Energy Efficient Scheduling of MapReduce Jobs

Evripidis Bampis[1], Vincent Chau[2], Dimitrios Letsios[1], Giorgio Lucarelli[1], Ioannis Milis[3], and Georgios Zois[1,3]

[1] Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, France.
{Evripidis.Bampis,Dimitrios.Letsios,Giorgio.Lucarelli, Georgios.Zois}@lip6.fr
[2] IBISC, Université d'Évry, France.
vincent.chau@ibisc.univ-evry.fr
[3] Dept. of Informatics, AUEB, Athens, Greece.
milis@aueb.gr

Abstract. MapReduce is emerged as a prominent programming model for data-intensive computation. In this work, we study power-aware MapReduce scheduling in the speed scaling setting first introduced by Yao et al. [FOCS 1995]. We focus on the minimization of the total weighted completion time of a set of MapReduce jobs under a given budget of energy. Using a linear programming relaxation of our problem, we derive a polynomial time constant-factor approximation algorithm. We also propose a convex programming formulation that we combine with standard list scheduling policies, and we evaluate their performance using simulations.

1 Introduction

MapReduce has been established as a standard programming model for parallel computing in data centers or computational grids and it is currently used for several applications including search indexing, web analytics or data mining. However, data centers consume an enormous amount of energy and hence, energy efficiency has emerged as an important issue in the data-processing framework. Several empirical works have been carried-out in order to study different mechanisms for the reduction of the energy consumption in the MapReduce setting and especially for the Hadoop framework [6,7,9]. The main mechanisms for energy saving are the *power-down* mechanism where in periods of low-utilization some servers are switched-off, and the *speed-scaling* mechanism (or DVFS for Dynamic Voltage Frequency Scaling) where the servers' speeds may be adjusted dynamically [21]. Until lately, most work in the MapReduce framework were focused on the power-down mechanism, but recently, Wirtz and Ge [20] showed that for some computation intensive MapReduce applications the use of intelligent speed-scaling may lead to significant energy savings. In this paper, we study power-aware MapReduce scheduling in the speed scaling setting from a theoretical point of view.

In a typical MapReduce framework, the execution of a MapReduce job creates a number of Map and Reduce tasks. Each Map task processes a portion of the input data and outputs a number of key-value pairs. All key-value pairs having the same key are then given to a Reduce task which processes the values associated with a key to generate the final result. This means that each Reduce task cannot start before the completion of the last Map task of the same job. In other words, there is a complete bipartite graph implying the precedences between Map and Reduce tasks of a job. However, the Map tasks of a job can be executed in parallel and the same holds for its Reduce tasks.

In what follows we consider a set of MapReduce jobs that have to be executed on a set of speed-scalable processors, i.e., on processors that can adjust dynamically their speed [21]. In the speed scaling setting, each task is associated with a work volume instead of a processing time and the scheduler has to decide not only the processor and the time interval where a task is executed, but also its speed over time, taking into account the energy consumption. High processor's speeds are in favor of performance at the price of high energy consumption. Each job consists of a set of Map tasks and a set of Reduce tasks, with every task having a positive work volume. Each job is also associated with a positive weight representing its importance/priority, and a release date (or arrival time). Like in [4,5], we consider that the Map and the Reduce tasks of each job are *preassigned* to the processors and in this way we take into account data locality, i.e. the fact that each Map task has to be executed on the server where its data are located. Given that the preemption of tasks, i.e. the possibility of interrupting a task and resuming it later, may cause important overheads we do not allow it. This is also the case often in practice: Hadoop does not offer the possibility of preemption [14]. Our goal is to schedule all the tasks to the processors, so as to minimize the total weighted completion time of jobs respecting a given budget of energy.

