Scheduling MapReduce Jobs Under Multi-round Precedences

D. Fotakis[1], I. Milis[2], O. Papadigenopoulos[1], V. Vassalos[2], and G. Zois[2](✉)

[1] School of Electrical and Computer Engineering,
National Technical University of Athens, Athens, Greece
fotakis@cs.ntua.gr, opapadig@corelab.ntua.gr

[2] Department of Informatics,
Athens University of Economics and Business, Athens, Greece
{milis,vassalos,georzois}@aueb.gr

Abstract. We consider non-preemptive scheduling of MapReduce jobs consisitng of multiple map-reduce rounds so as to minimize the average weighted completion time on identical and unrelated processors. For identical processors, we present LP-based $O(1)$-approximation algorithms, while for unrelated processors the approximation ratio naturally depends on the maximum number of rounds of any job (a small constant in practice). For the single-round case, we substantially improve on previously best known approximation ratios for both identical and unrelated processors. Moreover, we conduct an experimental analysis and compare the performance of our algorithms against a fast heuristic and a lower bound on the optimal solution, thus demonstrating their promising practical performance.

1 Introduction

The sharp rise in Internet's use has boosted the amount of data stored on the web and processed daily. MapReduce [6], and its open-source implementation Hadoop, is a fundamental platform for processing data sets on large clusters. A MapReduce job starts by allocating (randomly or arbitrarily) data to a set of processors. The computation over the dataset is broken into successive rounds, where, during each round, a two-phase (map-reduce) process is executed, in which the execution of any reduce task cannot begin until all of its corresponding map tasks have finished. A key observation is that, while the map and reduce phases in each round must be executed sequentially, the tasks in each phase can be executed in parallel. In addition to the computation cost of map and reduce phases, a significant cost is the *communication cost* of transmitting the intermediate data of a job from each map task to every reduce task. Although

I. Milis was partially supported by the Research Center of Athens University of Economics and Business (RC-AUEB). V. Vassalos and G. Zois were supported by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 604102 (Human Brain Project).

© Springer International Publishing Switzerland 2016
P.-F. Dutot and D. Trystram (Eds.): Euro-Par 2016, LNCS 9833, pp. 209–222, 2016.
DOI: 10.1007/978-3-319-43659-3_16

MapReduce is a distributed computation model, the scheduler of such a system is operating in a centralized manner and its performance is crucial for the efficiency of large MapReduce clusters shared by many users. These clusters typically deal with many jobs that consist of many tasks and of several map-reduce rounds. In such processing environments, the quality of a schedule is typically measured by the jobs' average completion time, which for a MapReduce job takes into account the time when the last reduce task finishes its work.

In this work, we present a general model and an algorithmic framework for scheduling a set of MapReduce jobs on parallel (identical or unrelated) processors with the goal to minimize their average weighted completion time. We consider an offline setting for our model, where each job is represented by multiple successive rounds, and each round consists of multiple map and reduce tasks corresponding to the map and reduce phases respectively. Each reduce task cannot begin its execution before all map tasks of the same round are finished, while the same also holds between reduce and map tasks of two successive rounds. Moreover, the tasks are associated with positive processing times, depending on the processor environment, and each job has a positive weight to represent its priority value. Concerning the communication cost that is incurred in each round, we assume that it is incorporated in the processing times of its reduce task.

Related Work. In the distributed setting of MapReduce's architecture, two main models have been proposed for analyzing the efficiency of MapReduce algorithms with respect to the number of rounds required. Karloff et al. [12] presented a model inspired by PRAM and proved that a large class of PRAM algorithms can be efficiently (i.e., the number of processors and their memory should be sublinear and the running time in each round should be polynomial in the input size) implemented in MapReduce. Recent results in this direction [13] have proposed substantial improvements on the number of rounds for various MapReduce algorithms. Afrati et al. [1] proposed a different model that is inspired by BSP and focuses on the trade-off between communication and computation cost. The main idea is that restricting the computation cost leads to a greater amount of parallelism and to a larger communication cost between the mappers and the reducers. In this context, [2] presents multi-round MapReduce algorithms, trying to optimize the tradeoff between the communication cost and the number of rounds.

In the context of MapReduce scheduling a significant volume of work focuses on the experimental evaluation of scheduling heuristics, trying to achieve good tradeoffs between various criteria (see e.g., [19]). On the other hand theoretical work (e.g., [4,8,15]) focuses on scheduling a set of MapReduce jobs on parallel processors to minimize the average (weighted) completion time, capturing the main practical insights in a MapReduce computation (e.g., task dependencies, data locality), in the restricted case where each job is executed in a single round. [4] presents approximation algorithms using simple models, equivalent to known variants of the open-shop problem, taking into account task precedences and assuming that the tasks are preassigned to processors. Moseley et al. [15] present a 12-approximation algorithm for the case of identical processors, modeling in this way MapReduce scheduling as a generalization of the so-called two-stage Flexible Flow-Shop problem. They also present a $O(1/\epsilon^2)$-competitive online algorithm, for any $\epsilon \in (0, 1)$, under $(1+\epsilon)$-speed augmentation. [8] studies the *single-round MapReduce scheduling* problem in the most general case of unrelated processors and present an LP-based 54-approximation algorithm. They also show how to incorporate the communication cost into their algorithm, with the same approximation ratio.

Contribution. Our model incorporates all the main features of the models in [1,12], aiming at an efficient scheduling and assignment of tasks in MapReduce environments. Note that, by assuming positive values for the tasks' execution times, which are polynomially bounded by the input size, we are consistent with both computation models [1,12]. We refer to our problem as the *multi-round MapReduce scheduling* problem or the *single-round MapReduce scheduling* problem (depending on the number of rounds). Our contribution is threefold. First, in terms of modeling the MapReduce scheduling process: (i) We consider the practical scenario of multi-round multi-task MapReduce jobs and capture their task dependencies, and (ii) we study both identical and unrelated processors, thus dealing with data locality. Second, in terms of algorithm design and analysis: (i) We propose an algorithmic framework for the *multi-round MapReduce scheduling* problem with proven performance guarantees, distinguishing between the case of indistinguishable and disjoint (map and reduce) sets of identical or unrelated processors, and (ii) our algorithms are based on natural LP relaxations of the problem and improve on the approximation ratios achieved in previous work [8,15]. Third, in terms of experimental analysis, we focus on the most general case of unrelated processors and show that our algorithms have an excellent performance in practice.

