Scheduling MapReduce Jobs on Unrelated Processors*

D. Fotakis
National Technical University of Athens
fotakis@cs.ntua.gr

I. Milis
Athens University of Economics and Business
milis@aueb.gr

E. Zampetakis
National Technical University of Athens
mzampet@corelab.ntua.gr

G. Zois
Université Pierre et Marie Curie and Athens University of Economics and Business
Georgios.Zois@lip6.fr

ABSTRACT

MapReduce framework is established as the standard approach for parallel processing of massive amounts of data. In this work, we extend the model of MapReduce scheduling on unrelated processors (Moseley et al., SPAA 2011) and deal with the practically important case of jobs with any number of Map and Reduce tasks. We present a polynomial-time $(32 + \epsilon)$-approximation algorithm for minimizing the total weighted completion time in this setting. To the best of our knowledge, this is the most general setting of MapReduce scheduling for which an approximation guarantee is known. Moreover, this is the first time that a constant approximation ratio is obtained for minimizing the total weighted completion time on unrelated processors under a nontrivial class of precedence constraints.

Keywords

MapReduce, Scheduling, Unrelated Processors

1. INTRODUCTION

Scheduling in MapReduce environments has become increasingly important during the last years, as MapReduce has been established as the standard programming model to implement massive parallelism in large data centers [5]. Applications of MapReduce such as search indexing, web analytics and data mining, involve the concurrent execution of several MapReduce jobs on a system like Google's MapReduce or Apache Hadoop. When a MapReduce job is executed, a number of Map and Reduce tasks are created. Each Map task operates on a portion of the input elements, translating them into a number of key-value pairs. Next, all key-value pairs are transmitted to the Reduce tasks, so that all pairs with the same key are available together at the same task. The Reduce tasks operate on the key-value pairs, combine the values associated with a key, and generate the final result. In addition to the many practical applications of MapReduce, there has been a significant interest in developing appropriate cost models and a computational complexity theory for MapReduce computation (see e.g., [3, 6]), in understanding the basic principles underlying the design of efficient MapReduce algorithms (see e.g., [1, 7]), and in obtaining upper and lower bounds on the performance of MapReduce algorithms for some fundamental computational problems (see e.g. [2] and the references therein).

Motivation and Previous Work. Many important advantages of MapReduce are due to the fact that the Map tasks or the Reduce tasks can be executed in parallel and essentially independent from each other. However, to best exploit massive parallelism available in typical MapReduce systems, one has to carefully allocate and schedule Map and Reduce tasks to actual processors (or computational resources, in general). This important and delicate task is performed in a centralized manner, by a process running in the master node. A major concern of the scheduler, among others, is to satisfy task dependencies within the tasks of the same MapReduce job; *all the Map tasks must finish before the execution of any Reduce task of the same job.* During the assignment and scheduling process, a number of different needs must be taken into account, e.g., transferring of the intermediate data (shuffle), data locality, and data skew, which give rise to the study of new scheduling problems.

Despite the importance and the challenging nature of scheduling in MapReduce environments, and despite the extensive investigation of a large variety of scheduling problems in parallel computing systems (see e.g., [13]), less attention has been paid to MapReduce scheduling problems. In fact, most of the previous work on scheduling in MapReduce systems concerns the experimental evaluation of scheduling heuristics, mostly from the viewpoint of finding good trade-offs between different objectives (see e.g., [14]). From a theoretical viewpoint, only few results on MapReduce scheduling have appeared so far [11, 4].These are based on simplified abstractions of MapReduce scheduling, closely-related to some variants of the classical Open Shop and Flow Shop scheduling models, that capture issues such as task dependencies,

*This work was supported by the project Handling Uncertainty in Data Intensive Applications, co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning", under the program THALES, and by the project Heracleitus II.

data locality, shuffle, and task assignment, under the key objective of minimizing the total weighted completion time of a set of MapReduce jobs.

