decrease the processor's temperature, and they studied the preemptive case of the problem of minimizing the maximum temperature of a set of jobs with arbitrary deadlines and release dates, with respect to the deadline feasibility QoS criterion. They showed that the problem can be solved optimally in polynomial time, using the Ellipsoid algorithm and proposed a $e^{\beta}2^{\beta+1}(6(\frac{\beta}{\beta-1})^{\beta}+1)$-competitive algorithm for the online case of the problem, where $e \approx 2.718$ is the Euler's number. Recently, Atkins et al. [17] proposed an optimal algorithm of running time $O(n^2)$ and gave a $\frac{e}{e-1}(2+3e\beta^{\beta})$-competitive algorithm, which improves the one in [25] for some values of β (e.g., for $\beta = 3$).

As the above online strategies for minimizing the maximum temperature, compute a peak temperature that exceeds the optimal (offline) temperature by some constant factor, it seems more realistic to study temperature-aware scheduling for thermal and cooling management in terms of keeping the processors' temperature low by avoiding to exceed a thermal threshold (as its violation reduces the lifetime or even damages the processor). In this context, Chrobak et al. [41] proposed a model for thermal and cooling management (see in Section 1.1) for scheduling a set of unit-length jobs, with arbitrary release dates and deadlines, in order to maximize the throughput of the schedule. They proved that the problem is strongly $\mathcal{NP}$-hard, even when jobs are released at time zero and have common deadlines, by a reduction from Numerical-3D-Matching [51]. Moreover, they studied the online case of the problem and, based on a charging scheme which maps the jobs executed by the algorithm to the jobs executed by the adversary, they proved a 1/2-competitive ratio for a class of reasonable greedy list scheduling algorithms, like Coolest First and Earliest Deadline First, Moreover, the authors showed that no deterministic online algorithm achieves a better factor than 1/2.

Generalizations of the thermal and cooling management model [41] have been studied by

Birks et al. [31, 30, 33], considering different cooling factors and different processing times or objectives. More specifically, in [30] they extended the latter result for all possible values of the processor's cooling factor, proposing an optimal competitive ratio that increases as the cooling factor decreases, while it is constant for any fixed cooling factor. In [31] they studied the maximum weighted throughput objective and proved that there is no constant competitive deterministic online algorithm. Instead, they proposed an $O(\log W)$-competitive randomized online algorithm, where W is the ratio of maximum to minimum job weights. They also proposed a constant competitive algorithm, whose ratio is $O(\log 1/\epsilon)$, for $\epsilon > 0$, under $(1+\epsilon)$ augmentation of the thermal threshold. Moreover, they considered the problem in a multi-processor environment and gave a $O(mW^{1/m})$-competitive algorithm, where m is the number of processors, that matches their proposed lower bound. In [33] the authors proved upper and lower bounds on the competitive ratio of all deterministic online algorithms in the case where the jobs have equal, not necessarily unit, lengths. They studied both non-preemptive and preemptive versions and showed that in both cases the competitive ratio depends on the cooling factor and the common length of jobs. Especially for the CoolestFirst algorithm they showed that it gives a matching upper bound with the corresponding lower bound for all deterministic online algorithms.

Different QoS criteria have been also studied under the same model. Bampis et al. [19] proposed approximation results for a multi-processor environment under the thermal and cooling management model, for both the makespan minimization objective as well as the minimization of the maximum temperature when the threshold constraint is removed. For makespan minimization, they showed that it cannot be approximated within a factor less than $4/3 - \epsilon$, for $\epsilon \in (0, 1)$ and proposed a 2ρ-approximation algorithm, where ρ is the approximation factor of the classical makespan minimization problem on identical parallel processors [56, 57, 62]. For minimizing the maximum temperature, they proposed a 4/3-approximation algorithm which is tight. They also studied the problem of minimizing the average temperature and showed that it can be solved optimally in polynomial time. Finally, Birks and Fung [32] studied the minimization of the total flow time and proposed a 2.618-approximation algorithm in the case where all jobs are released at time zero while, for arbitrary release dates, they showed that it cannot be approximated within a factor less than $\Omega(n^{\frac{1}{2}-\epsilon})$, where $\epsilon > 0$.

4.2 Contribution

As aptly stated by Bansal et al. [25] *"If the processor in a mobile device exceeds its energy bound, then the battery is exhausted. If a processor exceeds its thermal threshold, it is destroyed."* Over the last decade there has been an increased interest concerning temperature-aware algorithmic models integrated into the scheduling theory and focused on the operating systems level. Actually, it seems important for the job scheduler at the

operating system level to keep the computational overhead for the scheduler low, since it could deteriorate the performance and generate additional heating. Therefore, we are particularly interested in natural algorithms, which at every time slot schedule the job with the highest priority among those available for execution. The priority could depend on the heat contribution of the job as well as on its deadline.

For this purpose, we focus on the natural goal of throughput maximization and we adopt the thermal and cooling management model [41]. Here, we are interested in the offline setting of the above problem and more specifically in the $\mathcal{NP}$-hard case, where all jobs are available at time zero and have a common deadline [41].

We analyze the approximation factor of CoolestFirst based on the following rounding procedure. First, we assume that in an optimal schedule, all the jobs are executed on-time, and the common deadline D coincides with the makespan of the optimal schedule. So, we simply ask how many jobs can be executed by CoolestFirst within time D. Then, we partition all jobs into classes according to their heat contributions and we round the heat contributions of each class so as to make it harder for the algorithm and more easy for the optimal schedule. The main advantage of this technique is that the rounded instances contain only a small number of different jobs, and permit to describe the optimum schedule. We apply this technique in two different ways and derive two lower bounds on the approximation factor of CoolestFirst.

For the first lower bound, in Section 4.4, we discretize the heat contribution scale in geometrically decreasing intervals and partition jobs into classes corresponding to intervals, according to their heat contributions. The rounding scheme is quite standard: round down the heat contributions of the jobs in each class so as to obtain an upper bound on OPT, then test how many jobs can be scheduled by CoolestFirst. This gives a lower bound equal to $\frac{k}{k+1}$, where k is the class of the last job executed by CoolestFirst.

For the second one, in Section 4.5, we manage to refine our partition with smaller intervals, so as to improve the previous ratio. The key for this refinement is to partition the jobs in terms of a new concept, called *density* of the schedule, i.e., the average number of jobs executed per unit-time slot. As we note, the schedule produced by CoolestFirst on a rounded instance can be partitioned in blocks according to the jobs' heat contributions. In each of these blocks the schedule consists only of jobs of some heat contribution — say h — and some idle slots. Each block has density equal to the proportion of non-idle slots among the time interval. Consider for instance the schedule of CoolestFirst in Figure 4.1 and in particular the block of 7/6-jobs (it starts with the first idle slot after the last 1-job), its density is equal to $\frac{2}{3}$. Clearly, for the analysis we are interested in density and its relation to the heat contribution. Our main contribution is a theorem stating (roughly-speaking) that for every density –say ρ – there is a heat contribution h_ρ such that CoolestFirst produces a schedule with density ρ when the instance consists only of jobs with heat contribution h_ρ. In addition, we show that the values h_ρ are increasing with ρ, as one would expect.

The proof of this theorem is presented in Section 4.5, following the analysis of the rounding procedure. For the rounding scheme, we consider a given set of densities which, by the above theorem, correspond to a set of heat contributions and thus, we are able to partition jobs into intervals of different densities, according to their heat contributions. The jobs are rounded in a similar (but little more rough) manner as in the first case: we round to lower density jobs for the algorithm and to higher density jobs for the optimal schedule. Now, the analysis becomes more subtle, using a linear programming formulation of the rounded instance. Actually, we formulate a linear program whose objective value corresponds to a lower bound on the approximation ratio of CoolestFirst. Then, by solving the dual of this linear programm we yield a lower bound on the approximation factor of at least 0.72. Finally, in Section 4.6, we propose ideas for improving our results, as well as some interesting open questions.