The rest of the paper is organized as follows. In Sect. 2, we formally define our model and provide notation. In Sect. 3, we consider the *multi-round MapReduce scheduling* problem on identical indistinguishable and disjoint processors and we design a 4-approximation and an 11-approximation algorithm, respectively. Moreover, for the *single-round MapReduce scheduling* problem on identical disjoint processors we substantially improve on the results proposed by Moseley et al. [15], presenting an LP-based 8-approximation algorithm, instead of 12-approximation. In Sect. 4, we consider the *multi-round MapReduce scheduling* problem on the most general environment of unrelated processors and we propose an LP-based $O(r_{\max})$-approximation algorithm, where $r_{\max}$ is the maximum number of rounds over all jobs. As a corollary, for the *single-round MapReduce scheduling* problem, we show a 37.87-approximation, which significantly improves on the previously proposed 54-approximation algorithm in [8]. Furthermore, we comment on the hardness of the *multi-round MapReduce scheduling* problem. In Sect. 5, we compare our algorithms via simulations of random instances with a fast heuristic, proposed in [8], as well as with a lower bound on the optimal value of the *multi-round MapReduce scheduling* problem.

2 Problem Formulation

We consider a set $\mathcal{J} = \{1, 2, \ldots, n\}$ of n *MapReduce jobs* to be scheduled on a set $\mathcal{P} = \{1, 2, \ldots, m\}$ of m parallel *processors*. Each job $j \in \mathcal{J}$ is available at time zero and comprises of $r_j \in \mathbb{N}$, $r_j \geq 1$ rounds of computation, with each round consisting of a set of map tasks and a set of reduce tasks. Moreover, each job is associated with a positive weight, let w_j, indicating its significance and, therefore, its relative *priority* to the system. Let $\mathcal{M}$, $\mathcal{R}$ be the sets of all *map* and *reduce* tasks respectively. Each task $T_{k,j} \in \mathcal{M} \cup \mathcal{R}$ of a job $j \in \mathcal{J}$, where $k \in \mathbb{N}$, is associated with a positive processing time. Note that, by assuming task processing times that are polynomially bounded by the input size we are consistent with the two above computation models [1,12]. In every round, each reduce task of a job can start its execution only after the completion of all map tasks of the same job, while similar precedence constraints hold also between the reduce and the map tasks of two successive rounds. In other words, except for the precedence constraints emerged by the existence of map and reduce phases, there are also precedence constraints between consecutive rounds, so a map task of a round $r \in \{2, \ldots r_j\}$, of a job j, cannot start its execution unless all the reduce tasks of the previous round, $r - 1$, have completed their execution. The precedence constraints of a *multi-round MapReduce job* j can be represented by an r_j*-partite-like* directed acyclic graph, as the one depicted in Fig. 1, where r_j is the number of rounds and $l_j = 2r_j - 1$ is the length of a maximal path of the tasks' precedences. Throughout the analysis, in order to upper bound the approximation ratio of our algorithms, the latter parameter is used instead of the number of rounds. Note that, in order to refer to a precedence constraint between two tasks, we use the standard notation, $T_{k,j} \prec T_{k',j}$.

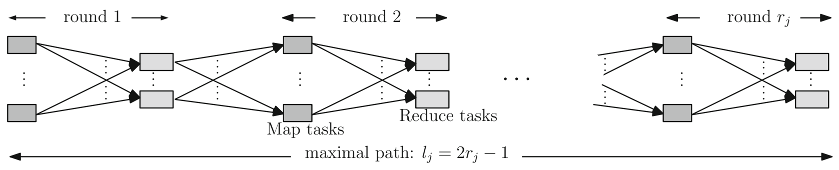

Fig. 1. A MapReduce job j of r_j rounds, and length $l_j = 2r_j - 1$.

To better capture data locality issues in task assignment, we distinguish between the standard *identical processors* environment, where the processing time of each task $T_{k,j}$, let $p_{k,j}$, is the same for every processor, and the most general *unrelated processors* environment, where there is a vector of processing times $\{p_{i,k,j}\}$, one for each processor $i \in \mathcal{P}$. Concerning the dedication of processors to either map or reduce tasks, we examine two cases: (a) The sets $\mathcal{P}_{\mathcal{M}}$ and $\mathcal{P}_{\mathcal{R}}$ are *indistinguishable* and the processors in $\mathcal{P}$ are processing both map and reduce tasks, and (b) the set $\mathcal{P}$ is divided into two *disjoint* sets $\mathcal{P}_{\mathcal{M}}$ and $\mathcal{P}_{\mathcal{R}}$, where $\mathcal{P} = \mathcal{P}_{\mathcal{M}} \cup \mathcal{P}_{\mathcal{R}}$, where the processors of $\mathcal{P}_{\mathcal{M}}$ process only map tasks, while the processors of $\mathcal{P}_{\mathcal{R}}$ process only reduce tasks.

For a given schedule we denote by C_j and $C_{k,j}$ the completion times of a job $j \in \mathcal{J}$ and a task $T_{k,j} \in \mathcal{M} \cup \mathcal{R}$ respectively. Note that, due to the task precedences along the r_j rounds of each job j, $C_j = \max_{T_{k,j} \in \mathcal{R}}\{C_{k,j}\}$. By $C_{max} = \max_{j \in \mathcal{J}}\{C_j\}$ we denote the *makespan* of the schedule, i.e. the completion time of the last finishing job. Our goal is to schedule *non-preemptively* all tasks on processors of $\mathcal{P}$, with respect to their precedences, so as to minimize the average weighted completion time, $\sum_{j \in \mathcal{J}} w_j C_j$.

