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 has 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 [6–8]. 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 [18]. Until lately, most work in the MapReduce framework were focused on the power-down mechanism, but recently, Wirtz and Ge [17] showed

* This work was partially supported by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning", under the programs THALES-ALGONOW (E. Bampis, G. Lucarelli, I. Milis) and HERACLEITUS II (G. Zois), and the project "Mathematical Programming and Non-linear Combinatorial Optimization" under the program PGMO (E. Bampis, V. Chau, G. Lucarelli).

F. Silva et al. (Eds.): Euro-Par 2014, LNCS 8632, pp. 198–209, 2014.
© Springer International Publishing Switzerland 2014

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 [18]. 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 [12]. The scheduler has to decide the time interval and the speed over time at which a task is executed, taking into account the energy consumption. High processor's speeds are in favor of performance at the price of high energy consumption. 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 [10] and the references therein). Extending on the above-mentioned 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. [12], 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.

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. Moreover, Megow et al. [11] 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, 12] in the speed scaling setting [18]. In Section 2, we present formally our problem and we introduce our notation. In Section 3, we present a constant-factor approximation algorithm. Using discretization of the possible speed values we give an interval indexed LP relaxation of our problem and we transform an optimal solution to this LP to a feasible solution for our problem by list scheduling in the order of tasks' α-points (see e.g. [9, 13]). This leads to a $O(1)$-energy $O(1)$-approximation algorithm, that is an algorithm that may use energy augmentation. More specifically, we call a schedule 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. [14]). Our algorithm describes a tradeoff between the approximation ratio and the energy augmentation as a function of α. By appropriately choosing α, our result becomes a constant-factor approximation for our problem. In Section 4, 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. For that, we propose a convex programming relaxation of our problem, for a prespecified order of jobs, which can be solved in polynomial time by the Ellipsoid algorithm. Then we combine the solution of this relaxation with FCFS and HDF and we compare experimentally their effectiveness.

2 Problem Statement 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 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 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 [18] (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 $\beta > 1$ (typically, $\beta \in [2, 3]$) 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.

All omitted proofs can be found in the full version of this work, available at `http://arxiv.org/abs/1402.2810`.

3 A Linear Programming Approach

In this section we present an $O(1)$-energy $O(1)$-approximation algorithm for the 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 MapReduce problem using the technique of α-points. Note that, by allowing energy augmentation we are able to describe a tradeoff between energy and performance. Moreover, we can derive a constant-factor approximation ratio (without energy augmentation) for the MapReduce problem by appropriately choosing some parameters.

3.1 Linear Programming Relaxation

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. An upper bound of $\left(\frac{E}{v_{\min}}\right)^{\frac{1}{\beta-1}}$ is easily obtained since the energy consumption of any task can not exceed the energy budget. A lower bound on the speed values is $\frac{v_{\min}}{\mathcal{C}}$, where $\mathcal{C}$ is an upper bound to the makespan of any optimal schedule; $\mathcal{C}$ can be computed by considering all jobs executed after the maximum release date. Then, by loosing a factor of $(1+\epsilon)$ with respect to an optimal solution, we can prove the following.

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* $s \in \mathcal{V}$*, where* $\mathcal{V}$ *is the set of all possible discrete speed values and* $\epsilon \in (0,1)$*.*

Next, we discretize the time horizon $(0, \mathcal{C}]$ of an optimal schedule by partitioning it 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 $\lambda(1+\delta)^{u-1} \geq \mathcal{C}$. 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 $u = \lceil \log_{1+\delta} \frac{\mathcal{C}}{\lambda} \rceil + 1$.

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) in the next page, is a linear programming relaxation of the problem where each task $T_{i,j} \in \mathcal{T}$ runs at a single speed $s \in \mathcal{V}$.

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 [16], 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

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

subject to :

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

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

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

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

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

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

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

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

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 $(\bar{y}_{i,j,s,t}, \bar{C}_{i,j}, \bar{C}_j)$.