Related Work. Chang et al. [4] consider a set of MapReduce jobs with their Map and Reduce tasks preassigned to processors and their goal is to minimize the total weighted completion time of jobs. They proposed approximation algorithms of ratios 3 and 2 for arbitrary and common release dates, respectively. However, they do not consider neither distinction nor dependencies between Map and Reduce tasks of a job. Moreover, their model falls into a well-studied problem known as *concurrent open-shop* (or *order scheduling*) for which the same approximation results are known (see [12] and the references therein). Extending on the abovementioned model, Chen et al. [5], proposed a more realistic one which takes into account the dependencies among Map and Reduce tasks and derived an 8-approximation algorithm for the same objective. Moreover, they managed to model also the transfer of the output of Map tasks to Reduce tasks and to derive a 58-approximation algorithm for this generalization. In a third model proposed by Moseley et al. [14], the dependencies between Map and Reduce tasks of a job are also taken into account while the assignment of tasks to processors is not given in advance. The authors studied the preemptive variant for both the case of identical and unrelated processors. They proposed constant approximation ratios of 12 and 6, respectively. For the unrelated processors case, they focused on the special case where each job has a single Map and a single Reduce task.

For the latter case on a single map and a single reduce processor they also proposed a QPTAS which becomes a PTAS for a fixed number of processing times of tasks. Recently, in [8] the authors proposed a $32 + \epsilon$-approximation algorithm for the unrelated processors case with multiple Map and Reduce tasks per job.

In the energy-aware setting, Angel et al. [2] proposed approximation algorithms for the problem of minimizing the total weighted completion time on unrelated parallel processors, under a model where the processing time and the energy consumption of the jobs are speed dependent. Other works in this setting, related to our problem, deal with single processor problems. Megow et al. [13] recently proposed a PTAS for the problem of minimizing the total weighted completion time on a single speed-scalable processor.

Our Results and Organization of the Paper We adopt the MapReduce model of [4] where the tasks are preassigned to processors but extended with dependencies between Map and Reduce tasks as in Chen et al. [5,14] in the speed scaling setting [21]. After a formal statement of our problem and notation, we present, in Section 3, a polynomial time LP-based $O(1)$-energy $O(1)$-approximation algorithm which allows energy augmentation, i.e., it may use more energy than an optimal solution which always respects the energy budget, while using discretization of the possible speed values and list scheduling in the order of tasks' α-points (see e.g. [16,11,19]). As we show, there is a tradeoff between the approximation ratio and energy augmentation as a function of α, where the schedule is converted to a constant-factor approximation for our problem. In Section 4, we are interested in natural list scheduling policies such as First Come First Serve (FCFS) and Smith Rule (SR). However, in our context we need to determine the speeds of every task in order to respect the energy budget. For that, we propose a convex programming relaxation of our problem when an order of the jobs is prespecified. This relaxation can be solved in polynomial time to arbitrary precision by the Ellipsoid algorithm [15]. Then we combine the solution of this relaxation with FCFS and SR and we compare experimentally their effectiveness. Finally, we conclude in Section 5.

2 Problem Definition and Notation

In the sequel 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 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 can start its execution after the completion of all Map tasks of the same job.

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 $v_{\max} = \max_{T_{i,j} \in \mathcal{T}}\{v_{i,j}\}$.

In this paper, we combine this abstract model for MapReduce scheduling with the speed scaling mechanism for energy saving [21] (see also [1] for a recent review). In this setting the power required by a processor running at time t with speed $s(t)$ is equal to $P((s(t)) = s(t)^{\beta}$, for a constant $\beta > 1$ (typical values of β are between 2 and 3) and its energy consumption is power integrated over time, i.e., $E = \int P(s(t))dt$.

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. 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 the total weighted completion time of the schedule, i.e., $\sum_{j \in \mathcal{J}} w_j C_j$, without exceeding the energy budget E. We refer to this problem as MapReduce problem.

As already mentioned above, the special case of the MapReduce problem where there are not dependencies between Map and Reduce tasks of each job and each processor runs at a constant speed, reduces to the *concurrent open-shop* problem which is known to be strongly $\mathcal{NP}$-complete [17]. It is also easy to adapt $\mathcal{NP}$-completeness reductions like the one for the concurrent open-shop problem in the speed scaling setting, and therefore, the MapReduce problem is also strongly $\mathcal{NP}$-hard.