3 Scheduling Tasks on Identical Processors

We first study the case of *multi-round MapReduce scheduling* on identical indistinguishable or disjoint processors. For indistinguishable processors, reducing the problem to standard job scheduling under precedence constraints, we immediately obtain a 4-approximation algorithm, a result that holds also for *single-round MapReduce scheduling*. Then, we present an 11-approximation algorithm for identical disjoint processors. For the same case, we also propose an improved 8-approximation algorithm for the *single-round MapReduce scheduling* problem, which substantially improves on the 12-approximation algorithm proposed in [15] for the same problem.

Indistinguishable Processors. We consider the *multi-round MapReduce scheduling* problem on identical indistinguishable processors. Finding an algorithm for this problem can be easily reduced to finding an algorithm for the classic problem of scheduling a set of jobs on identical processors, under precedence constraints of any kind, to minimize their average weighted completion time. More specifically, for any instance of our problem we can create an equivalent instance of the latter problem through the following transformation: For every task $T_{k,j} \in \mathcal{M} \cup \mathcal{R}$, we create a corresponding job j_k of equal processing time, $p_{k,j}$, and zero weight, $w_{j_k} = 0$. We maintain the same precedence constraints, emerged from the input of *multi-round MapReduce scheduling* problem, to the new problem, i.e. for every $T_{k,j} \succ T_{k',j}$ we set $j_k \succ j_{k'}$. For each MapReduce job j, we create a *dummy* job j_D of zero processing time and weight equal to the weight of j, i.e. $w_{j_D} = w_j$, and for every job j_k we demand that $j_k \succ j_D$. In other words, since the corresponding dummy task of a MapReduce job j has zero processing time, there exists an optimal schedule where it is executed exactly after the completion time of all corresponding jobs j_k and, therefore, indicate the completion time of the job itself in the MapReduce context. Moreover, every dummy job j_D carries the weight of the corresponding MapReduce job j. [16] shows a 4-approximation algorithm for scheduling a set of jobs on identical processors, under general precedence constraints, to minimize their average weighted completion time. Combining our transformation with this algorithm, we obtain that:

Theorem 1. *There is a 4-approximation algorithm for the* multi-round MapReduce scheduling *problem on identical indistinguishable processors.*

Disjoint Processors. Inspired by the algorithm of [10, Theorem 3.8], we present an $O(1)$-approximation algorithm which transforms a solution to an interval-indexed LP relaxation of our problem into an integral schedule by carefully applying, on each interval of execution, a variation of the well-known Graham's 2-approximation algorithm [9] for job scheduling on identical processors under precedence constraints to minimize makespan. Note that, in the following, we use the term $b \in \{\mathcal{M}, \mathcal{R}\}$ to refer to both map and reduce attributes.

For any set of tasks $S \subseteq b$, we define $p(S) = \sum_{T_{k,j} \in S} p_{k,j}$ and $p^2(S) = \sum_{T_{k,j} \in S} p_{k,j}^2$. The following (LP1) is an interval-indexed linear programming relaxation of our problem. Constraints (1) ensure that the completion time of a MapReduce job is at least the completion time of any of its tasks and that the completion time of any task is at least its processing time. Constraints (2) capture the relation of completion times of two tasks $T_{k,j} \succ T_{k',j}$. Constraints (3) have been proved [10] to hold for any feasible schedule on identical processors minimizing the average weighted completion time and give useful lower bounds to the completion times of tasks.

Let $(0, t_{\max} = \sum_{T_{k,j} \in \mathcal{M} \cup \mathcal{R}} p_{k,j}]$ be the time horizon of the schedule, where $t_{\max}$ is an upper bound on the makespan of any feasible schedule. We discretize the time horizon into intervals $[1,1], (1,2], (2,2^2], \ldots, (2^{L-1}, 2^L]$, where L is the smallest integer such that $2^{L-1} \geq t_{\max}$. Let $\mathcal{L} = \{1, 2, \ldots, L\}$. Note that, interval $[1,1]$ implies that no job finishes its execution before time 1; in fact, we can assume, w.l.o.g., that all processing times are positive integers. Let $\tau_0 = 1$ and $\tau_\ell = 2^{\ell-1}$. Our algorithm begins from a fractional solution to the LP, $(\bar{C}_{k,j}, \bar{C}_j)$, and separates tasks into intervals with respect to their completion times $\bar{C}_{k,j}$ as follows.

$$\textbf{(LP1)}: \quad \text{minimize} \sum_{j \in \mathcal{J}} w_j C_j$$

$$\text{s.t.} \quad C_j \geq C_{k,j} \geq p_{k,j} \qquad \forall T_{k,j} \in \mathcal{M} \cup \mathcal{R} \tag{1}$$

$$C_{k,j} \geq C_{k',j} + p_{k,j} \qquad \forall T_{k',j} \prec T_{k,j} \tag{2}$$

$$\sum_{T_{k,j} \in b} p_{k,j} C_{k,j} \geq \frac{p(S)^2 + p^2(S)}{2|\mathcal{P}_b|} \qquad b \in \{\mathcal{M}, \mathcal{R}\}, \forall S \subseteq b \tag{3}$$

Let $S(\ell) = \{T_{k,j} | \tau_{\ell-1} < \bar{C}_{k,j} \leq \tau_\ell\}$. Let also $S^M(\ell) \subseteq S(\ell)$ and $S^R(\ell) \subseteq S(\ell)$ be a partition of each set $S(\ell)$ into only map and only reduce tasks, respectively. We define $t_\ell^M = \frac{p(S^M(\ell))}{|\mathcal{P}_M|}$ and $t_\ell^R = \frac{p(S^R(\ell))}{|\mathcal{P}_R|}$ to be the average load of a map and reduce processor, respectively, for executing the map and reduce tasks of each set $S(\ell)$. Now, we can define an adjusted set of intervals as $\bar{\tau}_\ell = 1 + \sum_{k=1}^{\ell} (\tau_k + t_k^M + t_k^R) \quad \forall \ell \in \mathcal{L}$. We can schedule greedily the tasks of each set $S(\ell)$ in interval $(\bar{\tau}_{\ell-1}, \bar{\tau}_\ell]$, using the following variation of Graham's List Scheduling algorithm.