3.2 The Algorithm

In this section we use (LP) to derive a feasible schedule for the MapReduce problem. Our algorithm is based on the idea of list scheduling in order of α-points [9, 13]. 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 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

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

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. 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 $\mathcal{MR}(\alpha, \gamma)$ gives a formal description of our method.

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 α-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 σ_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 σ_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 $C_j = \max_{i \in \mathcal{P}} C_{i,j}$.

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. The next lemma gives a relation between these two quantities.

Lemma 2. *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 $\mathcal{MR}(\alpha, \gamma)$*, respectively. It holds that $E_{i,j} \leq \frac{1}{\gamma^{\beta-1}\alpha^{\beta}} \bar{E}_{i,j}$.*

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

Using Lemmas 2 and 3 as well as Lemma 1 we can prove the following theorem.

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

By choosing $\gamma = \frac{1}{\alpha^{\beta-1}\sqrt{\alpha}}$, no energy augmentation is used and ALGORITHM $\mathcal{MR}(\alpha, \gamma)$ becomes a constant-factor approximation for the MAPREDUCE problem, and the following theorem holds.

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

In Fig.1 we depict the tradeoff between energy augmentation and approximation ratio for some practical values of β.

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 [4]) our results are improved as follows.

Corollary 1. *There is a* $\frac{\alpha^{\beta-1}\sqrt{\alpha}+1}{\alpha^{\beta-1}\sqrt{\alpha}(1-\alpha)}(1+\varepsilon)$*-approximation algorithm for the* MAPREDUCE *problem without precedence constraints between Map and Reduce tasks, and a* $\frac{1}{\alpha^{\beta-1}\sqrt{\alpha}(1-\alpha)}(1+\varepsilon)$*-approximation algorithm for the* 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 1 gives the achieved ratios for practical values of β.

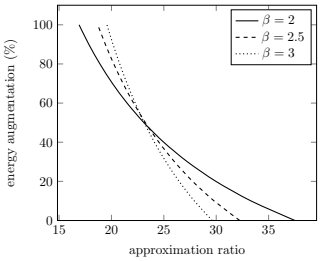

Fig. 1. Tradeoff between energy augmentation and approximation ratio when $\beta = \{2, 2.5, 3\}$

Table 1. Approximation ratios for the MAPREDUCE problem for different 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

4 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. 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, 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) is a convex programming relaxation of the MapReduce(σ) problem.

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

subject to :

$$\sum_{T_{i,j} \in \mathcal{T}} \frac{v_{i,j}^{\beta}}{p_{i,j}^{\beta-1}} \leq E \tag{9}$$

$$r_{j'} + \sum_{k=j'}^{j} p_{i,k} \leq C_{i,j}, \qquad \forall T_{i,j}, T_{i,j'} \in \mathcal{T}, j' \prec j \tag{10}$$

$$C_{i',j} + p_{i,j} \leq C_{i,j}, \qquad \forall T_{i,j} \in \mathcal{R}, T_{i',j} \in \mathcal{M} \tag{11}$$

$$C_{i,j} \leq C_j, \qquad \forall T_{i,j} \in \mathcal{T} \tag{12}$$

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

The objective function of (CP) is to minimize the weighted completion time of all jobs. Constraint (9) 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 (10) and (11) 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 on

it, as constraints (10) describe in a complete way the completion times of the tasks. However, this is not true for constraints (11) which are responsible for the precedence constraints. Finally, constraints (12) 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. It suffices to apply, the lines 6-8 of Algorithm $\mathcal{MR}(\alpha, \gamma)$, by considering the same order for all processors.

4.2 Experimental Evaluation of Scheduling Policies

We propose different orders of jobs and discuss how far is an optimal solution for the MapReduce(σ) problem using these orders with respect to an optimal solution for the MapReduce problem. Consider the following standard orders.

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.

The following proposition gives negative results concerning the approximation ratio that we can achieve if we use the FCFS or the HDF order.