3 A Linear Programming Approach

In this section we present a constant-factor approximation algorithm for the MapReduce problem. Our algorithm allows energy augmentation and derives a $O(1)$-energy $O(1)$-approximation schedule for the problem. As we show, there is a tradeoff where the schedule can be converted to a constant-factor approximate schedule for the MapReduce problem.

Our algorithm is based on a formulation of the problem as a linear programming relaxation. Then, we transform the solution obtained by the linear program to a feasible schedule for the MapReduce problem using the technique of α-points.

3.1 Discretization of Speeds

Before presenting the linear programming formulation, our first step is to discretize the possible speed values by loosing a factor of $(1+\epsilon)$ with respect to an optimal solution. In order to do this, we need the following propositions that bound the length of an optimal schedule and the possible speed values.

Proposition 1. *The makespan of any optimal schedule for the* 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 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. $\square$

Proposition 2. *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 1 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. $\square$

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 $s_{\max} = s_L(1+\epsilon)^k$.

Lemma 1. *There is a feasible $(1+\epsilon)$-approximate schedule for the* 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 $(1+\epsilon)$. $\square$

Henceforth we will consider the 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-MapReduce.

3.2 Linear Programming Relaxation

In what follows we give an interval-indexed linear programming relaxation of the DS-MapReduce problem. In order to do this, we discretize the time horizon of an optimal schedule as follows. By Proposition 1, 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 $T_{i,j}$ 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-MapReduce problem.

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

subject to :

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

$$\sum_{j:T_{i,j} \in \mathcal{T}} \sum_{s \in \mathcal{V}} y_{i,j,s,t} \leq 1, \qquad \forall i \in \mathcal{P}, 0 \leq t \leq u \tag{2}$$

$$C_{i,j} \geq \frac{1}{2} \sum_{s \in \mathcal{V}} y_{i,j,s,0}|I_0| \left(\frac{1}{p_{i,j,s}} + 1 \right) + \sum_{t=1}^{u} \sum_{s \in \mathcal{V}} \left(\frac{y_{i,j,s,t}|I_t|}{p_{i,j,s}} \tau_t + \frac{1}{2} y_{i,j,s,t}|I_t| \right), \quad \forall T_{i,j} \in \mathcal{T} \tag{3}$$

$$C_j \geq C_{i,j}, \qquad \forall T_{i,j} \in \mathcal{T} \tag{4}$$

$$\sum_{T_{i,j} \in \mathcal{T}} \sum_{s \in \mathcal{V}} \sum_{t=0}^{u} y_{i,j,s,t}|I_t| s^{\beta} \leq E \tag{5}$$

$$\sum_{t=0}^{\ell} \sum_{s \in \mathcal{V}} \frac{y_{i,j,s,t}|I_t|}{p_{i,j,s}} \geq \sum_{t=0}^{\ell} \sum_{s \in \mathcal{V}} \frac{y_{i',j,s,t}|I_t|}{p_{i',j,s}}, \qquad \forall T_{i,j} \in \mathcal{M}, T_{i',j} \in \mathcal{R}, 0 \leq \ell \leq u \tag{6}$$

$$y_{i,j,s,t} = 0, \qquad \forall T_{i,j} \in \mathcal{T}, s \in \mathcal{V}, t : \tau_t < r_j \tag{7}$$

$$y_{i,j,s,t}, C_{i,j}, C_j \geq 0, \qquad \forall T_{i,j} \in \mathcal{T}, s \in \mathcal{V}, 0 \leq t \leq u \tag{8}$$

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 (1) ensures that $T_{i,j}$ is entirely executed. Constraints (2) enforce that the total amount of processing time that is executed within an interval I_t cannot exceed its length. In [18], 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). Constraints (4) ensure that the completion time of each job is the maximum over the completion times of all its tasks. Constraint (5) 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 (6) 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 (7) 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 $(\bar{y}_{i,j,s,t}, \bar{C}_{i,j}, \bar{C}_j)$.