4.3 Preliminaries

As already mentioned, in our analysis we will partition the schedule produced by CoolestFirst into time intervals containing only jobs of identical heat contributions, scheduled as soon as it is admissible. Therefore, it is useful to define $G(h, T)$ as the schedule of jobs of heat contribution h with initial temperature $T_0 = T$. For notational convenience we describe the schedule $G(h, T)$ as a binary sequence $\omega = (w_0, w_1, \ldots, w_t) \in \{0, 1\}^\star$, $t \in \mathbb{N}$, where $w_t = 0$ if time slot t is idle and $w_t = 1$ otherwise (see Figure 4.2 for an example). The critical part of our analysis is based on the concept of *density* of a schedule $G(h, T)$, which is the proportion of 1's in the infinite sequence $G(h, T)$.

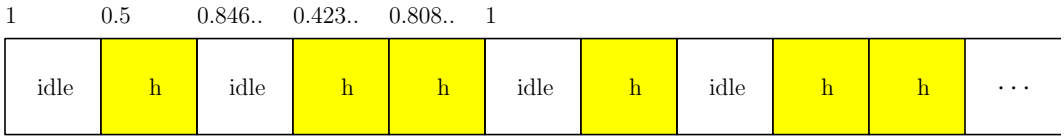


Figure 4.2: Example: a prefix of the infinite schedule generated by $h = 31/26$-jobs, obtaining $G(h) = (01011)^*$ for a density of $3/5$.

The following proposition analyses the sensibility of the optimal schedule to the initial temperature. According to it we can assume w.l.o.g. that $T_0 = 1$ and we write $G(h)$ as a shortcut for $G(h, 1)$.

Proposition 4.1 *For the optimum throughput OPT_T when the initial temperature is T, $0 \leq T < 1$, it holds that $OPT_1 \leq OPT_T \leq OPT_1 + 1$.*

Proof. First, we observe that any schedule which is feasible with some initial temperature, is also feasible for any cooler initial temperature. This implies the first inequality.
For the second inequality, let S be a schedule with throughput OPT_T. If S is also feasible

when the initial temperature is 1, then we have $\mathrm{OPT}_T = \mathrm{OPT}_1$. Otherwise there is a time t, where S schedules an h-job, which cannot be scheduled with initial temperature 1. Therefore $h > 1$. Let t be minimal and let S' be a schedule that is identical at all time points with S, except that it is idle at t. We claim that S' is feasible when the initial temperature is 1, which implies $\mathrm{OPT}_T \leq \mathrm{OPT}_1 + 1$.
By the choice of t, S' is feasible up to time t. Now by $h > 1$, S has a temperature greater than 0.5 at $t+1$. Since S' is idle at time t, it has a temperature not more than 0.5 at $t+1$. Our first observation in this proof applies again, implying that S' is feasible from time $t+1$ on as well. ∎

4.4 A first analysis

In this section we propose a rough lower bound on the approximation factor of the Algorithm CoolestFirst. This is done by performing a rounding procedure, based on a partition of all possible jobs' heat contribution values into intervals that are geometrically decreasing as the heat contribution approaches the hottest job of the instance.

For every $i \in \mathbb{N}$ we define the number $h^i := 2 - 2^{1-i}$ and let $\mathcal{H} := \{h^i : i \in \mathbb{N}\}$. Then, the hot jobs can be divided into classes, where the *i-th job class*, $i \geq 1$ consists of the interval $(h^i, h^{i+1}]$ (see Figure 4.3). Extending our definition, we call the $[0, 1]$ interval as the 0-th class, consisting of all cool jobs.

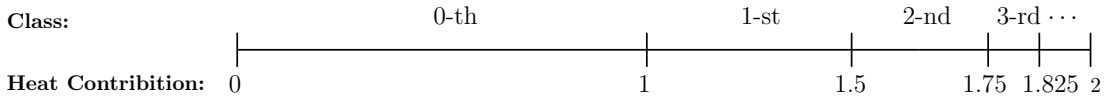


Figure 4.3: A partition of scheduled jobs into classes.

The next lemma describes an optimal schedule for instances with heat contributions from $\mathcal{H}$.

Lemma 4.1 *Let I be a set of jobs of heat contributions from $\mathcal{H}$. Then, the following steps produce an optimal schedule.*
(i) Run CoolestFirst *on all jobs from $\mathcal{H}\backslash\{h^0\}$.*
(ii) Schedule greedily the 0-jobs in the idle time slots left by the previous step.

Proof. We will prove this by an exchange argument. Consider an optimal schedule and let $t, t+1$ be two time slots such that the schedule is either idle or executes a 0-job at t and executes an h-job, with $h > 0$, at time $t+1$. Suppose that the temperature at time slot t is cool enough to execute h and then exchange the two time slots. This can lower the temperature at $t+1$ by $h/4$ and will preserve feasibility of the schedule. In such a schedule all h-jobs, having $h > 0$, are scheduled earliest possible, and in an arbitrary order.

In particular any h^i-job, for $i \geq 1$, is preceded by exactly $i-1$ time slots, being idle or scheduling a 0-job, and the temperature right after their execution is exactly 1.

Therefore, every scheduled h^i-jobs form a block of i consecutive time slots, and these blocks can be reordered freely, while preserving feasibility of the schedule. This completes the proof. ∎

The previous lemma permits an output sensitive analysis of the approximation factor of CoolestFirst.

Theorem 4.1 *Let k be the largest integer such that* CoolestFirst *schedules some hot job from the k-th class on some instance. Then, the approximation factor of* CoolestFirst *on this instance is at least $k/(k+1)$.*

Proof. Let S be the schedule produced by CoolestFirst on the instance. Let n_i be the number of jobs in S from the i-th class with $0 \leq i \leq k$. By this notation the throughput obtained by the algorithm is at most $\sum_{i=0}^{k} n_i$. Note that by definition of the algorithm all jobs not scheduled by CoolestFirst have heat contribution more than h^k.

In order to provide an upper bound for the optimal schedule, we round the jobs down. For $i = 0, \ldots, k$ every job from the i-th class is rounded down to h^i and all remaining jobs are rounded down to h^k. Since by replacing jobs by cooler jobs in a schedule, it preserves its feasibility, this does not decrease the optimal throughput.

What is the optimal throughput of the rounded instance? Since all jobs now belong to $\mathcal{H}$, we can apply Lemma 4.1. Therefore, the optimal schedule can be produced by first applying CoolestFirst on all hot jobs, resulting in a schedule S' in which later the n_0 0-jobs are filled. Then S' consists of two parts. The first part contains all the jobs scheduled by S and ranges over some interval $[0, v]$, while the second part consists of h^k-jobs, scheduled in the remaining interval $[v+1, D]$. In order to upper bound the jobs of the second part, we need to bound $D - v$.

The schedule S is partitioned into intervals of the form $[t_i, u_i]$ for every i: it is defined as the interval with minimal t_i and minimal u_i such that it contains exactly all the i-th class jobs scheduled in S and no other job. The last interval of this form might not end at time D, but then it is followed only by idle time slots.

Clearly, $u_i - t_i \leq n_i(i+1)$. The rounded version of these n_i jobs use at least $n_i i$ time slots in S' including the leading idle time slots. From this we deduce that

$$D - v \leq n_0 + \sum_{i=1}^{k-1} [n_i(i+1) - n_i i] + n_k(k+1)$$

Therefore, $D - v \leq \sum_{i=0}^{k} n_i + k n_k$. In the interval $[v+1, D]$ at most $(D-v)/k$ h^k-jobs

are scheduled in S'. Hence, the total number of jobs scheduled in S' is at most

$$\sum_{i=0}^{k} n_i + \frac{1}{k}\sum_{i=0}^{k} n_i = \frac{k+1}{k}\sum_{i=0}^{k} n_i.$$

This concludes the $\frac{k}{k+1}$-approximation factor of CoolestFirst. ∎

The above analysis is not tight. Consider the instance consisting of two 0-jobs and two 1.5-jobs with common deadline 4. The optimal schedule contains all four jobs alternating between their heat contributions, while CoolestFirst ends with an idle slot, and therefore has factor 3/4. Theorem 4.1 gives approximation factor 1/2, creating the need for a refined analysis.

4.5 A finer analysis

To refine the analysis of the previous section, we want to partition the heat contribution scale at heat contributions that are not necessarily from $\mathcal{H}$.

First, we note that the output of CoolestFirst for a rounded instance (as the one in Theorem 4.1) results in a schedule that can be expressed as the concatenation of prefixes of schedules $G(h)$, one for each different h value. In fact, it consists of blocks of jobs with the same heat contribution where every block has some density ρ_h. Recall that a schedule $G(h)$ is formulated as a binary sequence $(w_0, w_1, \ldots, w_{t-1})$, $w_t \in \{0,1\}^\star$, where a job is executed at time $t-1$ if $w_t = 1$, otherwise t is an idle time slot.