Restricted-Resource List Scheduling. Consider two different types of available resources, i.e. the map and the reduce processors, while each task can be scheduled only on a specific resource type. Whenever a processor becomes available, execute on it any available unscheduled task that corresponds to its type.

Lemma 1. *The tasks of $S(\ell)$ can be scheduled non-preemptively at interval $(\bar{\tau}_{\ell-1}, \bar{\tau}_{\ell}]$ by applying* Restricted-Resource List Scheduling.

Proof. Using the analysis of [10] we can prove that the makespan of each set $S(\ell)$ is upper bounded by the total processing time of the longest chain of precedences and the average processing time of a map (resp. reduce) processor. By definition of $S(\ell)$ and constraints (1), we know that the former value can be at most τ_ℓ. Therefore, if the algorithm starts by assigning tasks at time $\bar{\tau}_{\ell-1}$, it should have finished by time $C \leq \bar{\tau}_{\ell-1} + \tau_\ell + t_\ell^M + t_\ell^R$. Then, by definition of $\bar{\tau}_{\ell-1}$ we have that $C \leq 1 + \sum_{k=1}^{\ell-1}(\tau_k + t_k^M + t_k^R) + \tau_\ell + t_\ell^M + t_\ell^R \leq 1 + \sum_{k=1}^{\ell}(\tau_k + t_k^M + t_k^R) = \bar{\tau}_\ell$. □

Note that the resulting schedule respects the tasks precedences since by (1), for any pair of tasks such that $T_{k,j} \succ T_{k',j}$, it must be the case that $T_{k,j} \in S(\ell)$ and $T_{k',j} \in S(\ell')$ with $\ell \leq \ell'$. Now we are able to prove the following theorem.

Theorem 2. *There is an 11-approximation algorithm for the* multi-round MapReduce scheduling *problem on identical disjoint processors.*

Proof. Consider the completion time $C_{k,j}$ of a task $T_{k,j} \in S(\ell)$. By constraints (1) and (2) we know that the length of any chain that ends with $T_{k,j}$ is upper bounded by $\bar{C}_{k,j}$. Therefore, using the previous lemma and since $T_{k,j} \in S(\ell)$, we can see that for its completion time it holds: $C_{k,j} \leq \bar{\tau}_{\ell-1} + t_\ell^M + t_\ell^R + \bar{C}_{k,j} = 1 + \sum_{k=1}^{\ell-1}(\tau_k + t_k^M + t_k^R) + t_\ell^M + t_\ell^R + \bar{C}_{k,j} = \tau_\ell + \sum_{k=1}^{\ell}(t_k^M + t_k^R) + \bar{C}_{k,j}$. Constraints (3) imply that, for the last finishing -say map- task, $T_{k',j'}$ of the set $S(\ell')$, it holds $\bar{C}_{k',j'} \geq \frac{1}{2|\mathcal{P}_M|}\sum_{k=1}^{\ell'} p(S(k))$, while the same holds for the reduce tasks. Therefore: $\sum_{k=1}^{\ell}(t_k^M + t_k^R) = \sum_{k=1}^{\ell} t_k^M + \sum_{k=1}^{\ell} t_k^R \leq \frac{1}{|\mathcal{P}_M|}\sum_{k=1}^{\ell} p(S^M(k)) + \frac{1}{|\mathcal{P}_R|}\sum_{k=1}^{\ell} p(S^R(k)) \leq 2\tau_\ell + 2\tau_\ell = 4\tau_\ell$. Since by definition of $S(\ell)$, $\tau_\ell \leq 2\bar{C}_{k,j}$ it is the case that: $C_{k,j} \leq \tau_\ell + 4\tau_\ell + \bar{C}_{k,j} \leq 11\bar{C}_{k,j}$. The theorem follows by applying the previous inequality to the objective function. □

Remark. A simple transformation of the previous algorithm yields a 7-approximation algorithm for indistinguishable processors. However, Theorem 1 also applies and gives a 4-approximation algorithm for the *single-round MapReduce scheduling* problem.

The Single-Round Case. For the special case of *single-round MapReduce scheduling*, we obtain an 8-approximation algorithm, improving on the 12-approximation algorithm of [15]. Our algorithm refines the idea of merging independent schedules of only map and only reduce tasks, σ_M and σ_R respectively, on their corresponding sets of processors into a single schedule, by applying a 2-approximation algorithm similar to that in [5, Lemma 6.1]. Note that [5] considers a more general case of scheduling a set of job orders, instead of jobs consisting of tasks, while the completion time of each order is specified by the completion of the job that finishes last.

$$\begin{aligned} \textbf{(LP2)}: \quad & \text{minimize} \sum_{j \in \mathcal{J}} w_j C_j \\ \text{s.t.} \quad & C_j \geq M_{k,j} + \frac{p_{k,j}}{2} & \forall T_{k,j} \in b \quad (4) \\ & \sum_{T_{k,j} \in S} p_{k,j} M_{k,j} \geq \frac{p(S)^2}{2|\mathcal{P}_b|} & \forall S \subseteq b \quad (5) \end{aligned}$$

For the partial schedules σ_b of only map and only reduce tasks, since we have no precedence constraints between tasks, let $M_{k,j}$ be the midpoint of a task $T_{k,j} \in b$ in any non-preemptive schedule, i.e., $M_{k,j} = C_{k,j} - \frac{p_{k,j}}{2}$. [7] shows that in any feasible schedule on m identical processors, for every $S \subseteq b$: $\sum_{T_{k,j} \in S} p_{k,j} M_{k,j} \geq \frac{p(S)^2}{2m}$.