3.3 The Algorithm

In this section we use (LP) to derive a feasible schedule for the DS-MapReduce problem. Depending on the choice of some parameters, this schedule may exceed the energy budget. As we show, there is a tradeoff where a constant factor approximation ratio can be derived.

Our algorithm is based on the idea of list scheduling in order of α-points [10]. 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.,

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

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, combining the ideas of [5] with the notion of α-points [10], 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 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

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

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 tradeoff between the energy consumption and the weighted completion time of jobs. 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 $\mathcal{MR}(\alpha, \gamma)$ gives a formal description of our algorithm.

Note that the processing time of a task $T_{i,j} \in \mathcal{T}$ to an optimal solution to (LP) is

$$\bar{p}_{i,j} = \sum_{t=0}^{u} \sum_{s \in \mathcal{V}} \bar{y}_{i,j,s,t}|I_t|$$

```
ALGORITHM $\mathcal{MR}(\alpha, \gamma)$
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 $\alpha$-point $t^{\alpha}_{i,j}$, the processing time $p_{i,j}$ and the speed $s_{i,j}$.
4: for each processor $i \in \mathcal{P}$ do
5:     Compute the priority list $\sigma_i$.
6: for each time where a processor $i \in \mathcal{P}$ becomes available do
7:     Select the first available task, let $T_{i,j}$, in $\sigma_i$ which has not been yet executed.
8:     Schedule $T_{i,j}$, non-preemptively, with processing time $p_{i,j}$.
       Let $C_{i,j}$ be the completion time of task $T_{i,j}$.
9: for each job $j \in \mathcal{J}$ do
10:    Compute its completion time $C_j = \max_{i \in \mathcal{P}} C_{i,j}$.
```

Hence, the energy consumption $\bar{E}_{i,j} = \sum_{s \in \mathcal{V}} \sum_{t=0}^{u} \bar{y}_{i,j,s,t} |I_t| s^{\beta}$ 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 give the relation between these two quantities, we need the following technical lemma.

Lemma 2. *Let $s_1, s_2, \ldots, s_k$ and $a_1, a_2, \ldots, a_k$ be positive values and $\beta > 2$. Then, it holds that*

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

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

$$\left(\frac{\sum_{i=1}^{k} a_i}{\sum_{i=1}^{k} a_i \frac{1}{s_i}} \right)^{\beta - 1} \leq \frac{\sum_{i=1}^{k} a_i s_i^{\beta-1}}{\sum_{i=1}^{k} a_i} \tag{9}$$

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

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

which is translated as

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

Therefore, in order to show inequality (9), it suffices to show that

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

Thus, it suffices to prove that

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

An equivalent representation of the above expression is

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

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

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. □

Lemma 3. *Let $\bar{E}_{i,j}$ and $E_{i,j}$ be the energy consumption of the task $T_{i,j} \in \mathcal{T}$ to the optimal solution to (LP) and to the solution of* Algorithm $\mathcal{MR}(\alpha, \gamma)$*, 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

$$E_{i,j} = v_{i,j} s_{i,j}^{\beta-1} = v_{i,j}\left(\frac{v_{i,j}}{p_{i,j}}\right)^{\beta-1}$$

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

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 2 we get

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

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. □

The following lemma provides a lower bound to the completion time $\bar{C}_{i,j}$ of the task $T_{i,j} \in \mathcal{T}$ given by the (LP).

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

The following theorem gives the approximation ratio of ALGORITHM $\mathcal{MR}(\alpha,\gamma)$.

Theorem 1. ALGORITHM $\mathcal{MR}(\alpha,\gamma)$ *is a* $\frac{1}{\gamma^{\beta-1}\alpha^{\beta}}$*-energy* $\frac{\gamma^2+3\gamma+1}{1-\alpha}(1+\delta)$*-approximation algorithm for the* DS-MAPREDUCE *problem, where* $\alpha \in (0,1)$*,* $\gamma > 0$ *and* $\delta > 0$*.*

Proof. Consider the schedule $\mathcal{S}$ produced by ALGORITHM $\mathcal{MR}(\alpha,\gamma)$ 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 of 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

$$C_{i,j} \leq \tau_{t^{\alpha}_{i,j}+1} + \sum_{k \in \sigma_i(j)} p_{i,k} \tag{10}$$