As mentioned in Section 4.3, an important measure of a schedule $G(h)$ is its density, representing the proportion of 1's in the infinite word $G(h)$. The following theorem provides a relation between the density of $G(h)$ and the heat contribution h.

Theorem 4.2 *For every $\rho \in \mathbb{Q} \cap [0,1]$ there is a heat contribution $h_\rho \in [0,2]$ such that the following property holds: For every integer ℓ, the ℓ-length prefix $\{w_0, w_1, \ldots, w_\ell\}$ of $G(h_\rho)$ satisfies*

$$\lfloor \ell \cdot \rho \rfloor \leq \sum_{t=1}^{\ell} w_t \leq \lceil \ell \cdot \rho \rceil.$$

Moreover for $\rho < \rho'$ we have $h_\rho > h_{\rho'}$.

Before actually proving this theorem, which is done in Section 4.5, we will show it can help to improve the analysis of the previous section.

Let $\mathcal{R} = \{\rho_0, \rho_1, \ldots, \rho_l\}$, where $\rho_i \in \mathbb{Q} \cap [0,1]$, $i = 0, 1, \ldots l$, be a set of a constant number of densities with $1 = \rho_0 > \rho_1 \ldots > \rho_l > 0$. These densities partition the interval $[0,1]$. By Theorem 4.2 the set $\mathcal{R}$ defines a sequence of heat contributions $1 = h_{\rho_0} < \ldots < h_{\rho_l}$ which partition the hot jobs further into the intervals $(h_{\rho_0}, h_{\rho_1}], \ldots, (h_{\rho_{l-1}}, h_{\rho_l}], (h_{\rho_l}, 2]$. Again we want to analyze the approximation factor of CoolestFirst in the case that the algorithm

schedules at least some job of heat contribution at least $h_{\rho_{l-1}}$. For an arbitrary instance, let $x_i,\ 0 \leq i \leq l$, be the number of jobs with heat contribution from the interval $(h_{\rho_{i-1}}, h_{\rho_i}]$.

We proceed in a similar manner as in the previous section, but we cannot simply round for every interval its jobs to its lower bound, because we do not know any good upper bound on the number of jobs in the optimal schedule. Instead for every $\rho_j,\ j = 1, 2, \ldots, l$ there is a rough upper bound, based in the following rounding. Every cool job is rounded to a 0-job, every hot job of heat contribution less or equal than $h_{\rho_{j-1}}$ is rounded to a 1-job, and all the remaining jobs are rounded to $h_{\rho_{j-1}}$-jobs. This permits us to apply the following lemma.

Lemma 4.2 *Consider an instance where all jobs have a heat contribution* $0, 1$ *or* h *and can all be completed before the deadline* D*. Then, there is an optimal schedule that is produced by the following steps.*
(i) Run CoolestFirst *on the* 1*- and* h*-jobs.*
(ii) Schedule greedily the 0*-jobs in the time slots left idle by the previous step.*

The proof uses the same exchange argument used to show Lemma 4.1 and is omitted.

Now, by using Lemma 4.2 for the rounded instance, we have the inequality

$$\sum_{i=1}^{j-1} x_i + \sum_{i=j}^{l} \frac{x_i}{\rho_{j-1}} \leqslant D, \qquad \forall j = 1, 2, \ldots, l-1, l. \tag{4.1}$$

With the previous statements in mind we can analyse the performance of CoolestFirst based on a rounding scheme using densities rather than heat contributions.

Theorem 4.3 *Fix an arbitrary positive integer constant* l*. Suppose that on some instance, the last job executed by* CoolestFirst *has heat contribution at least* h_ρ*, for some density* $\rho \geq (\sqrt{l} - 1)/(l - 1)$*. Then, the approximation factor of* CoolestFirst *is at least*

$$\frac{l-1}{l} - \frac{l-2}{l}\rho + \frac{l-1}{l}\rho^2,$$

up to an additive term of $2l\rho$*.*

Proof. Let I be an arbitrary instance. In order to lower bound the approximation factor of CoolestFirst we round the jobs to lower density jobs for the algorithm and to higher densitity jobs for the optimal schedule as described before. For this purpose we define the set of densities $\mathcal{R} = \{\rho_0, \rho_1, \ldots, \rho_l\}$, with

$$\rho_i := 1 - \frac{1-\rho}{l} i,\ i = 0, \ldots, l,$$

and we consider the linear programming formulation (P-LP).

$$
\begin{aligned}
\textbf{(P-LP)}\text{: minimize } & \sum_{i=0}^{l-1} x_i + (D-v)\,\rho \\
\text{subject to } & \sum_{i=0}^{l-1} x_i/\rho_i - v = 0 && \text{(a)}\\
& D - v \geqslant 0 && \text{(b)}\\
& D - \sum_{i=0}^{l} x_i \geqslant 0 && \text{(c)}\\
& D - \sum_{i=1}^{j-1} x_i - \sum_{i=j}^{l} \frac{x_i}{\rho_{j-1}} \geqslant 0 \qquad \forall j = 1, 2 \ldots, l && (y_j)\\
& \sum_{i=0}^{l} x_i = 1 && \text{(e)}\\
& x_0, \ldots, x_l, v, D \geq 0
\end{aligned}
$$

$$
\begin{aligned}
\textbf{(D-LP)}\text{: maximize } & e \\
\text{subject to } & e - c + a \leq 1 && (x_0)\\
& e - c + a/\rho_i - \sum_{j=1}^{i} y_j/\rho_{j-1} - \sum_{j=i+1}^{l} y_j \leq 1 \quad \forall i = 1, \ldots, l-1 && (x_i)\\
& e - c - \sum_{j=1}^{l} y_j/\rho_{l-1} \leq 0 && (x_l)\\
& b + c + \sum_{j=1}^{l} y_j \leq \rho && \text{(D)}\\
& b + a \geq \rho && \text{(v)}\\
& y_0, \ldots, y_l, b, c \geq 0,\ e, a \in \mathbb{R}
\end{aligned}
$$

The first part of the proof consists in showing that the optimum value of this linear program lower bounds the asymptotic approximation of CoolestFirst.

First, we can assume w.l.o.g. that the optimal schedule contains all jobs and only jobs not hotter than h_ρ, and in addition has makespan exactly the deadline. Let $\bar{D}$ be the deadline of instance I and $\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_l \in \mathbb{N}$ be the number of jobs in I belonging to each of the jobs' intervals defined by $\mathcal{R}$. Namely, $\bar{x}_0$ is the number of cool jobs, while $\bar{x}_i$ is the number of jobs belonging to $(h_{\rho_{i-1}}, h_{\rho_i}]$ for $i = 1, \ldots, l$. Now for the CoolestFirst we round the heat contribution of each job to the higher value of the interval it belongs to.

We call $\bar{v}$ the last time the algorithm schedules some job from $[0, h_{\rho_{l-1}}]$ in this rounded instance. Then, by Theorem 4.2 we have the equality

$$\bar{v} = \bar{x}_0 + \left\lceil \sum_{i=1}^{l-1} \frac{\bar{x}_i}{\rho_i} \right\rceil .$$

and the number of jobs schedule by CoolestFirst are

$$\sum_{i=0}^{l-1} \bar{x}_i + \left\lfloor (\bar{D} - \bar{v})\rho \right\rfloor . \tag{4.2}$$

Clearly $\bar{v} \leq \bar{D}$. Also, since we assumed that the optimum schedule contains all jobs, we have

$$\sum_{i=0}^{l} \bar{x}_i \leq \bar{D}.$$

The next step in our proof is to round the jobs for the optimum schedule. For every $j = 1, \ldots, l$ we use the rounding mentioned earlier. So, by Lemma 4.2 we have

$$\sum_{i=1}^{j-1} \bar{x}_i + \left\lceil \sum_{i=j}^{l} \frac{\bar{x}_i}{\rho_{j-1}} \right\rceil \leqslant \bar{D}.$$

Now, the approximation factor of CoolestFirst is upper bounded by the factor between (4.2) and the sum $\sum \bar{x}_i$. Note that, by removing the integer roundings in the (in)equalities above could result in a decrease of at most $2l$ of the difference $D - v$. This means that the expression (4.2) would be decreased by at most $2l\rho$.