Now, consider the linear programming formulation (LP2). Note that, although the number of inequalities of this linear program is exponential, it is known [17] that it can be solved in polynomial time using the ellipsoid algorithm. Thus, consider an optimal solution $(\bar{M}_{k,j}, \bar{C}_j)$ to this formulation with objective value $\sum_{j \in \mathcal{J}} w_j \bar{C}_j$. If we greedily assign tasks on the processors of $\mathcal{P}_b$ in a non-decreasing order of $\bar{M}_{k,j}$ using Graham's list scheduling, then, for the resulting schedule σ_b, it holds that:

Lemma 2. *There is a 2-approximate schedule of map (resp. reduce) tasks on identical map (resp. reduce) processors to minimize their average weighted completion time.*

The second step of our algorithm is to merge the two partial schedules σ_M and σ_R into a single one. To succeed it, we can use the merging technique proposed in [15]. If we denote by $C_j^{\sigma_M}$ and $C_j^{\sigma_R}$ the completion times of a job j in σ_M and σ_R respectively, we can define the *width* of each job j to be $\omega_j = \max\{C_j^{\sigma_M}, C_j^{\sigma_R}\}$. The algorithm schedules the tasks of each job on the same processors that they have been assigned in σ_M and σ_R, in non-decreasing order of ω_j, with respect to the precedences. This merging routine is known [8, Theorem 2] to result in a schedule where the completion time of each job is at most $2 \max\{C_j^{\sigma_M}, C_j^{\sigma_R}\}$, leading to the following theorem:

Theorem 3. *There is an 8-approximation algorithm for the* single-round MapReduce scheduling *problem on identical disjoint processors.*

Remark. The same analysis yields an 8-approximation algorithm for *single-round MapReduce scheduling* on identical indistinguishable processors. We only have to define the width of each job to be $\omega_j = C_j^{\sigma_M} + C_j^{\sigma_R}$.

4 Scheduling Tasks on Unrelated Processors

In this section, we consider the *multi-round MapReduce scheduling* problem on unrelated processors. We present a $\mathcal{O}(l_{\max})$-approximation algorithm, where $l_{\max} = \max_{j \in \mathcal{J}} l_j$ is the maximum length over all jobs' maximal paths in the underlying precedence graph. Since $l_{max} = 2r_{\max} - 1$, our algorithm is also a $O(r_{\max})$-approximation, where $r_{\max}$ is the maximum number of rounds over all jobs. Our technique builds on ideas proposed in [8]. We formulate an interval-indexed LP relaxation for *multi-round MapReduce scheduling* so as to handle the multi-round precedences. Unlike [8,15], we avoid the idea of creating partial schedules of only map and only reduce tasks and then combining them into one. Moreover, applying the following algorithm for the *single-round MapReduce scheduling* problem, we derive a 37.87-approximation algorithm, thus improving on the 54-approximation algorithm of [8]. Even though in the following analysis, we consider the case of indistinguishable processors, we can simulate the case of disjoint processors by simply setting $p_{i,k,j} = +\infty$ for every map (resp. reduce) task $T_{k,j}$ when i is a reduce (resp. map) processor. In the sequel, we denote by $\mathcal{T} = \mathcal{M} \cup \mathcal{R}$ the set of all tasks.

We use an interval-indexed LP relaxation. Let $(0, t_{\max} = \sum_{T_{k,j} \in \mathcal{T}} \max_{i \in \mathcal{P}} p_{i,k,j}]$ be the time horizon of potential completion times, where $t_{\max}$ is an upper bound on the makespan of any feasible schedule. Similarly with (LP1), we discretize the time horizon into intervals $[1,1], (1,(1+\delta)], ((1+\delta),(1+\delta)^2], \ldots, ((1+\delta)^{L-1},(1+\delta)^L]$, where $\delta \in (0,1)$ is a small constant, and L is the smallest integer such that $(1+\delta)^{L-1} \geq t_{\max}$. Let $I_\ell = ((1+\delta)^{\ell-1},(1+\delta)^\ell]$, for $1 \leq \ell \leq L$, and $\mathcal{L} = \{1,2,\ldots,L\}$. Clearly, the number of intervals is polynomial in the size of the instance and in $\frac{1}{\delta}$.

We introduce an assignment variable $y_{i,k,j,\ell}$ indicating whether task $T_{k,j} \in \mathcal{T}$ is completed on processor $i \in \mathcal{P}$ within the interval I_ℓ. Furthermore, let $C_{k,j}$ be the completion time variable for a task $T_{k,j} \in \mathcal{T}$ and C_j be the completion time variable for a job $j \in \mathcal{J}$. (LP3) is an LP relaxation of the *multi-round MapReduce scheduling* problem, whose corresponding integer program is itself a $(1+\delta)$-relaxation.

Algorithm 1. MULTI-ROUND MRS: An algorithm for *multi-round MapReduce scheduling* on unrelated processors

```
1: Compute a fractional solution to the LP $(\bar{y}_{i,k,j,\ell}, \bar{C}_{k,j}, \bar{C}_j)$.
2: Partition the tasks into sets $S(\ell) = \{T_{k,j} \in b \mid (1+\delta)^{\ell-1} \leq \alpha \bar{C}_{k,j} < (1+\delta)^\ell\}$,
3: where $\alpha > 1$ is a fixed constant.
4: for each $\ell = 1 \ldots L$ do
5:     if $S(\ell) \neq \emptyset$ then
6:         Let $G_\ell$ be the precedence graph of the tasks of $S(\ell)$.
7:         $V_{1,\ell}, \ldots, V_{t,\ell}, \ldots, V_{l_{\max}+1,\ell} \leftarrow$ DECOMPOSE$(G_\ell)$
8:         for each $V_{t,\ell}$, in increasing order of $t$ do
9:             Integrally assign the tasks of $V_{t,\ell}$ on $\mathcal{P}$ using [18, Theorem 2.1].
10:            Schedule tasks of $V_{t,\ell}$ on $\mathcal{P}$, as early as possible, w.r.t. their precedences.
```