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

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

where the last inequality holds by applying constraint (2) of the (LP). Thus, from inequality (10) we have

$$C_{i,j} \leq (\gamma+1)\tau_{t^{\alpha}_{i,j}+1} \tag{11}$$

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 Reduce 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

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

where the first inequality holds by inequality (11) and the second by the constraint (6) 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

$$C_{i,j} \leq t + p_{i,j'} + \sum_{k \in \sigma_i(j)} p_{i,k} \tag{12}$$

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 executed at time t and hence it is available. Then, by equation (12) we have

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

As $\tau_{t^{\alpha}_{i,j}+1} = (1+\delta)\tau_{t^{\alpha}_{i,j}}$, using Lemma 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 (4) 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-MapReduce problem, the theorem follows. $\square$

Note that in the absence of precedence constraints between the tasks, the above analysis can be improved. Indeed, we can consider that all tasks are Map tasks, and hence an upper bound to their completion time into the schedule created by ALGORITHM $\mathcal{MR}(\alpha, \gamma)$ is given by Inequality (11). Then, the following corollary holds.

Corollary 1. ALGORITHM $\mathcal{MR}(\alpha, \gamma)$ *is a* $\frac{1}{\gamma^{\beta-1}\alpha^{\beta}}$*-energy* $\frac{\gamma+1}{1-\alpha}(1+\delta)$*-approximation algorithm for the* DS-MAPREDUCE *problem without precedence constraints, where* $\alpha \in (0, 1)$*,* $\gamma > 0$ *and* $\delta > 0$*.*

Moreover, in the absence of both precedence constraints between tasks and release dates of jobs, our analysis can be further improved. As before, we can consider that all tasks are Map tasks. In addition, we can drop the demand that a task $T_{i,j} \in \mathcal{T}$ becomes available for the algorithm after the time $\tau_{t^{\alpha}_{i,j}+1}$. Hence, Inequality (10) is simplified to $C_{i,j} \leq \sum_{k \in \sigma_i(j)} p_{i,k}$, as all tasks are released at time 0 and they are available at any time. Then, the following corollary holds.

Corollary 2. ALGORITHM $\mathcal{MR}(\alpha, \gamma)$ *is a* $\frac{1}{\gamma^{\beta-1}\alpha^{\beta}}$*-energy* $\frac{\gamma}{1-\alpha}(1+\delta)$*-approximation algorithm for the* DS-MAPREDUCE *problem without precedence constraints and release dates, where* $\alpha \in (0, 1)$*,* $\gamma > 0$ *and* $\delta > 0$*.*

By combining Lemma 1, Theorem 1 and Corollaries 1 and 2, and as we can select an ε such that $(1 + \delta)(1 + \epsilon) \leq (1 + \varepsilon)$, the following theorem holds.

Theorem 2. *There is a* $\frac{1}{\gamma^{\beta-1}\alpha^{\beta}}$*-energy* $\frac{\gamma^2+3\gamma+1}{1-\alpha}(1+\varepsilon)$*-approximation algorithm for the* MAPREDUCE *problem, a* $\frac{1}{\gamma^{\beta-1}\alpha^{\beta}}$*-energy* $\frac{\gamma+1}{1-\alpha}(1 + \varepsilon)$*-approximation algorithm for the* MAPREDUCE *problem without precedence constraints, and a* $\frac{1}{\gamma^{\beta-1}\alpha^{\beta}}$*-energy* $\frac{\gamma}{1-\alpha}(1 + \varepsilon)$*-approximation algorithm for the* MAPREDUCE *problem without precedence constraints and release dates, where* $\alpha \in (0, 1)$*,* $\gamma > 0$ *and* $\varepsilon > 0$*.*

In Fig.1 we depict a tradeoff between energy augmentation and approximation ratio for some practical values of β. Note that, by choosing $\gamma = \frac{1}{\alpha\sqrt[\beta-1]{\alpha}}$, energy augmentation is not allowed and the schedule can be converted to a constant-factor approximate schedule. In this case the following theorem holds.