The last step in our proof consists in relaxing the integrality constraint of $\bar{x}_0, \ldots, \bar{x}_l, \bar{v}, \bar{D}$, and normalizing the sum $\sum_{i=0}^{l} x_i$ to 1. So let $x_0, \ldots, x_l, v, D$, be the result of dividing the above numbers respectively by $\sum_{i=0}^{l} \bar{x}_i$. Clearly, all the inequalities on the linear program are satisfied by these values, and the objective value lower bounds the approximation factor of CoolestFirst. This concludes the first part of our proof.

It remains to lower bound the objective value of this linear program. This will be done by providing a specific solution to the dual linear program, as described by (D-LP).

It is not difficult to verify that the following values provide a solution to (D-LP), in particular the lower bound on ρ of the statement ensures that $c \geq 0$.

$$
\begin{aligned}
a &= \rho \\
b &= 0 \\
c &= e + \rho - 1 \\
y_j &= 0 && \forall j = 1, \ldots, l-1 \\
y_l &= 1 - e \\
e &= \frac{l-1}{l} - \frac{l-2}{l}\rho + \frac{l-1}{l}\rho^2.
\end{aligned}
$$

This completes the proof of the theorem. ∎

By using the first derivative of e in ρ, we can show that minimum is obtained at

$$\rho = \frac{l-2}{2l-2}$$

and has value

$$e_{\min} = \frac{3l-4}{4l-4}.$$

For example, for $l = 10$ this would show a lower bound on asymptotic approximation factor of CoolestFirst of $0.722\ldots$ while, for large values of l, the ratio goes to 0.75. However, we cannot use the limit of $e_{\min}$, when l tends to $+\infty$, in order to provide a bound on the asymptotic approximation factor because the additive constant $(2\rho l)$ is increasing with l.

4.6 Discrete lines

In this section we investigate the relation between the density and the heat contribution of a set of h-jobs, aiming to provide a detailed proof of Theorem 4.2.

The first of the following two procedures, called CoolestFirst(h) for the sake of uniformity, produces a binary sequence for a given value of h. In fact it produces only the part of the sequence that spans between two consecutive temperatures equal to one. For notational convenience, we force CoolestFirst(h) to return the digits of ω in reverse order i.e., $(w_{t-1}, w_{t-2}, \ldots, w_0)$. However, a sequence ω produced by the CoolestFirst(h), seems to be very similar with a sequence that corresponds to the discretization of a line with rational slope and zero offset (see Figure 4.4).

The procedure Staircase(p, q), shown below, produces the reverse of such a sequence for a slope equal to $\frac{p}{q-p}$, where p, q are considered to be co-prime integers.

Suppose now that q equals the length of ω, i.e., $T_{q \cdot k} = 1$, $k \in \mathbb{N}$. Then, the density of the schedule produced will be equal to $\rho_h = \sum_t w_t/q$. Let also $int(\omega) = \sum_{t=0}^{q-1} w_t 2^t$ be the decimal expansion of ω.

The following proposition establishes a very interesting (monotone) relation between

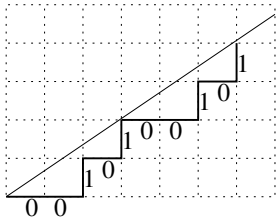


Figure 4.4: A discrete line with (slope, offset)= $(\frac{2}{3}, 0)$

Algorithm Two procedures that produce a binary sequence.

```
1: CoolestFirst(h)
2: T = 1;
3: t = 0;
4: repeat
5:     if (T + h)/2 > 1 then
6:         w_t = 0;
7:         T = T/2
8:     else
9:         w_t = 1;
10:        T = (T + h)/2
11:    end if
12:    t = t + 1
13: until T = 1
14: return ω = (w_{t−1}, w_{t−2}, ..., w_0)
```

```
1: Staircase(p, q)
2: y = 0;
3: t = 0;
4: repeat
5:     if y < p/(q − p) then
6:         a_t = 1;
7:         y = y + 1
8:     else
9:         a_t = 0;
10:        y = y − p/(q − p)
11:    end if
12:    t = t + 1
13: until y = 0
14: return α = (a_0, a_1, ..., a_{t−1})
```

the heat contribution h, the sequence ω and the density ρ_h of a schedule.

Proposition 4.2 *For a heat contribution h it holds that $h = \frac{2^q-1}{int(\omega)}$.*

Proof. During the execution of CoolestFirst(h), the digits of the sequence ω are produced in the order $w_0, w_1, \ldots, w_{q-1}$ and span between temperatures $T_0 = 1$ and $T_q = 1$. It holds that $T_q = \frac{T_0 + h\sum_{t=0}^{q-1} w_t 2^t}{2^q}$ and hence, $h = \frac{2^q-1}{int(\omega)}$. ∎

We shall prove that for a given density $\frac{p}{q}$ the procedure CollestFirst(h), for a heat contribution $h = \frac{2^q-1}{int(\alpha)}$, produces the same sequence with Staircase(p, q), i.e., $\omega = \alpha$. The following proposition summarizes the properties of the sequence α.

Proposition 4.3 *The sequence α produced by* Staircase(p, q) *starts with $a_0 = 1$, contains exactly p ones, has length equal to q, i.e., $\alpha = (a_0, a_1, \ldots, a_{q-1})$, finishes with $a_{q-1} = 0$ and it is non-periodic.*

Proof. Algorithm Staircase(p, q) starts with $y = 0$ and, hence, $a_0 = 1$. During its execution the value of the variable y is bounded by $0 \le y < 1 + \frac{p}{q-p}$, that is $0 \le \frac{y\cdot(q-p)}{q} < 1$. Moreover, each step $t = 0, 1, 2, ...$ of the procedure starts with $y = \ell - \frac{(t-\ell)\cdot p}{q-p} = \frac{\ell\cdot q - t\cdot p}{q-p}$, where

ℓ is the number of ones produced so far. Hence, when the t starts we have $\ell = \frac{p \cdot t}{q} + \frac{y \cdot (q-p)}{q}$ and since ℓ is an integer and $\frac{y \cdot (q-p)}{q} < 1$, it holds that $\ell = \left\lceil \frac{p \cdot t}{q} \right\rceil + \left\lfloor \frac{y \cdot (q-p)}{q} \right\rfloor = \left\lceil \frac{p \cdot t}{q} \right\rceil$.

When step $(q-1)$ starts we have $\ell = \left\lceil \frac{p \cdot (q-1)}{q} \right\rceil = p$. Hence, this step starts with $y = \frac{p \cdot q - (q-1) \cdot p}{q-p} = \frac{p}{q-p}$ and the procedure sets $a_{q-1} = 0$, reduces y to 0 and stops after having executed q steps. Therefore, α contains exactly p ones, has length equal to q, i.e., $\alpha = (a_0, a_1, \ldots, a_{q-1})$, and finishes with $a_{q-1} = 0$.

As p, q are co-primes, there is no integer $k > 1$ such that q can be divided to k groups (periods), each one having $\frac{p}{k}$ ones and $\frac{q-p}{k}$ zeros. Therefore, α is non-periodic. ∎

We fix now $\alpha^k = (a_0^k, a_1^k \ldots, a_{q-1}^k),\ 0 \leq k \leq q-1$, to be the k-th left circular shift of α, with $\alpha^0 = \alpha$. The next proposition is derived by the definition of the circular shifts and the non-periodicity of α.

Proposition 4.4 *For two circular shifts, $\alpha^k, \alpha^{k'}$, $k \neq k' \bmod q$ of α, it holds that*
(i) $a_t^k = a^{k'}_{(t+k-k') \bmod q}$, $t = 0, 1, \ldots, q-1$.
(ii) $\alpha^k \neq \alpha^{k'}$.

Proof. (i) Follows directly from the definition of the k-th left circular shift.
(ii) Assume, by contradiction, that there exist $k, k' \leq q-1$, where $k \neq k'$, such that $\alpha^k = \alpha^{k'}$, i.e., $w_i^k = w_i^{k'}$, for each $i,\ 0 \leq i \leq q-1$. From (i), it holds that $w_i^k = w^{k'}_{(i+k-k') \bmod q}$, and thus $w_i^{k'} = w^{k'}_{(i+k-k') \bmod q}$. This implies that α has a period equal to $|k-k'| < q$, a contraction, by Proposition 4.3. ∎

Let us denote the lexicographic relation between two binary sequences by $\succ$. Let also $1\alpha^k = (1, a_0^k, a_1^k \ldots, a_{q-1}^k)$ and $\alpha 1 = (a_0, a_1, \ldots, a_{q-1}, 1)$.