$$\begin{aligned}
\textbf{(LP3)}: \quad & \text{minimize} \sum_{j \in \mathcal{J}} w_j C_j \\
\text{s.t.} \quad & \sum_{i \in \mathcal{P}, \ell \in \mathcal{L}} y_{i,k,j,\ell} \geq 1, && \forall T_{k,j} \in \mathcal{T} \quad (6)\\
& C_j \geq C_{k,j}, && \forall T_{k,j} \in \mathcal{T} \quad (7)\\
& C_{k,j} \geq C_{k',j} + \sum_{i \in \mathcal{P}} p_{i,k,j} \sum_{\ell \in \mathcal{L}} y_{i,k,j,\ell}, && \forall T_{k',j} \prec T_{k,j} \quad (8)\\
& \sum_{i \in \mathcal{P}} \sum_{\ell \in \mathcal{L}} (1+\delta)^{\ell-1} y_{i,k,j,\ell} \leq C_{k,j}, && \forall T_{k,j} \in \mathcal{T} \quad (9)\\
& \sum_{T_{k,j} \in \mathcal{T}} p_{i,k,j} \sum_{t \leq \ell} y_{i,k,j,t} \leq (1+\delta)^{\ell}, && \forall i \in \mathcal{P}, \ell \in \mathcal{L} \quad (10)\\
& p_{i,k,j} > (1+\delta)^{\ell} \Rightarrow y_{i,k,j,\ell} = 0, && \forall i \in \mathcal{P}, T_{k,j} \in b, \ell \in \mathcal{L} \quad (11)\\
& y_{i,k,j,\ell} \geq 0, && \forall i \in \mathcal{P}, T_{k,j} \in b, \ell \in \mathcal{L}
\end{aligned}$$

Constraints (6) ensure that every task is completed on a processor of the set $\mathcal{P}$ in some time interval. Constraints (7) denote that the completion time of a job is determined by the completion time of its last finishing task. Constraints (8) describe the relation between the completion times of two jobs $T_{k,j} \succ T_{k',j}$, where the term $\sum_{i \in \mathcal{P}} p_{i,k,j} \sum_{\ell \in \mathcal{L}} y_{i,k,j,\ell}$ refers to the fractional processing time of $T_{k,j}$. Constraints (9) impose a lower bound on the completion time of each task. For each $\ell \in \mathcal{L}$, constraints (10), (11) are validity constraints which state that the total processing time of jobs executed up to an interval I_ℓ on a processor $i \in \mathcal{P}$ is at most $(1+\delta)^{\ell}$, and that if processing a task $T_{k,j}$ on a processor $i \in \mathcal{P}$ is greater than $(1+\delta)^{\ell}$, $T_{k,j}$ should not be scheduled on i, respectively.

Algorithm 1 considers a fractional solution $(\bar{y}_{i,k,j,\ell}, \bar{C}_{k,j}, \bar{C}_j)$ to (LP3) and rounds it to an integral schedule. It begins by separating the tasks into disjoint sets $S(\ell), \ell \in \mathcal{L}$ according to their fractional completion times $\bar{C}_{k,j}$. Since some of the tasks of each $S(\ell)$ may be related with precedence constraints, we proceed into a further partitioning of each set $S(\ell), \ell \in \mathcal{L}$ into pairwise disjoint sets $V_{t,\ell}, 1 \leq t \leq l_{\max} + 1$, with the following property: all the predecessors of any task in $V_{t,\ell}$ must belong either in a set $V_{t',\ell}$ with $t' < t$, or in a set $S(\ell')$ with $\ell' < \ell$. Let G be the precedence graph, given as input of the *multi-round MapReduce scheduling* problem. The above partitioning process on G can be done in polynomial time by the following simple algorithm.

DECOMPOSE(G). Identify the nodes of zero in-degree, i.e., $\delta^-(v) = 0$, in G. Add them in a set $V_{t,\ell}$, starting with $t = 1$, remove them from the graph, and set $t \leftarrow t + 1$. Repeat until there are no more nodes. Output the sets of tasks.

As the maximum path length in the precedence graph is $l_{\max}$, for each $\ell \in \mathcal{L}$, we could have at most $l_{\max} + 1$ sets $V_{t,\ell}$, with some of them possibly empty. Now, since there are no precedence constraints among the tasks of each set $V_{t,\ell}$, we integrally assign these tasks using the algorithm of [18, Theorem 2.1] in an increasing order of ℓ and t. The next lemmas prove an upper bound on the integral makespan of the tasks of every set $S(\ell)$ and $V_{t,\ell}$.

Lemma 3. *Suppose that we ignore any possible precedences among the tasks in $S(\ell)$, for each $\ell \in \mathcal{L}$. Then we can (fractionally) schedule them on the processors $\mathcal{P}$ with makespan at most $\frac{\alpha}{\alpha-1}(1+\delta)^{\ell}$.*

Now, since every set of tasks $V_{t,\ell}$ is a subset of $S(\ell)$, the aforementioned result on the fractional makespan of $S(\ell)$ also holds for every $V_{t,\ell} \subseteq S(\ell)$.

Lemma 4. *The tasks of every set $V_{t,\ell} \subseteq S(\ell)$ can be integrally scheduled on the processors $\mathcal{P}$ with makespan at most $(\frac{\alpha}{\alpha-1}+1)(1+\delta)^{\ell}$.*

Consider now a set of tasks $S(\ell)$ whose decomposition results in a sequence of pairwise disjoint subsets $V_{1,\ell}, \ldots, V_{t,\ell}, \ldots, V_{l_{\max}+1,\ell}$. Using the Lemma 4, we see that if we integrally schedule each subset $V_{t,\ell}$ in a time window of $(\frac{\alpha}{\alpha-1}+1)(1+\delta)^{\ell}$ and then place the schedules in an increasing order of t, the resulting schedule would respect all constraints and would have makespan at most $(l_{\max}+1)(\frac{\alpha}{\alpha-1}+1)(1+\delta)^{\ell}$. Now, we can prove the following.