In this direction, the theoretical model of Moseley et al. [11] generalizes a variant of the Flow Shop scheduling model, referred to as 2-stage Flexible Flow Shop (FFS), which is known to be strongly $\mathcal{NP}$-hard, even for jobs of a single Map and Reduce task and a single map and reduce processor (see in [11]). They consider the cases of both identical and unrelated processors and the goal is to minimize the total completion time of the jobs. For identical processors, they present a 12-approximation algorithm, and a $O(1/\epsilon^2)$-competitive online algorithm, for any $\epsilon \in (0,1)$, under the assumption that the processors used by the online algorithm are $1+\epsilon$ times faster than the processors used by the optimal schedule. Since the identical processors setting fails to capture issues as data locality and to model communication costs between the Map and the Reduce tasks, Moseley et al. also consider the case of unrelated processors, which provides a more expressive theoretical model of scheduling in MapReduce environments. Nevertheless, they only consider the very restricted (and practically not so interesting) case where each job has a single Map and a single Reduce task, and present a 6-approximation algorithm and a $O(1/\epsilon^5)$-competitive online algorithm, for any $\epsilon \in (0,1)$, under the assumption that the processors of the online algorithm are $1+\epsilon$ times faster.

A similar model of MapReduce scheduling so as to minimize the total completion time was proposed by Chen et al. [4]. In contrast with the model of [11], they assume that tasks are preassigned to processors and, in this restricted setting, they present an LP-based 8-approximation algorithm. Moreover, they deal with the shuffle phase in MapReduce systems and present a 58-approximation algorithm.

Contribution and Results. We adopt the theoretical model of [11] and consider MapReduce scheduling on unrelated processors. However, departing from [11], we deal with the general (and practically interesting) case where each job has any number of Map and Reduce tasks and we succeed in obtaining a polynomial-time constant approximation algorithm for minimizing the total weighted completion time. More specifically, we consider a set of MapReduce jobs to be executed on a set of unrelated processors. Each job consists of a set of Map tasks, that can be executed only on map processors, and a set of Reduce tasks, that can be executed only on Reduce processors. Each task has a different processing time for each processor and is associated with a positive weight, representing its importance. All jobs are available at time zero. Map or Reduce tasks can run simultaneously on different processors and, for each job, every Reduce task can start its execution after the completion of all the job's Map tasks. The goal is to find an assignment of the tasks to processors and schedule them *non-preemptively* so as to minimize their total weighted completion time.

In terms of classical scheduling, the model we consider in this work is a special case of total weighted completion time minimization on unrelated processors under precedence constraints. Despite its importance and generality, only few results are known for this problem. These results concern only the case of treelike precedence constraints [8]. More specifically, in [8], Kumar et al. propose a polylogarithmic approximation algorithm for the case where the undirected graph underlying the precedence constraints is a forest (a.k.a. treelike precedences). Their algorithm is based on a reduction from total weighted completion time minimization to an appropriate collection of makespan minimization problems. Based on ideas of [8], we present a $(32+\epsilon)$-approximation algorithm for this problem that operates in two steps. In the first step, our algorithm computes a $(8+\epsilon)$-approximation schedule for the Map tasks (resp. Reduce tasks) by combining a time indexed LP-relaxation of the problem with a well-known approximation algorithm for the makespan minimization problem on unrelated processors [9]. In fact, the makespan minimization algorithm runs on each time interval of the LP solution and computes an assignment of the Map (resp. Reduce) tasks to processors. In the second step, based on an idea from [11], we merge the two schedules, produced for the Map tasks and the Reduce tasks, into a single schedule that respects the precedence constraints. Using techniques from [11], we show that the merging step increases the approximation ratio by a factor of at most 4.

On the practical side, the theoretical model of [11] for MapReduce scheduling on unrelated processors deals with the most of the important aspects of the problem. So, considering jobs with any number of Map and Reduce tasks in this model is particularly important for practical applications, since the basic idea behind MapReduce computation is that each job is split into a large number of Map and Reduce tasks that can be executed in parallel (see e.g., [3, 6, 1, 2]). On the theoretical side, to the best of our knowledge, this is the first time that a constant approximation ratio is obtained for the problem of minimizing the total weighted completion time on unrelated processors under a nontrivial class of precedence constraints.