Theorem 3. *There is a* $\frac{(\alpha\sqrt[\beta-1]{\alpha})^2+3\alpha\sqrt[\beta-1]{\alpha}+1}{(\alpha\sqrt[\beta-1]{\alpha})^2(1-\alpha)}(1+\varepsilon)$*-approximation algorithm for the* MAPREDUCE *problem, a* $\frac{\alpha\sqrt[\beta-1]{\alpha}+1}{\alpha\sqrt[\beta-1]{\alpha}(1-\alpha)}(1+\varepsilon)$*-approximation algorithm for the* MAPREDUCE *problem without precedence constraints, and a* $\frac{1}{\alpha\sqrt[\beta-1]{\alpha}(1-\alpha)}(1 + \varepsilon)$*-approximation algorithm for the* MAPREDUCE *problem without precedence constraints and release dates, where* $\alpha \in (0, 1)$ *and* $\varepsilon > 0$*.*

The ratios of the above theorem can be optimized by selecting the appropriate value of α for each β. Table 1 gives the achieved ratios for practical values of β.

β	general	without precedence	without precedence & without 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 1. Approximation ratios for the MapReduce problem for different values of β.

4 A Convex Programming Approach

We are interested in natural list scheduling policies such as First Come First Serve (FCFS) and Smith Rule (SR). However, in our context we need to determine the speeds of every task in order to respect the energy budget. For that, we propose a convex programming relaxation of our problem when an order of the jobs is prespecified.

4.1 The Convex Program

Let $\sigma = \langle 1, 2, \ldots, n \rangle$ be a given order of the jobs. Consider now the restricted version of the 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 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 MapReduce(σ) problem.

In order to formulate our problem as a convex program, let $p_{i,j}$ be a variable that corresponds to the processing time of task $T_{i,j} \in \mathcal{T}$. Moreover, for each task $T_{i,j} \in \mathcal{T}$, we introduce a variable $C_{i,j}$ that determines 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. Consider the following convex programming formulation of the

MapReduce(σ) problem.

$$\begin{aligned}
&(CP) : \text{minimize} \sum_{j \in \mathcal{J}} w_j C_j \\
&\text{subject to :} \\
&\sum_{T_{i,j} \in \mathcal{T}} \frac{v_{i,j}^{\beta}}{p_{i,j}^{\beta-1}} \leq E && && (13) \\
&r_{j'} + \sum_{k=j'}^{j} p_{i,k} \leq C_{i,j}, && \forall T_{i,j}, T_{i,j'} \in \mathcal{T}, j' \prec j && (14) \\
&C_{i',j} + p_{i,j} \leq C_{i,j}, && \forall T_{i,j} \in \mathcal{R}, T_{i',j} \in \mathcal{M} && (15) \\
&C_{i,j} \leq C_j, && \forall T_{i,j} \in \mathcal{T} && (16) \\
&s_{i,j}, C_{i,j}, C_j \geq 0, && \forall T_{i,j} \in \mathcal{T}, j \in \mathcal{J}
\end{aligned}$$

The objective function of (CP) is to minimize the weighted completion time of all jobs. Constraint (13) guarantees that the energy budget is not exceeded. Constraints (14) and (15) 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) will return the optimal value of the objective function, instead of a lower bound of it, as constraints (14) describe in a complete way the completion times of the tasks. However, this is not true for constraints (15) which are responsible for the precedence constraints. Finally, constraints (16) ensure that the completion time of each job is the maximum over the completion times among all of its tasks.

As the optimal solution to (CP) does not necessarily describe a feasible schedule, we need to apply an algorithm that uses the processing times found by (CP) and the order σ so as to create a feasible schedule for the MapReduce(σ) problem, and hence for the MapReduce problem. In fact, it suffices to apply, for example, the Lines 6-8 of Algorithm $\mathcal{MR}(\alpha, \gamma)$, by considering the same order for all processors.