Theorem 4. *Algorithm 1 is an $\alpha[(l_{\max}+1)\frac{\alpha}{\alpha-1}+l_{\max}\frac{\alpha}{\delta(\alpha-1)}+l_{\max}+1+\frac{l_{\max}+1}{\delta}](1+\delta)$-approximation for the* multi-round MapReduce scheduling *problem on unrelated processors, where $l_{\max}$ is the maximum length over all maximal paths in the precedence graph, and $\alpha > 1$, $\delta > 0$ are fixed constants.*

Proof. First, we need to note that the tasks of each set $S(\ell)$ can be scheduled integrally in the processors of $\mathcal{P}$ with makespan equal to the sum of makespans of the subsets $V_{t,\ell}, 1 \leq t \leq l_{\max}+1$. The rounding theorem of [18, Theorem 2.1] suggests that the makespan of an integral schedule of tasks in $V_{t,\ell}$ is at most the fractional assignment, $\Pi_{t,\ell} \leq \frac{\alpha}{\alpha-1}(1+\delta)^{\ell}$, of tasks to processors plus the maximum processing time on every processor, $p_{t,\ell}^{max} \leq (1+\delta)^{\ell}$. Therefore, the sets $V_{1,\ell}$ to $V_{l_{\max},\ell}$ can be scheduled with makespan at most $l_{\max}(\frac{\alpha}{\alpha-1}+1)(1+\delta)^{\ell}$, in order to respect the precedences among them. Now, consider the sets $V_{l_{\max}+1,\ell}, \forall \ell \in \mathcal{L}$. Clearly, these must include the last finishing tasks of any chain in the precedence graph. Therefore, by constraints (10), it is the case that $\sum_{t \leq \ell} \Pi_{l_{\max}+1,t} \leq \frac{\alpha}{\alpha-1}(1+\delta)^{\ell}$.

Now, let $T_{k,j} \in \mathcal{T}$ be the last finishing task of a job $j \in \mathcal{J}$ which is scheduled on a processor $i \in \mathcal{P}$. Suppose, w.l.o.g., that $T_{k,j}$ belongs to the set $S(\ell)$. By Lemma 4 and Lemma 3, taking the union of the schedules of tasks in $S(\ell')$, with $\ell' \leq \ell$, it must hold that the completion time of $T_{k,j}$ in the resulting schedule is:

$$C_{k,j} \leq \sum_{\ell' \leq \ell} [l_{\max}(\frac{a}{a-1}+1)(1+\delta)^{\ell'} + \Pi_{l_{\max}+1,\ell'} + p_{l_{\max}+1,\ell'}^{max}]$$
$$\leq \alpha\left((l_{\max}+1)\frac{\alpha}{\alpha-1} + l_{\max}\frac{\alpha}{\delta(\alpha-1)} + l_{\max} + 1 + \frac{l_{\max}+1}{\delta}\right)(1+\delta)\bar{C}_{k,j}.$$

□

As for *single-round MapReduce scheduling*, for all the maximal paths of each job j in the underlying graph, $l_j = 1$. By Theorem 4 with $(\alpha, \delta) \approx (1.65, 0.80)$, we get that:

Corollary 1. *There is a 37.87-approximation algorithm for the* single-round MapReduce scheduling *problem on unrelated processors.*

A Note on the Computational Complexity. Concerning the hardness of *multi-round MapReduce scheduling* on unrelated processors, we note it is a generalization of the standard job-shop scheduling, where the precedence constraints are restricted to be a disjoint union of chains and the task assignment is given in advance, under the average weighted completion time objective. However, for the latter one, we know that it is NP-hard to obtain an $O(1)$-approximation and it does not admit an $O(\log^{1-\epsilon} lb)$-approximation algorithm for any $\epsilon > 0$, unless NP $\subseteq$ ZTIME$(2^{\log^{1/\epsilon} n})$, where lb is a standard lower bound on the makespan of any schedule [14]. Thus, the best we can expect is no more than a logarithmic improvement on our approximation ratio.

5 Simulation Results

We conclude with simulation results for *multi-round MapReduce scheduling* on unrelated processors. We compare our algorithm against the simple heuristic Fast-MR of [8] and against a lower bound derived from (LP3). We provide evidence that the empirical approximation ratio of Algorithm 1 is significantly better than the theoretical one.

Fast-MR operates in two steps. First, it computes an online assignment of tasks to processors, using the online algorithm of [3], and then, it schedules them using a variant of Weighted Shortest Processing Time first wrt. the multi-round task precedences.

Computational Experience and Results. We generate instances consisting of 30 indistinguishable processors and from 5 to 50 jobs. Each job consists of 5 rounds, where the number of map and reduce tasks in each round ranges from 20 to 35 and from 5 to 15, respectively. The weight of each job is uniformly distributed in $[1, n]$, where n is the number of jobs. Moreover, the parameters of Algorithm 1 are fixed to $\delta = 0.96$ and $\alpha = 1.69$. To better capture the unrelated nature of the processors as well as data locality issues, we generate the task processing times in each processor in a processor-task correlated way, extending on the model of [11]. Specifically, the processing times $\{p_{i,k,j}\}_{i \in \mathcal{P}}$ of each map task are equal to $b_j a_{j,i}$ plus some noise selected u.a.r. from $[0, 10]$, where b_j and $a_{j,i}$ are selected u.a.r. from $[1, 10]$, for each job $j \in \mathcal{J}$ and each processor $i \in \mathcal{P}$. The processing time of each reduce task, taking into account that is practically larger, is set to $3 b_j a_{j,i}$ plus some noise selected u.a.r. from $[0, 10]$. In this context, we simulate both Algorithm 1 and Fast-MR by running 10 different trials for each possible number of jobs. Since in various applications a MapReduce computation is performed within a single round, we also simulate

Algorithm 1 in the single-round case, called Single-Round MRS and compare it against Fast-MR. Note that in the latter case, we fix $\alpha = 1.65, \delta = 0.80$ according to Corollary 1. The instances and the results are available at http://www.corelab.ntua.gr/~opapadig/mrrounds/.