Notation. 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 unrelated processors. Each job is available at time zero, is associated with a positive weight w_j and consists of a set $\mathcal{M}$ of Map tasks and a set $\mathcal{R}$ of Reduce tasks. Each task is denoted by $\mathcal{T}_{k,j} \in \mathcal{M} \cup \mathcal{R}$, where $k \in N$ is the task index of job $j \in \mathcal{J}$ and is associated with a vector of non-negative processing times $\{p_{i,k,j}\}$, one for each processor $i \in \mathcal{P}_b$, where $b \in \{\mathcal{M}, \mathcal{R}\}$. Let $\mathcal{P}_{\mathcal{M}}$ and $\mathcal{P}_{\mathcal{R}}$ be the sets of map and reduce processors respectively. Each job has at least one Map and one Reduce task that can run simultaneously on different processors and 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 $C_{k,j}$ the completion times of each job $j \in \mathcal{J}$ and each task $\mathcal{T}_{k,j} \in \mathcal{M} \cup \mathcal{R}$ respectively. Note that, due to the precedence constraints between Map and Reduce tasks, $C_j = \max_{\mathcal{T}_{k,j} \in \mathcal{R}}\{C_{k,j}\}$. By $C_{max} = \max_{j \in \mathcal{J}}\{C_j\}$ we denote the makespan of the schedule, i.e., the completion time of the job which finishes last. Our goal is to schedule *non-preemptively* all Map tasks on processors of $\mathcal{P}_{\mathcal{M}}$ and all Reduce tasks on processors of $\mathcal{P}_{\mathcal{R}}$, with respect to their precedence constraints, so as to minimize the total weighted completion time of the schedule, i.e., $\sum_{j \in \mathcal{J}} w_j C_j$. We refer to this problem as MapReduce scheduling problem.

2. A CONSTANT APPROXIMATION ALGORITHM

In this section, we present a $(32+\epsilon)$-approximation algorithm, for $\epsilon \in (0,1)$, executed in the following two steps:

(i) it computes a $(8+\epsilon)$-approximate schedule for assigning and scheduling all Map tasks (resp. Reduce tasks) on processors of the set $\mathcal{P}_\mathcal{M}$ (resp. $\mathcal{P}_\mathcal{R}$) and (ii) it merges the two schedules in one, with respect to the precedence constraints between Map and Reduce tasks of each job, increasing the approximation ratio by a factor of 4.

2.1 Scheduling Map and Reduce Tasks

Next, we propose an algorithm for the problem of minimizing the total weighted completion time of all Map (resp. Reduce) tasks on processors of the set $\mathcal{P}_\mathcal{M}$ (resp. $\mathcal{P}_\mathcal{R}$). For notational convenience, we use a dual variable $b \in \{\mathcal{M}, \mathcal{R}\}$ to refer on either Map or Reduce sets of tasks.

We define $(0, t_{\max} = \sum_{\mathcal{T}_{k,j} \in b} \max_{i \in \mathcal{P}_b} p_{i,k,j}]$ to be the time horizon of potential completion times, where $t_{\max}$ is an upper bound on the makespan of a feasible schedule. We discretize the time horizon into intervals $(1,1], (1,(1+\delta)], ((1+\epsilon),(1+\delta)^2], \ldots, ((1+\delta)^{L-1}, (1+\delta)^L]$, where $\delta \in (0,1)$ is a small constant, and L is the smallest integer such that $(1+\delta)^{L-1} \geq t_{\max}$. Let $I_\ell = ((1+\delta)^{\ell-1}, (1+\delta)^\ell]$, for $0 \leq \ell \leq L$, and $\mathcal{L} = \{0, 1, 2, \ldots, L\}$. Note that, the number of intervals is polynomial in the size of the instance and to $1/\delta$. For each processor $i \in \mathcal{P}_b$, task $T_{k,j} \in b$ and $\ell \in \mathcal{L}$, we introduce a variable $y_{i,k,j,\ell}$ that denotes the fraction of task $T_{k,j}$ assigned to processor i in time interval I_ℓ. Furthermore, for each task $T_{k,j} \in \mathcal{T}$, we introduce a variable $C_{k,j}$ corresponding to its completion time, and a variable $z_{k,j}$ corresponding to its fractional processing time. For every job $j \in \mathcal{J}$, we also introduce a dummy task D_j, with zero processing time on every processor, which has to be processed after the completion of every other task $T_{k,j} \in b$. $LP(b)$ is an interval-indexed linear programming relaxation of our problem.

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

subject to :

$$\sum_{i \in \mathcal{P}_b, \ell \in \mathcal{L}} y_{i,k,j,\ell} = 1, \qquad \forall \mathcal{T}_{k,j} \in b \quad (1)$$

$$z_{k,j} = \sum_{i \in \mathcal{P}_b} p_{i,k,j} \sum_{\ell \in \mathcal{L}} y_{i,k,j,\ell}, \qquad \forall \mathcal{T}_{k,j} \in b \quad (2)$$

$$C_{D_j} \geq C_{k,j} + z_{k,j}, \qquad \forall j \in \mathcal{J}, \mathcal{T}_{k,j} \in b \quad (3)$$