4.2 Scheduling Policies

In this section we propose different orders of jobs and we discuss how far is an optimal solution for the MapReduce(σ) problem using these orders with respect to the optimal solution for the MapReduce problem. Two standard orders of jobs are the following.

First Come First Serve (FCFS): for each pair of jobs $j, j' \in \mathcal{J}$, if $r_j < r_{j'}$ then $j \prec j'$ in σ.

Smith Rule (SR): 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.

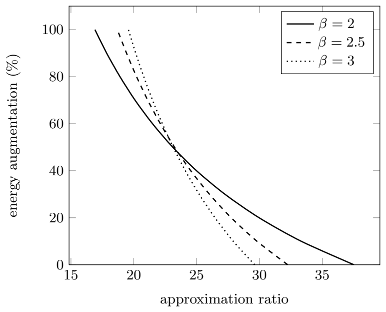

Fig. 1. Tradeoff between energy augmentation and approximation ratio when $\beta = \{2, 2.5, 3\}$.

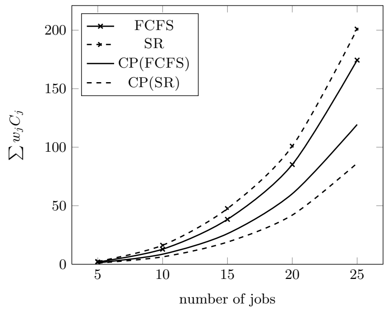

Fig. 2. Experimental comparison of the solutions of FCFS and SR (scaled down by a factor of 10^3).

The following propositions present negative results concerning the approximation ratio that we can achieve if we use the FCFS or the SR order.

Proposition 3. *Let OPT and OPT_{FCFS} be the optimal solutions for the* MapReduce *and the* MapReduce(FCFS) *problems, respectively. There is an instance for which it holds that $\frac{OPT_{\text{FCFS}}}{OPT} = \Omega(n)$.*

Proof. Consider an instance 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 MapReduce and the 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 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 $OPT = O(m^2)$.

On the other hand, in an optimal solution for the 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 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 objective of the 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

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

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

Proposition 4. *Let OPT and OPT_{SR} be the optimal solutions for the* MapReduce *and the* MapReduce(SR) *problems, respectively. There is an instance for which it holds that $\frac{OPT_{\text{SR}}}{OPT} = \Omega(n)$.*

Proof. We consider a simplified instance 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 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 in Proposition 3 for the MapReduce(FCFS) problem, for the total weighted completion time of the jobs $1, 2, \ldots, n-1$ it holds that

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

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 MapReduce problem we have that

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

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

On the other hand, in an optimal schedule for the MapReduce(SR) problem, the jobs are scheduled starting from r according to the SR 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 in Proposition 3 for the MapReduce(FCFS) problem, we have that $OPT_{SR} = nr + O(n^3)$.

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

4.3 Experimental Evaluation of Scheduling Policies

In this section, our goal is to compare the FCFS and SR policies with respect to the quality of the solution that they produce.

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 [5], 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 in $[1, 10]$, while the work of each Reduce task $v_{i,j} \in \mathcal{R}$ is equal to a random number in $[1, 10]$ plus $\frac{3\sum_{T_{i',j} \in \mathcal{M}} v_{i',j}}{|\{T_{i',j} \in \mathcal{M}\}|}$, 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 in $[1, 10]$. For the release date of a job, we select with probability $1/2$ every interval $(t, t+1]$. Then, the release date is equal to a random value in this interval. The energy budget that we used is $E = 1000$. We have also set $\beta = 2$. We 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 as well as the code we used in our experiments are freely available at
`http://www.ibisc.univ-evry.fr/~vchau/research/mapreduce/`.

As mentioned before, the (CP) does not lead to a feasible solution for our problem. In order to get such a solution we apply the following algorithm. At each time t where a processor becomes available we select to schedule the task $T_{i,j}$ of higher priority such that: (i) $T_{i,j}$ is already released at t, (ii) if $T_{i,j}$ is a Reduce task, then all Map tasks of the same job have been already completed at t, and (iii) $T_{i,j}$ has not been yet executed.