The following proposition gives two useful relations between the output α of the Algorithm StairCase(p, q) and its circular shifts.

Proposition 4.5 *For each $k,\ 1 \leq k \leq q-1$, it holds that*
(i) $\alpha \succ \alpha^k$.
(ii) If $a_0^k = 0$, then $1\alpha^k \succ \alpha 1$.

Proof. Let $y_k,\ 0 < k \leq q-1$, be the intermediate values of y at the beginning of the step k of Algorithm StairCase(p, q). First, we consider the procedure StairCase(p, q) initiated not by $y = 0$, but by one of those intermediate values, say $y = y_k$. Then, if the procedure is allowed to iterate until y becomes again y_k, it will produce again a sequence of length q. In fact, this sequence will be the k-th circular shift of α, as the procedure in the first k steps, will produce the last $q-k-1$ digits of α and in the next $k+1$ steps the first $k+1$ digits of α. Next, we claim that if $y_k < y_{k'},\ k \neq k'$, then $\alpha^k \succ \alpha^{k'}$. To see this assume, by contradiction, that $y_k < y_{k'}$ and $\alpha^k \preceq \alpha^{k'}$ and let u be the minimum index, such that $a_u^k \neq a_u^{k'}$ and $a_u^{k'} = 1$. As $a_i^k = a_i^{k'}, 0 \leq i \leq u-1$ it follows that the difference of

the y values, after those first $u-1$ steps, of the two runs of the procedure initiated with y_k and $y_{k'}$, is equal to $y_k - y_{k'}$. At step u, the procedure produces a $a_u^k = 0$ (for y_k) and $a_u^{k'} = 1$ (for $y_{k'}$). Hence, this step starts with $y \geq \frac{p}{q-p}$ and $y' < \frac{p}{q-p}$, respectively. However, $y - y' = y_k - y_{k'} < 0$, a contradiction.

For the point (i) of the proposition just observe that α and α^k are produced by two runs of the procedure initiated by $y_0 = 0$ and $y_k > 0$, respectively.

For the point (ii), observe first that $a_0 = 1$ (by Proposition 4.3) and $a_{q-1}^1 = 1$ (by Proposition 4.4). Therefore, $1\alpha^1 = \alpha 1$. Thus, it suffices to prove that if $a_{q-1}^k = 0$, then $\alpha^k \succ \alpha^1$ for each $k,\ 2 \leq k \leq q-1$. To produce α^1 and α^k the procedure starts with y_1 and y_k respectively. As $y_0 = 0$, we have that $y_1 = y_0 + 1 = 1$. As $a_{q-1}^k = 0$, it follows that $y_k = y_{k-1} - \frac{p}{q-p}$ and since $y_{k-1} < 1 + \frac{p}{q-p}$ (recall that this inequality holds for all values of y_k) we get $y_k < 1$. Therefore, by the claim above the relation in point (ii) holds. ∎

The following lemma together with Proposition 4.3 provides a proof for Theorem 4.2.

Lemma 4.3 *For a given density $\frac{p}{q}$ the binary sequences, α produced by* StairCase(p,q) *and ω produced by* CoolestFirst(h)*, with $h = \frac{2^q-1}{int(\alpha)}$, are equal.*

Proof. Let $T_t,\ 0 \leq t \leq q-1$, be the temperature in the beginning of each execution step of CoolestFirst(h). Recall that $T_0 = 1$ and by CoolestFirst it follows that $T_t \in [1-\frac{h}{2}, 1]$.

In order to produce a digit w_t, the procedure examines whether $T_t + h > 2$. By setting $h = \frac{2^q-1}{int(\alpha)}$, the latter inequality can be written as

$$T_t \cdot int(\alpha) + 2^q > 2int(\alpha) + 1. \tag{4.3}$$

As each T_t is calculated by a division by 2, the quantity $T_t \cdot int(\alpha)$ corresponds to the decimal expansion of the left circular shift $\alpha^{(q-t) \bmod q}$, $t = 0, \ldots, q-1$. Let $k = (q-t) \bmod q$. By converting (4.3) to its binary equivalent, we yield that

$$1\alpha^k \succ \alpha 1. \tag{4.4}$$

and by Lemma 4.5, if $a_{q-1}^k = 0$ then CoolestFirst(h) produces $w_t = 0$, otherwise it produces 1. Hence, at each step of the procedure we have that $w_t = a_{q-1}^k$. By applying Proposition 4.4, we yield that $a_{q-1}^k = a_{(q-1+q-t) \bmod q} = a_{q-t-1}$. ∎

4.7 Concluding remarks

We proposed two different rounding procedures to lower bound the approximation factor of CoolestFirst algorithm and proved that it is between 0.72 and 0.75. Our main contribution is not the rounding procedure itself, which is rather standard, but the technical lemmas behind it.

The main question that remains open is whether our throughput maximization problem accepts a PTAS or not. However, for further improvements on the approximation ratio, a useful note is that it seems better to use cool jobs in order to fill idle times between hot jobs, instead of processing them in the beginning of the schedule. Indeed, this suggests an algorithm that gives higher priority to hot jobs over cool jobs. However, we don't have the right tools to analyze this new algorithm.

Another direction is to investigate different discretizations for the rounding scheme while also consider different cooling factors. Furthermore, it is of special interest to deal with the multi-processor case when we are allowed to shift jobs from hotter to cooler processors (i.e., to allow migration).

Finally, recall that in the thermal and cooling management model [41], the processor runs at constant (unit) speed, so that the scheduler is allowed to leave an idle time slot whenever any of the jobs (released but not yet scheduled) violate the given thermal threshold. So, a more realistic direction for future work is to study the case where several different speed levels are available for the scheduler, which, instead of being idle before some possible violation of the thermal threshold, it can reduce its speed to an appropriate speed level.

Chapter 5

Conclusions

In this thesis we proposed approximation and online algorithms for scheduling problems that are motivated by aspects of energy and temperature management in computing environments and large-scale data processing models. The methodology followed was strongly supported by analytical tools, including formal modeling of discrete and optimization problems, mathematical programming techniques, established combinatorial arguments and computational complexity reductions. In the following we give an overview of some interesting future directions, based on the research conducted under this thesis.

Speed Scaling scheduling. In their seminal paper, Yao et al. [111] proposed an elegant optimal greedy algorithm for the single processor preemptive problem of energy minimization. On the other hand the non-preemptive case of this problem was proven to be $\mathcal{NP}$-hard [15] while, the authors proposed a $2^{5\beta-1}$-approximation algorithm. Since then, interesting LP-based approximation algorithms have been proposed, improving the latter ratio: (i) to $(1+\epsilon)^{\beta}\tilde{B}_{\beta}$-approximation [22], for $\epsilon \in (0,1)$, where $\tilde{B}_{\beta}$ a generalization of the Bell number that is also valid for fractional values of β, and (ii) to $(12(1+\epsilon))^{\beta-1}$-approximation [1], for $\epsilon \in (0,1)$. In fact, the first one gives better results for any $\beta < 77$, and thus, for all practical values of $\beta(1,3]$. Therefore, an intriguing algorithmic question that remains open is the design of a polynomial time approximation scheme (PTAS) for this problem.

Another interesting question deals with the single processor scheduling problem for minimizing the total weighted completion time of a set of jobs, where each job has an arbitrary release date and a budget of energy is given as input. In a recent paper by Megow and Verschae [82], a PTAS was proposed in the case where all jobs are released at time zero. They also gave a $(2+\epsilon)$-approximation, for $\epsilon \in (0,1)$, for arbitrary release dates when preemption is allowed. Thus, it is of great interest to ask for a PTAS in the latter case, either when preemption is allowed or not.