$$\sum_{i \in \mathcal{P}_b} \sum_{\ell \in \mathcal{L}} (1+\delta)^{\ell-1} y_{i,k,j,\ell} \leq C_{k,j} \leq \sum_{i \in \mathcal{P}_b} \sum_{\ell \in \mathcal{L}} (1+\delta)^{\ell} y_{i,k,j,\ell}, \quad \forall \mathcal{T}_{k,j} \in b \quad (4)$$

$$\sum_{\mathcal{T}_{k,j} \in b} p_{i,k,j} \sum_{t \leq \ell} y_{i,k,j,t} \leq (1+\delta)^\ell, \qquad \forall i \in \mathcal{P}_b, \ell \in \mathcal{L} \quad (5)$$

$$p_{i,k,j} > (1+\delta)^\ell \Rightarrow y_{i,k,j,\ell} = 0, \ \forall i \in \mathcal{P}_b, \mathcal{T}_{k,j} \in b, \ell \in \mathcal{L} \quad (6)$$

$$y_{i,k,j,\ell} \geq 0, \qquad \forall i \in \mathcal{P}_b, \mathcal{T}_{k,j} \in b, \ell \in \mathcal{L} \quad (7)$$

Our objective is to minimize the sum of weighted completion times of all jobs. Constraint (1) ensures that each task is entirely assigned to processors of the set $\mathcal{P}_b$ and constraint (2) defines its fractional processing time. Constraint (3) ensures that, for each job $j \in \mathcal{J}$, the completion of each task $\mathcal{T}_{k,j}$ precedes the completion of task D_j. Constraint (4) adapts a lower and an upper bound on the completion time of each task. For each $\ell \in \mathcal{L}$, constraints (5) and (6) are validity constraints which state that the total fractional processing time on each processor is at most $(1+\delta)^\ell$, and that if it takes time more than $(1+\delta)^\ell$ to process a task $T_{j,k}$ on a processor $i \in \mathcal{P}_b$, then $T_{k,j}$ should not be scheduled on i, respectively.

Assignment and Scheduling. Let $(\bar{y}_{i,k,j,l}, \bar{z}_{k,j}, \bar{C}_{k,j})$ be an optimal (fractional) solution to $LP(b)$. For each $2 \leq \ell \leq L$, we define the set of tasks $S(\ell) = \{T_{k,j} \in b \mid (1+\delta)^{\ell-2}/2 \leq \bar{C}_{k,j} \leq (1+\delta)^{\ell-1}/2\}$, that complete their execution within the interval I_ℓ. By definition, for each task $\mathcal{T}_{k,j} \in S(\ell)$, it must hold that $2(1+\delta)\bar{C}_{k,j} \leq (1+\delta)^\ell$.

We will assign all jobs of each set $S(\ell)$ to processors in $\mathcal{P}_b$ according to the following algorithm.

Algorithm MAKESPAN

1: Compute a basic feasible solution $(\bar{x}_{i,k,j})$ to $LP(T^\star, b)$.
2: Assign all tasks having integral values to processors of $\mathcal{P}_b$ as in $(\bar{x}_{i,k,j})$.
3: Let a graph $G = (A \cup \mathcal{P}_b, E)$, where $A = \{\mathcal{T}_{k,j} \mid 0 < x_{i,j,k} < 1\}$ and $E = \{\{\mathcal{T}_{k,j}, i\} \mid \mathcal{T}_{k,j} \in A, i \in \mathcal{P}_b \text{ and } 0 < x_{i,k,j} < 1\}$. Compute a perfect matching M on G.
4: Assign each $\mathcal{T}_{k,j} \in A$ to $i \in \mathcal{P}_b$, as indicated by M.
5: **for** each assigned task $\mathcal{T}_{k,j}$ **do**
6: Schedule $\mathcal{T}_{k,j}$ as early as possible, non-preemptively, with processing time $p_{i,k,j}$ on processor $i \in \mathcal{P}_b$ that is assigned to. Let $C_{k,j}$ be the completion time of $\mathcal{T}_{k,j}$.