Proposition 1. *There are instances for which the optimal solutions to the* MapReduce(FCFS) *and the* MapReduce(HDF) *problems are within a factor of* $\Omega(n)$ *from the optimal solution to the* MapReduce *problem.*

In what follows we compare the FCFS and HDF policies with respect to the quality of the solution 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 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 $\frac{3 \sum_{T_{i',j} \in \mathcal{M}} v_{i',j}}{|\{T_{i',j} \in \mathcal{M}\}|}$, 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/`.

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 must have been already completed at t, and (iii) $T_{i,j}$ has not been yet executed.

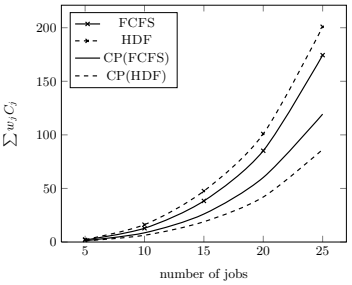

Fig. 2. Comparing solutions for FCFS and HDF (scaled down by a factor of 10^3)

As shown in Fig. 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 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.

5 Conclusions

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 presented counterexamples for them, as well as, 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 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. However, 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. Albers, S.: Algorithms for dynamic speed scaling. In: STACS, pp. 1–11 (2011)
2. Angel, E., Bampis, E., Kacem, F.: Energy aware scheduling for unrelated parallel machines. In: Green Computing Conference, pp. 533–540 (2012)
3. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. SIAM J. on Computing 39(4), 1294–1308 (2009)
4. Chang, H., Kodialam, M.S., Kompella, R.R., Lakshman, T.V., Lee, M., Mukherjee, S.: Scheduling in mapreduce-like systems for fast completion time. In: INFOCOM, pp. 3074–3082 (2011)
5. Chen, F., Kodialam, M.S., Lakshman, T.V.: Joint scheduling of processing and shuffle phases in mapreduce systems. In: INFOCOM, pp. 1143–1151 (2012)
6. Feller, E., Ramakrishnan, L., Morin, C.: On the performance and energy efficiency of Hadoop deployment models. In: BigData Conference, pp. 131–136 (2013)
7. Feng, B., Lu, J., Zhou, Y., Yang, N.: Energy efficiency for MapReduce workloads: An in-depth study. In: ADC, pp. 61–70 (2012)
8. Goiri, I., Le, K., Nguyen, T.D., Guitart, J., Torres, J., Bianchini, R.: GreenHadoop: leveraging green energy in data-processing frameworks. In: EuroSys, pp. 57–70 (2012)
9. Hall, L.A., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: Off-line and on-line algorithms. In: ACM-SIAM SODA, pp. 142–151 (1996)
10. Mastrolilli, M., Queyranne, M., Schulz, A.S., Svensson, O., Uhan, N.A.: Minimizing the sum of weighted completion times in a concurrent open shop. Oper. Res. Letters 38(5), 390–395 (2010)
11. Megow, N., Verschae, J.: Dual techniques for scheduling on a machine with varying speed. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 745–756. Springer, Heidelberg (2013)
12. Moseley, B., Dasgupta, A., Kumar, R., Sarlós, T.: On scheduling in map-reduce and flow-shops. In: ACM-SPAA, pp. 289–298 (2011)
13. Phillips, C.A., Stein, C., Wein, J.: Minimizing average completion time in the presence of release dates. Math. Programming 82(1-2), 199–223 (1998)
14. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. Theory Comput. Syst. 43, 67–80 (2008)
15. Roemer, T.A.: A note on the complexity of the concurrent open shop problem. Journal of Scheduling 9, 389–396 (2006)
16. Schulz, A.S., Skutella, M.: Scheduling unrelated machines by randomized rounding. SIAM J. Discr. Mathematics 15(4), 450–469 (2002)
17. Wirtz, T., Ge, R.: Improving MapReduce energy efficiency for computation intensive workloads. In: IGCC, pp. 1–8 (2011)
18. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced cpu energy. In: IEEE- FOCS, pp. 374–382 (1995)