Power-down with Speed Scaling. Speed scaling is one of the main technologies used for saving energy. As mentioned in Section 1.1, another common technique is power-down, which involves switching devices into sleep, idle, and/or off states when not needed, introducing some delay issues and energy cost for switching back to the active state. However, in practice both speed scaling and power-down are applied to reduce the energy consumption of computing devices, thus it is more realistic to study algorithmic problems that combine both. The first theoretical work on this setting was due to Irani et al. [65] while much progress was recently made by Albers et al. [8], showing that the problem of minimizing the energy consumption on a single processor, with respect to a combined model of a single speed-scalable processor that is equipped with a sleep state is $\mathcal{NP}$-hard. Moreover, Antoniadis et al. [9] proposed a FPTAS for the latter case. These results provide evidence that the problems under the combined model are difficult to tackle with whilst attempting to solve them opens a challenging future direction.

MapReduce scheduling. Most of the theoretical models proposed for MapReduce scheduling (see e.g. [83, 40, 38]) involve the design of LP-based algorithms with LPs consisting of a large (polynomial) number of variables. As a result, the time complexity of such algorithms, although polynomial, appears almost not applicable for practical applications. A promising direction is to focus on the design of efficient combinatorial algorithms for MapReduce scheduling problems, by developing for example algorithmic techniques based on the duality paradigm of mathematical programming or on a charging scheme description.

In [83], Moseley et al. formulated MapReduce scheduling as a generalization of the classical two-stage flexible flow shop (FFS) problem for both identical and unrelated processors. As we show, in Section 3.5.2, the latter was generalized for a special third stage (formed due to the data shuffle in MapReduce framework). To this direction it would be of independent interest to improve upon these results and derive constant approximations for variants of multi-stage flow scheduling problems related to MapReduce scheduling.

In the online version of the MapReduce scheduling problem, constant-competitive algorithms have been proposed under resource (speed) augmentation, when preemption is allowed [83]. However, task preemption in MapReduce is usually quite different from that in classical CPU scheduling: when a task is suspended, it does not resume at a later time, but it is forced to start over again (see e.g., [113]). Therefore, it seems reasonable to study these problems under different online scheduling models (e.g. preemption-restart model [95]).

When designing scheduling strategies for MapReduce jobs one critical issue that must be taken into account by the scheduler is the *skew* management i.e., the load imbalance among map tasks (map skew) or reduce tasks (reduce skew). Usually in such techniques the unprocessed data (of tasks with the longest time-remaining) are scanned either locally or in parallel in order to collect information for repartitioning (see e.g. in [73]). An interesting direction for future work is to incorporate a skew mitigation technique together with

a preemptive scheduling policy into the MapReduce scheduling process.

Temperature-aware scheduling. Most of the theoretical study on temperature-aware scheduling problems aims to model the thermal and cooling behavior of processors. As mentioned in Section 1.1, the first approach to this direction was proposed by [25]. In fact, they studied the offline and online version of speed scaling scheduling with preemptions to minimize the maximum temperature. In the offline case, they proved that minimizing the maximum temperature, with respect to the deadline feasibility QoS measure, can be stated as a convex program and thus, can be solved optimally in polynomial time with arbitrary precision, applying the Elipsoid algorithm. An open question is the design of an efficient polynomial time combinatorial algorithm for this problem.

Another approach is the thermal and cooling mechanism that we adopt in Chapter 4, proposed by Chrobak et al. [41]. In this context, only a few results are known for the single processor case where jobs have arbitrary processing volumes (i.e., non unit-length). In fact, only recently Birks and Fung [33] studied the special case of equal-length jobs and proved upper and lower bounds on the competitive ratio of all deterministic online algorithms that depend on the cooling factor and the common length of jobs. Moreover, according to the thermal and cooling mechanism [41], the processor runs at constant speed, and the scheduler can leave an idle time unit whenever the execution of any available job violates the thermal threshold. However, instead of idling when the threshold is violated, it would be interesting to consider an extension of a model allowing speed scaling on a range of continuous or discrete speeds.

Bibliography

[1] Vincent Cohen-Addad, Zhentao Li, Claire Mathieu, and Ioannis Milis. Energy-efficient algorithms for non-preemptive speed-scaling To appear in *Proceedings of 12th Workshop on Approximation and Online Algorithms (WAOA)*, 2014.

[2] Foto N. Afrati, Dimitris Fotakis, and Jeffrey D. Ullman. Enumerating subgraph instances using MapReduce. In *Proceedings of the 29th IEEE Conference on Data Engineering, (ICDE '13)*, pages 62–73, 2013.

[3] Foto N. Afrati, Anish Das Sarma, Semih Salihoglu, and Jeffrey D. Ullman. Upper and Lower Bounds on the Cost of a MapReduce Computation. In *Proceedings of the 39th International Conference on Very Large Data Bases*, 6(4):277–288, 2013.

[4] Foto N. Afrati and Jeffrey D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Transactions on Knowledge and Data Engineering*, 23(9):1282–1298, 2011.

[5] Susanne Albers. Energy-efficient algorithms. *Communications of ACM*, 53(5):86–6, 2010.

[6] Susanne Albers. Algorithms for dynamic speed scaling. In *Proceedings of the 28th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 1–11, 2011.

[7] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4):49, 2007.

[8] Susanne Albers and Antonios Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. *ACM Transactions on Algorithms*, 10(2):9, 2014.

[9] Antonios Antoniadis, Chien-Chung Huang, and Sebastian Ott. A Fully Polynomial-Time Approximation Scheme for Speed Scaling with Sleep State. To appear in *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015.

[10] Lachlan LH. Andrew, Adam Wierman, and Ao Tang. Optimal speed scaling under arbitrary power functions. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):39–41, 2009.

[11] Eric Angel, Evripidis Bampis, and Fadi Kacem. Energy aware scheduling for unrelated parallel machines. In *Proceedings of the International Green Computing Conference (IGCC)*, pages 533–540, 2012.

[12] Eric Angel, Evripidis Bampis, and Vincent Chau. Throughput maximization in the speed-scaling setting. In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, pages 53–62, 2014.

[13] Eric Angel, Evripidis Bampis, Vincent Chau, and Dimitrios Letsios. Throughput maximization for speed-scaling with agreeable deadlines. In *Proceedings of the 10th International Conference Theory and Applications of Models of Computation (TAMC)*, pages 10–19, 2013.

[14] Eric Angel, Evripidis Bampis, Vincent Chau, and Nguyen K. Thang. Throughput maximization in multiprocessor speed-scaling. To appear in *Proceedings of the 25th International Symposium on Algorithms and Computation (ISAAC)*, 2014.

[15] Antonios Antoniadis and Chien-Chung Huang. Non-preemptive speed scaling. *Journal of Scheduling*, 16(4):385–394, 2013.

[16] Apple Inc. OS X Mavericks: core technologies overview. October, 2013.

[17] Leon Atkins, Guillaume Aupy, Daniel Cole, and Kirk Pruhs. Speed scaling to manage temperature. In *Proceedings of the 1st International Theory and Practice of Algorithms in (Computer) Systems (TAPAS)*, pages 9–20, 2011.

[18] Kenneth R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.

[19] Evripidis Bampis, Dimitrios Letsios, Giorgio Lucarelli, Vangelis Markakis, and Ioannis Milis. On multiprocessor temperature-aware scheduling problems. *Journal of Scheduling*, 16(5), 2013.

[20] Evripidis Bampis, Dimitrios Letsios, and Giorgio Lucarelli. Green scheduling, flows and matchings. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC)*, pages 106–115, 2012.

[21] Evripidis Bampis, Dimitrios Letsios, and Giorgio Lucarelli. A note on multiprocessor speed scaling with precedence constraints. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 138–142, 2014.

[22] Evripidis Bampis, Dimitrios Letsios, and Giorgio Lucarelli. Speed-scaling with no preemptions. To appear in *Proceedings of the 25th International Symposium on Algorithms and Computation (ISAAC)*, 2014.

[23] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 693–701, 2009.

[24] Nikhil Bansal and Subhash Khot. Inapproximability of hypergraph vertex cover and applications to scheduling problems. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 250–261, 2010.

[25] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1), 2007.

[26] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2009.

[27] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. *Algorithmica*, 60(4):877–889, 2011.

[28] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. *ACM Transactions on Algorithms (TALG)*, 9(2):18, 2013.

[29] Nikhil Bansal, Ho-Leung Chan, Kirk Pruhs, and Dmitriy Katz. Improved bounds for speed scaling in devices obeying the cube-root rule. In *Proceedings of the 36th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 144–155, 2009.