Algorithm MAKESPAN has been proposed in a seminal paper by Lenstra et al. [9] and it is based on the so-called parametric pruning technique in an LP setting. More specifically, if T is an estimation on the optimal makespan of a schedule of the jobs in $S(\ell)$, then by pruning away all task-processor pairs for which $p_{i,k,j} > T$, we are able to define a set of variables corresponding only to triples of the set $\mathcal{Q}_T = \{(i,k,j) | p_{i,k,j} \leq T\}$; note that this pruning process has been already taken under consideration by constraints (6) of $LP(b)$. Since $T \in \cup_{\ell' \leq \ell} I_{\ell'}$, using binary search on $\cup_{\ell' \leq \ell} I_{\ell'}$ with T as the search variable, we can find the minimum value of T such that the following system of linear constraints is feasible.

$$LP(b,T):$$

$$\sum_{i:(i,k,j) \in \mathcal{Q}_T} x_{i,k,j} = 1 \qquad \forall T_{k,j} \in b \quad (8)$$

$$\sum_{T_{k,j}:(i,k,j) \in \mathcal{Q}_T} x_{i,k,j} p_{i,k,j} \leq T \qquad \forall i \in \mathcal{P}_b \quad (9)$$

$$x_{i,k,j} \geq 0 \qquad \forall (i,k,j) \in \mathcal{Q}_T$$

Each variable $x_{i,k,j}$ denotes the fractional processor assignment of each task $\mathcal{T}_{k,j} \in S(\ell)$. Now, if $T^\star$ is the minimum value for which $LP(b,T)$ is feasible, then $T^\star$ is a lower bound on the optimal integral makespan.

Similarly as in [9], it can be proved that a basic feasible solution to $LP(b,T)$ has at most $|b| + |\mathcal{P}_b|$ non-zero variables, from which at least $|b| - |\mathcal{P}_b|$, must be set integrally. Then, the number of fractional $x_{i,k,j}$ values must be at most $2|\mathcal{P}_b|$. If we formulate a bipartite graph $G = (A \cup \mathcal{P}_b, E)$, where A is the set of tasks having fractional $x_{i,k,j}$ values and $E = \{\{\mathcal{T}_{k,j}, i\} \mid \mathcal{T}_{k,j} \in A, i \in \mathcal{P}_b \text{ and } 0 < x_{i,k,j} < 1\}$, then, according to the latter property, we deduce that G is a connected graph with at most $2|\mathcal{P}_b|$ vertices and at most $2|\mathcal{P}_b|$ edges. However, this means that G has the special topology of a pseudo-forest (a collection of trees with one possi-

ble extra edge) which enables the computation of a perfect matching on it. Hence, by executing steps 2-6 of Algorithm MAKESPAN, a non-preemptive schedule of tasks in $S(\ell)$ can be found.

The following lemma provides a tight upper bound on the makespan of the schedule computed by Algorithm MAKESPAN.

LEMMA 1. *Algorithm* MAKESPAN *is a 2-approximation algorithm for scheduling the tasks of the set $S(\ell)$ so as to minimize their makespan.*

In the next lemma, using filtering [10] we modify the $y_{i,k,j,\ell}$ values of the solution to $LP(b)$ to find an upper bound on the value of T^*.

LEMMA 2. *Consider a feasible solution to $LP(b,T)$. For each set of jobs $S(\ell)$ that complete their execution within the interval I_ℓ, it holds that $T^\star \leq 2(1+\delta)^\ell$, for $\delta \in (0,1)$.*

As consequence of filtering in Lemma 2 the completion time of each task in $\mathcal{S}(\ell)$ is increased by a factor of 4; this result has already proven to be tight (see Section 2 in [12]).

Algorithm TASKSCHEDULING(b)

1: Compute an optimal solution $(\bar{y}_{i,k,j,l}, \bar{z}_{k,j}, \bar{C}_{k,j})$ to $LP(b)$.
2: **for** each $\ell \in \mathcal{L}$ **do**
3: compute $S(\ell) = \{T_{k,j} \in b \mid (1+\delta)^{\ell-2}/2 \leq \bar{C}_{k,j} \leq (1+\delta)^{\ell-1}/2\}$
4: **for** each ℓ such that $S(\ell) \neq \emptyset$ **do**
5: Schedule all tasks in $S(\ell)$ by running Algorithm MAKESPAN.