As shown in Fig. 2 the heuristic based on FCFS outperforms the heuristic based on SR. In fact, the first heuristic gives up to $16-21\%$ better solutions that 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 SR leads to $26\% - 43\%$ smaller values of the objective function with respect 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 MapReduce(σ) problem given by the convex program is equal to 1.46 for FCFS and 2.43 for SR; the variance is less than 0.1 in both cases. However, as we already mentioned, this ratio cannot be considered as the approximation ratio for the MapReduce problem, as its optimal solution can be significantly smaller than the optimal solution for the MapReduce(σ) problem using the FCFS and SR orders.

5 Conclusions

We presented a constant-approximation algorithm for the problem of scheduling a set of MapReduce jobs in order to minimize their total weighted completion time under a given budget of energy. Our algorithm uses an optimal solution to an LP relaxation in interval-indexed variables and converts it to a feasible non-preemptive schedule of the MapReduce problem using the idea of list scheduling in order of a-points. Moreover, we proposed a convex programming relaxation of the problem when a prespecified order of jobs is given. Based on the solution of this convex programming relaxation, we explored the efficiency of standard scheduling policies, by presenting counterexamples for them as well as by experimentally evaluating their performance. It has to be noticed that our results can be extended also to the case where multiple Map or Reduce tasks of a job are executed on the same processor. An interesting direction for future work concerns the online case of the problem. Although, it can be proved that there is no an $O(1)$-competitive deterministic algorithm (see Theorem 13 in [3]), a possible way to overcome this is to consider resource (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.

References

1. S. Albers. Algorithms for dynamic speed scaling. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 1–11, 2011.

2. E. Angel, E. Bampis, and F. Kacem. Energy aware scheduling for unrelated parallel machines. In *Green Computing Conference*, pages 533–540, 2012.
3. N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2009.
4. H. Chang, M. S. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee. Scheduling in mapreduce-like systems for fast completion time. In *IEEE Proceedings of the 30th International Conference on Computer Communications*, pages 3074–3082, 2011.
5. F. Chen, M. 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*, pages 1143–1151, 2012.
6. E. Feller, L. Ramakrishnan, and C. Morin. On the performance and energy efficiency of Hadoop deployment models. In *BigData Conference*, pages 131–136, 2013.
7. B. Feng, J. Lu, Y. Zhou, and N. Yang. Energy efficiency for MapReduce workloads: An in-depth study. In *ADC*, pages 61–70, 2012.
8. D. Fotakis, I. Milis, E. Zampetakis, and G. Zois. Scheduling mapreduce jobs on unrelated processors. *Technical Report, arxv.org*, abs/1312.4203, 2013.
9. I. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. GreenHadoop: leveraging green energy in data-processing frameworks. In *EuroSys*, pages 57–70, 2012.
10. L. A. Hall, A.S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.
11. L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 142–151, 1996.
12. M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010.
13. N. Megow and J. Verschae. Dual techniques for scheduling on a machine with varying speed. In *Proceedings of the 40th ICALP*, pages 745–756, 2013.
14. B. Moseley, A. Dasgupta, R. Kumar, and T. 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.
15. A. Nemirovski, I. Nesterov, and Y. Nesterov. *Interior Point Polynomial Algorithms in Convex Programming.* Society for Industrial and Applied Mathematics, 1994.
16. C. A. Phillips, C. Stein, and J. Wein. Task scheduling in networks. *SIAM Journal on Discrete Mathematics*, 10(4):573–598, 1997.
17. T.A. Roemer. A note on the complexity of the concurrent open shop problem. *Journal of Scheduling*, 9:389–396, 2006.
18. A. S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. *SIAM J. Discrete Math.*, 15(4):450–469, 2002.
19. Martin Skutella. List scheduling in order of -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*, page 250291. Springer, 2006.
20. T. Wirtz and R. Ge. Improving MapReduce energy efficiency for computation intensive workloads. In *IGCC*, pages 1–8, 2011.

21. F. F. Yao, A. J. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 374–382. IEEE, 1995.