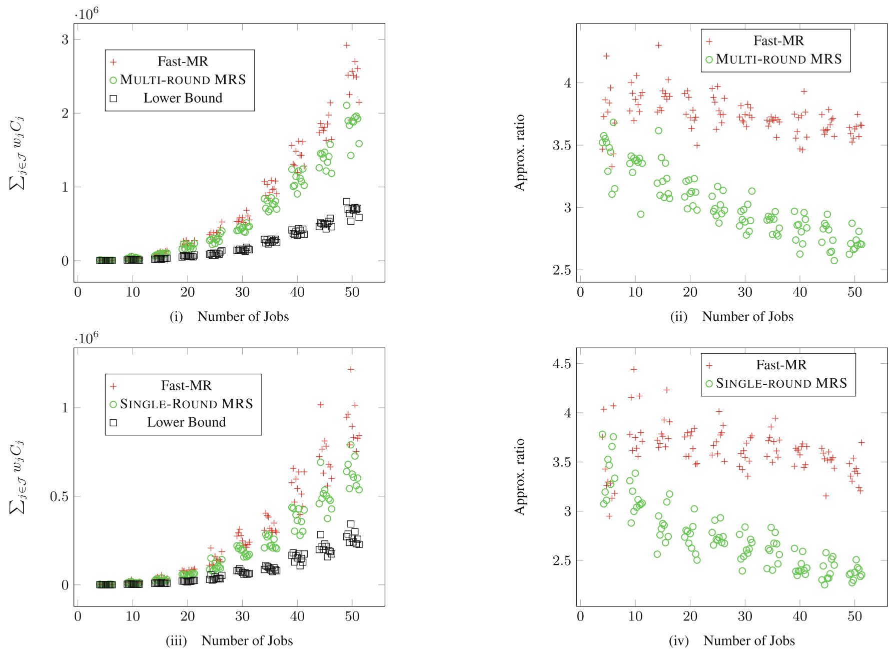

Fig. 2. Simulation results for the single-round and multi-round cases, in terms of absolute values and (empirical) approximation ratios. (Color figure online)

In Figs. 2 (i)–(ii), we note that Algorithm 1 outperforms the Fast-MR heuristic, for any simulated number of jobs. More specifically, the empirical approximation ratio of Fast-MR, ranges from 3.32 to 4.30, while the ratio of Algorithm 1 ranges from 2.57 to 3.68. More interestingly, the gap between the performance guarantee of the two algorithms is growing as the number of jobs is increasing: For $n = 5$ jobs the average ratios of the algorithms Algorithm 1 and Fast-MR are 3.43 and 3.72, while for $n = 50$, the average ratio converges to 2.71 and 3.62, respectively. Over all trials, we can see that Algorithm 1 produces up to 28.4 % better solutions. In Figs. 2 (iii)–(iv), we note that Single-round MRS also outperforms Fast-MR, producing up to 36.7 % better solutions. Similarly to Algorithm 1, its empirical approximation ratio ranges from 2.25 to 3.78 (vs. the ratio of Fast-MR which ranges from 2.94 to 4.44), while the gap against the approximation ratio of Fast-MR increases as the number of jobs increasing (e.g., for $n = 50$, Single-round MRS achieves ratio 2.37, while Fast-MR 3.40). Note that, the empirical approximation ratios in both multi-round and single-round cases of our algorithm are far from our theoretical worst-case approximation guarantees.

References

1. Afrati, F.N., Das Sarma, A., Salihoglu, S., Ullman, J.D.: Upper and lower bounds on the cost of a MapReduce computation. VLDB **6**(4), 277–288 (2013)
2. Afrati, F., Joglekar, M., Salihoglu, C.R.S., Ullman, J.D.: GYM: A multiround join algorithm in MapReduce (2014). arXiv:1410.4156
3. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. JACM **44**(3), 486–504 (1997)
4. Chen, F., Kodialam, M.S., Lakshman, T.V.: Joint scheduling of processing and shuffle phases in mapreduce systems. In: INFOCOM, pp. 1143–1151 (2012)
5. Correa, J.R., Skutella, M., Verschae, J.: The power of preemption on unrelated machines and applications to scheduling orders. Math. Oper. Res. **37**(2), 379–398 (2012)
6. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: OSDI, pp. 137–150 (2004)
7. Eastman, W.L., Even, S., Iaacs, I.M.: Bounds for the optimal scheduling of n jobs on m processors. Manage. Sci. **11**, 268–279 (1964)
8. Fotakis, D., Milis, I., Papadigenopoulos, O., Zampetakis, E., Zois, G.: Scheduling MapReduce jobs and data shuffle on unrelated processors. In: Bampis, E. (ed.) SEA 2015. LNCS, vol. 9125, pp. 137–150. Springer, Heidelberg (2015)
9. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAP **17**(2), 416–429 (1969)
10. Hall, L.A., Schulz, A.S., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. MOR **22**, 513–544 (1997)
11. Hariri, A.M., Potts, C.N.: Heuristics for scheduling unrelated parallel machines. Comp. and Oper. Res. **18**(3), 323–331 (1991)
12. Karloff, H., Suri, S., Vassilvitskii, S.: A model of computation for MapReduce. In: SODA, pp. 263-285 (2010)
13. Kumar, R., Moseley, B., Vassilvitskii, S., Vattani, A.: Fast greedy algorithms in mapreduce and streaming. In: SPAA, pp. 1–10 (2013)
14. Mastrolilli, M., Svensson, O.: Hardness of approximating flow and job shop scheduling problems. JACM **58**(5), 20 (2011)
15. Moseley, B., Dasgupta, A., Kumar, R., Sarlós, T.: On scheduling in Map-Reduce and flow-shops. In: SPAA, pp. 289–298 (2011)
16. Queyranne, M., Schulz, A.S.: Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. SICOMP **35**(5), 1241–1253 (2006)
17. Queyranne, M.: Structure of a simple scheduling polyhedron. Math. Program. **58**(1), 263–285 (1993)
18. Shmoys, D.B., Tardos, É.: An approximation algorithm for the generalized assignment problem. Math. Program. **62**, 461–474 (1993)
19. Yoo, D.-J., Sim, K.M.: A comparative review of job scheduling for MapReduce. In: CCIS, pp. 353–358 (2011)