[30] Martin Birks and Stanley P. Y. Fung. Temperature aware online scheduling with a low cooling factor. In *Proceedings of the 7th International Conference in Theory and Applications of Models of Computation (TAMC)*, pages 105—-116, 2010.

[31] Martin Birks, Daniel Cole, Stanley P. Y. Fung, and Huichao Xue. Online algorithms for maximizing weighted throughput of unit jobs with temperature constraints. *Journal of Combinatorial Optimimization*, 26(2):237–250, 2013.

[32] Martin Birks and Stanley P. Y. Fung. Temperature aware online algorithms for minimizing flow time. In *Proceedings of the 10th International Conference in Theory and Applications of Models of Computation (TAMC)*, pages 20–31, 2013.

[33] Martin Birks and Stanley P. Y. Fung. Temperature aware online algorithms for scheduling equal length jobs. *Theoretical Computer Science*, 508:54–65, 2013.

[34] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[35] David P. Bunde. Power-aware scheduling for makespan and flow. *Journal of Scheduling*, 12(5):489–500, 2009.

[36] Rodrigo A. Carrasco, Garud Iyengar, and Clifford Stein. Energy aware scheduling for weighted completion time and weighted tardiness. *arXiv preprint abs/1110.0685*, 2011.

[37] Ho-Leung Chan, Wun-Tat Chan, Tak Wah Lam, Lap-Kei Lee, Kin-Sum Mak, and Prudence W. H. Wong. Energy efficient online deadline scheduling. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 795–804, 2007.

[38] Hyunseok Chang, Murali S. Kodialam, Ramana R. Kompella, T. V. Lakshman, Myungjin Lee, and Sarit Mukherjee. Scheduling in mapreduce-like systems for fast completion time. In *IEEE Proceedings of the 30th International Conference on Computer Communications (INFOCOM)*, pages 3074–3082, 2011.

[39] Chandra Chekuri and Sanjeev Khanna. Approximation algorithms for minimizing the weighted sum of completion times. In Joseph Y-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall/CRC, 2004.

[40] Fangfei Chen, Murali S. Kodialam, and T. V. Lakshman. Joint scheduling of processing and shuffle phases in mapreduce systems. In *IEEE Proceedings of the 31st International Conference on Computer Communications (INFOCOM)*, pages 1143–1151, 2012.

[41] Marek Chrobak, Christoph Dürr, Mathilde Hurand, and Julien Robert. Algorithms for temperature-aware task scheduling in microprocessor systems. *Sustainable Computing: Informatics and Systems*, 1(3):241–247, 2011.

[42] José R Correa, Martin Skutella, and José Verschae. The power of preemption on unrelated machines and applications to scheduling orders. *Mathematics of Operations Research*, 37(2):379–398, 2012.

[43] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150, 2004.

[44] James Donald and Margaret Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 78–88, 2006.

[45] Carla Schlatter Ellis. The case for higher-level power management. In *Proceedings of the 7th IEEE Workshop on Hot Topics in Operating Systems*, pages 162–167, 1999.

[46] Ericsson. Mobility report: on the pulse of networked society. June 2014.

[47] Eugen Feller, Lavanya Ramakrishnan, and Christine Morin. On the performance and energy efficiency of hadoop deployment models. In *Proceddings IEEE International Conference on Big Data*, pages 131–136, 2013.

[48] Boliang Feng, Jiaheng Lu, Yongluan Zhou, and Nan Yang. Energy efficiency for mapreduce workloads: An in-depth study. In *Proceedings of the Twenty-Third Australasian Database Conference-Volume 124*, pages 61–70, 2012.

[49] Amos Fiat, and Gerhard J. Woeginger. Online algorithms: The state of the art. *LNCS, Springer-Verlag 118*, 1998.

[50] Michael R. Garey, David S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[51] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[52] Naveen Garg, Amit Kumar, and Vinayaka Pandit. Order scheduling models: Hardness and algorithms. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 96–107, 2007.

[53] Íngo Goiri, Kien Le, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. Greenhadoop: leveraging green energy in data-processing frameworks. In *Proceedings of the 7th ACM European Conference on Computer Systems (EUROSYS)*, pages 57–70, 2012.

[54] Michael A. Gomaa, Mohamed D. Powell, and T. N. Vijaykumar. Heat-and-run: leveraging smt and cmp to manage power density through the operating system. *SIGPLAN Not.*, 39(11):260–270, 2004.

[55] Teofilo Gonzalez and Sartaz Sahni. Flowshop and jobshop schedules: complexity and approximation. *Operations research*, 26(1):36–52, 1978.

[56] Ronald L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[57] Ronald L. Graham. Bounds on multiprocessing anomalies. *SIAM Journal of Applied Mathematics*, 17:263–269, 1969.

[58] Leslie A. Hall. Approximation algorithms for scheduling. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-hard problems*, pages 1–45. PWS, Boston, 1997.

[59] Leslie Hall and David B. Shmoys. Jackson's rule for single machine scheduling: making a good heuristic better. *Mathematics of Operations Research*, 17:22–35, 1992.

[60] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

[61] Leslie A. Hall, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 142–151, 1996.

[62] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[63] Wiebke Höhn and Tobias Jacobs. On the performance of smith's rule in single-machine scheduling with nonlinear cost. In *Proceedings of the 11th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 482–493, 2012.

[64] Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[65] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3, 2007.

[66] James R. Jackson. Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, UCLA, 1955.

[67] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, pages 61–68, 1954.

[68] Nikolai Joukov and Josef Sipek. Greenfs: Making enterprise computers greener by protecting them better. In *ACM SIGOPS Operating Systems Review*, volume 42, pages 69–80, 2008.

[69] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010.

[70] Amit Kumar, Rajsekar Manokaran, Madhur Tulsiani, and Nisheeth K. Vishnoi. On lp-based approximability for strict csps. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1560–1573, 2011.

[71] Amit Kumar, Li Shang, Li-Shiuan Peh, and Niraj K Jha. Hybdtm: a coordinated hardware- software approach for dynamic thermal management. In *Proceedings of the 43rd Annual Conference on Design Automation*, pages 548–553, 2006.

[72] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in MapReduce and streaming. In *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–10, 2013.

[73] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (PODS)*, pages 25–36, 2012.

[74] Tak-Wah Lam, Lap-Kei Lee, Isaac K-K. To, and Prudence W.H. Wong. Online speed scaling based on active job count to minimize flow plus energy. *Algorithmica*, 65(3):605–633, 2013.

[75] Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinooy Kan, and David B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, editors, *Handbooks in Operations Research and Management Science, Vol 4., Logistics of Production and Inventory*, pages 445–522. North-Holland, 1993.

[76] Jan Karel Lenstra, Alexander H.G. Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[77] Joseph Y.-T. Leung, Haibing Li, and Michael Pinedo. Scheduling orders for multiple product types to minimize total weighted completion time. *Discrete Applied Mathematics*, 155(8):945–970, 2007.

[78] Jacob Leverich and Christos Kozyrakis. On the energy (in)efficiency of hadoop clusters. *ACM SIGOPS Operating Systems Review*, 44(1):61–65, 2010.

[79] Minming Li and F. Frances Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM Journal of Computing*, 35(3):658–671, 2005.

[80] Monaldo Mastrolilli, Maurice Queyranne, Andreas S. Schulz, Ola Svensson, and Nelson A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010.

[81] Nicole Megow and José Verschae. Scheduling on a machine with varying speed: Minimizing cost and energy via dual schedules. *arXiv preprint abs/1211.6216*, 2012.

[82] Nicole Megow and José Verschae. Dual techniques for scheduling on a machine with varying speed. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 745–756, 2013.

[83] Benjamin Moseley, Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. On scheduling in map-reduce and flow-shops. In *Proceedings of the 23rd ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 289–298, 2011.

[84] Arkadii Nemirovski, and Yurii Nesterov. *Interior Point Polynomial Algorithms in Convex Programming.* Society for Industrial and Applied Mathematics, 1994.

[85] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[86] Cynthia A. Phillips, Clifford Stein, and Joel Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223, 1998.

[87] Michael Pinedo. *Scheduling: theory, algorithms, and systems.* Springer, 2012.

[88] Chris N. Potts. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 28:1436–1441, 1980.

[89] Apache Hadoop Project. Powered by hadoop. In *http://wiki.apache.org/hadoop/PoweredBy*, 2011.

[90] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4, 2008.