Running Algorithm TASKSCHEDULING(b), we compute a schedule for all Map (resp. Reduce) tasks such that:

THEOREM 1. TASKSCHEDULING*(b) is a $(8+\varepsilon)$-approximation algorithm, for scheduling a set of Map (Reduce) tasks on a set of unrelated processors $\mathcal{P}_\mathcal{M}$ ($\mathcal{P}_\mathcal{R}$), in order to minimize their total weighted completion time, for $\varepsilon \in (0,1)$.*

PROOF SKETCH. Let $C_{k,j}$ be the completion time of a task $T_{k,j} \in S(\ell)$, in the schedule of Algorithm TASKSCHEDULING(b) and let $C_{max}(\ell)$ be the makespan of the schedule of Algorithm MAKESPAN on the jobs in $S(\ell)$. Since, $C_{k,j} \leq C_{max}(\ell)$, for all $\mathcal{T}_{k,j} \in b$, it suffices to prove that $C_{k,j} \leq 8(1+\delta)^2\bar{C}_{k,j}$: we combine Lemma 1 and Lemma 2 with the definition of the set $S(\ell)$. Then, as we can select an ε such that $(1+\delta)^2 \leq (1+\varepsilon)$, the theorem follows. Note that this ratio is tight. □

2.2 Merging Task Schedules

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

Algorithm MRS. In each time instant where a processor $i \in \mathcal{P}_b$ becomes available, either it processes the Map task, assigned to $i \in \mathcal{P}_\mathcal{M}$ in $\sigma_\mathcal{M}$, with the minimum width, or the available (w.r.t. its precedence constraints) Reduce task, assigned to $i \in \mathcal{P}_\mathcal{R}$ in $\sigma_\mathcal{R}$, with the minimum width.

By an analysis similar to that in [11], we can prove that:

THEOREM 2. *Algorithm* MRS *is a $(32+\epsilon)$-approximation for the* MAPREDUCE SCHEDULING *problem, for $\epsilon \in (0,1)$.*

PROOF SKETCH. By execution of Algorithm MRS, the feasibility of the resulted schedule can be easily verified. To prove the theorem, it suffices to prove that in such a schedule, σ, all tasks of a job $j \in \mathcal{J}$ are completed by time $2\max\{C_j^{\sigma_\mathcal{M}}, C_j^{\sigma_\mathcal{R}}\}$. Let C_j^σ, be the completion time of a job $j \in \mathcal{J}$ in σ. Note that, for each of the Map tasks of j, their completion time is upper bounded by ω_j. On the other hand, the completion time of each Reduce task is upper bounded by a quantity equal to $r + \omega_j$, where r is the earliest time when the task is available to be scheduled in σ. However, $r = C_j^{\sigma_\mathcal{M}} \leq \omega_j$ and thus $C_j^\sigma \leq 2\omega_j = 2\max\{C_j^{\sigma_\mathcal{M}}, C_j^{\sigma_\mathcal{R}}\}$. By applying Theorem 1 and as we can select an ϵ such that $\epsilon \leq 4\varepsilon$, the theorem follows. □

3. REFERENCES

[1] F. Afrati, D. Fotakis, and J. Ullman. Enumerating subgraph instances using MapReduce. *IEEE-ICDE*: 62-73, 2013.

[2] F. Afrati, A. D. Sarma, S. Salihoglu, and J. Ullman. Upper and Lower Bounds on the Cost of a MapReduce Computation. *VLDB*: 6(4):277-288, 2013.

[3] F. Afrati and J. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE-TKDE*: 23(9):1282-1298, 2011.

[4] F. Chen, M. S. Kodialam, and T. V. Lakshman. Joint scheduling of processing and shuffle phases in mapreduce systems. *INFOCOM*: 1143-1151, 2012.

[5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*: 137-150, 2004.

[6] H. Karloff, S. Suri, and S. Vassilvitskii. A Model of Computation for MapReduce. *SODA*: 938-948, 2010.

[7] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in MapReduce and streaming. *ACM-SPAA*: 1-10, 2013.

[8] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan. Scheduling on unrelated machines under tree-like precedence constraints. *Algorithmica*: 55(1):205-226, 2009.

[9] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*: 46:259-271, 1990.

[10] J. Lin and J. S. Vitter. epsilon-approximations with minimum packing constraint violation. *SODA*: pages 771–782, 1992.

[11] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós. On scheduling in map-reduce and flow-shops. *ACM-SPAA*: 289-298, 2011.

[12] J. R. Correa and M. Skutella and J. Verschae. The Power of Preemption on Unrelated Machines and Applications to Scheduling Orders. *Math. Oper. Res.*: 379-398, 2012.

[13] M. Pinedo. *Scheduling: theory, algorithms, and systems.* Springer, 2012.

[14] D.-J. Yoo and K. M. Sim. A comparative review of job scheduling for mapreduce. *IEEE-ICCIS*: 353-358, 2011.