[91] Kirk Pruhs, Rob van Stee, and Patchrawat Uthaisombut. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems*, 43:67–80, 2008.

[92] Thomas A. Roemer. A note on the complexity of the concurrent open shop problem. *Journal of Scheduling*, 9:389–396, 2006.

[93] Andreas Schulz and Martin Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15:450–469, 2002.

[94] Petra Schuurman and Gerhard J. Woeginger. A polynomial time approximation scheme for the two-stage multiprocessor flow shop problem. *Theoretical Computer Science*, 237(1):105–122, 2000.

[95] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, 1995.

[96] David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

[97] Martin Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the ACM*, 48(2):206–242, 2001.

[98] Martin Skutella. List scheduling in order of a-points on a single machine. In Evripidis Bampis, Klaus Jansen, and Claire Kenyon, editors, *Efficient Approximation and Online Algorithms: Recent Progress on Classical Combinatorial Optimization Problems and New Applications*, volume 3484 of *Lecture Notes in Computer Science*, pages 250–291. Springer, 2006.

[99] Daniel D. Sleator and Robert E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM* , 32(3):652–686, 1985.

[100] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[101] Sebastian Stiller and Andreas Wiese. Increasing speed scheduling and flow scheduling. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC)*, pages 279–290, 2010.

[102] Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakesh Patel, and Franklin Baez. Reducing power in high-performance microprocessors. In *Proceedings of the ACM 35th annual Design Automation Conference*, pages 732–737, 1998.

[103] Jeffrey D. Ullman. Designing good mapreduce algorithms. *XRDS: Crossroads, The ACM Magazine for Students*, 19(1):30–34, 2012.

[104] Oscar C. Vásquez. Energy in computing systems with speed scaling: optimization and mechanisms design. *arXiv preprint abs/1212.6375*, 2012.

[105] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2003.

[106] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer S.J. Pister. Smart dust: Communicating with a cubic-millimeter computer. *IEEE Computer*, 34(1):44–51, 2001.

[107] Adam Wierman, Lachlan L.H. Andrew, and Ao Tang. Power-aware speed scaling in processor sharing systems. In *Proceedings of The 33rd Annual IEEE International Conference on Computer Communications (INFOCOM)*, pages 2007–2015, 2009.

[108] David P. Williamson and David B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[109] Thomas Wirtz and Rong Ge. Improving mapreduce energy efficiency for computation intensive workloads. In *Proceedings of the International Green Computing Conference (IGCC)*, pages 1–8, 2011.

[110] Jun Yang, Xiuyi Zhou, Marek Chrobak, Youtao Zhang, and Lingling Jin. Dynamic thermal management through task scheduling. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 191–201, 2008.

[111] Frances F. Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995.

[112] Dongjin-J. Yoo and Kwang M. Sim. A comparative review of job scheduling for mapreduce. In *IEEE Proceedings of the International Symposium on Cloud Computing and Intelligece Systems (CCIS)*, pages 353–358, 2011.

[113] Matei Zaharia, Dhruba Borthakur, Joydeep S. Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Job scheduling for multi-user mapreduce clusters. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55*, 2009.

Appendix A

General form of KKT conditions

A useful tool throughout our analysis is the well-known Karush, Kuhn, Tucker (KKT) conditions, which are first order, necessary and sufficient conditions, for a solution in convex programming to be optimal.

Assume that we are given the following convex program:

$$\begin{aligned} \min & f(x) \\ g_i(x) &\leq 0, & 1 \leq i \leq q \\ h_j(x) &= 0, & 1 \leq j \leq r \\ x &\in \Re^n \end{aligned}$$

Suppose that the program is strictly feasible, i.e. there is a point x such that $g_i(x) < 0$ and $h_j(x) = 0$ for all $1 \leq i \leq q$ and $1 \leq j \leq r$, where all functions g_i and h_j are differentiable at x. Let λ_i and μ_j be the dual variables associated to the constraints $g_i(x) \leq 0$ and $h_j(x) = 0$, respectively. The Karush-Kuhn-Tucker (KKT) conditions are:

$$g_i(x) \leq 0, \qquad 1 \leq i \leq q \tag{1}$$

$$h_j(x) = 0, \qquad 1 \leq j \leq r \tag{2}$$

$$\lambda_i \geq 0, \qquad 1 \leq i \leq q \tag{3}$$

$$\lambda_i g_i(x) = 0, \qquad 1 \leq i \leq q \tag{4}$$

$$\nabla f(x) + \sum_{i=1}^{q} \lambda_i \nabla g_i(x) + \sum_{j=1}^{r} \mu_j \nabla h_j(x) = 0 \tag{5}$$

KKT conditions are necessary and sufficient for solutions $x \in \Re^n$, $\lambda \in \Re^q$ and $\mu \in \Re^r$ to be primal and dual optimal, where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_q)$ and $\mu = (\mu_1, \mu_2, \ldots, \mu_r)$. We refer to the conditions (1) and (2) as primal feasible, to (3) as dual feasible, to (4) as complementary slackness and to (5) as stationarity conditions, respectively.

List of Figures

List of Algorithms

Problèmes algorithmiques dans les systèmes informatiques sous contraintes d'énergie

Resumé: Cette thèse se focalise sur des algorithmes efficaces en énergie pour des problèmes d'ordonnancement de tâches sur des processeurs pouvant varier la vitesse d'exécution ainsi que sur des processeurs fonctionnant sous un mécanisme de réchauffement-refroidissement, où pour un budget d'énergie donné ou un seuil thermique, l'objectif consiste à optimiser un critère de Qualité de Service. Une partie de notre recherche concerne des problèmes d'ordonnancement de tâches apparaissant dans des environnements de traitement de grandes données. Dans ce contexte, nous nous focalisons sur le paradigme MapReduce en considérant des problèmes d'ordonnancement efficaces en énergie sur un ensemble de processeurs, ainsi que pour la version classique.

D'un côté, nous proposons des résultats de complexité, des algorithmes optimaux ou approchés pour différentes variantes du problème de la minimisation du retard maximal d'un ensemble de tâches sur un seul processeur pouvant varier la vitesse d'exécution. Ensuite, nous considérons le problème d'ordonnancement MapReduce dans les versions avec la consommation d'énergie ou non sur des processeurs non-reliés où le but est de minimiser le temps d'achèvement pondéré. Nous étudions deux cas spéciaux et les généralisations de ces deux problèmes en proposant des algorithmes d'approximation constante. Enfin, nous étudions le problème d'ordonnancement sous contraintes thermiques sur un seul processeur fonctionnant en-dessous d'un seuil de température stricte où chaque tâche a sa propre contribution thermique et le but est de maximiser le nombre de tâche exécutée. Nous considérons le cas où les tâches ont des durées unitaires et ayant la même date d'échéance.
Mots clés : Ordonnancement, algorithme d'approximation, processeur de variation de vitesse, seuil thermique, énergie, gestion de température, ordonnancement MapReduce, algorithme en-ligne.

Algorithmic problems in power management of computing systems

Abstract: This thesis is focused on energy-efficient algorithms for job scheduling problems on speed-scalable processors, as well as on processors operating under a thermal and cooling mechanism, where, for a given budget of energy or a thermal threshold, the goal is to optimize a Quality of Service criterion. A part of our research concerns scheduling problems arising in large-data processing environments. In this context, we focus on the MapReduce paradigm and we consider problems of energy-efficient scheduling on multiple speed-scalable processors as well as classical scheduling on a set of unrelated processors.

First, we propose complexity results, optimal and constant competitive algorithms for

different energy-aware variants of the problem of minimizing the maximum lateness of a set of jobs on a single speed-scalable processor. Then, we consider energy-aware MapReduce scheduling as well as classical MapReduce scheduling (where energy is not our concern) on unrelated processors, where the goal is to minimize the total weighted completion time of a set of MapReduce jobs. We study special cases and generalizations of both problems and propose constant approximation algorithms. Finally, we study temperature-aware scheduling on a single processor that operates under a strict thermal threshold, where each job has its own heat contribution and the goal is to maximize the schedule's throughput. We consider the case of unit-length jobs with a common deadline and we study the approximability of CoolestFirst scheduling, i.e., the job with the smaller heat contribution is scheduled first.
Keywords: Scheduling, approximation algorithm, speed-scalable processor, thermal threshold, energy-efficiency, temperature management, MapReduce scheduling, online algorithm.