Minimizing Makespan and Total Completion Time in MapReduce-like Systems

Yuqing Zhu*, Yiwei Jiang†, Weili Wu*, Ling Ding‡, Ankur Teredesai‡, Deying Li§✉ , and Wonjun Lee¶
*Department of Computer Science, University of Texas at Dallas. Email: yuqing.zhu@utdallas.edu
†Department of Mathematics, Zhejiang Sci-Tech University.
‡Institute of Technology, University of Washington Tacoma.
§School of Information, Renmin University of China.
¶Dept. of Computer Science & Engineering, Korea University.

***Abstract*—Effectiveness of MapReduce as a big data processing framework depends on efficiencies of scale for both map and reduce phases. While most map tasks are preemptive and parallelizable, the reduce tasks typically are not easily decomposed and often become a bottleneck due to constraints of data locality and task complexity. By assuming that reduce tasks are non-parallelizable, we study offline scheduling of minimizing makespan and minimizing total completion time, respectively. Both preemptive and non-preemptive reduce tasks are considered. On makespan minimization, for preemptive version we design an algorithm and prove its optimality, for non-preemptive version we design an approximation algorithm with the worst ratio of $\frac{3}{2} - \frac{1}{2h}$ where h is the number of machines. On total complete time minimization, for non-preemptive version we devise an approximation algorithm with worst case ratio of $2-\frac{1}{h}$, and for preemptive version we devise a heuristic. We confirm that our algorithms outperform state-of-art schedulers through experiments.**

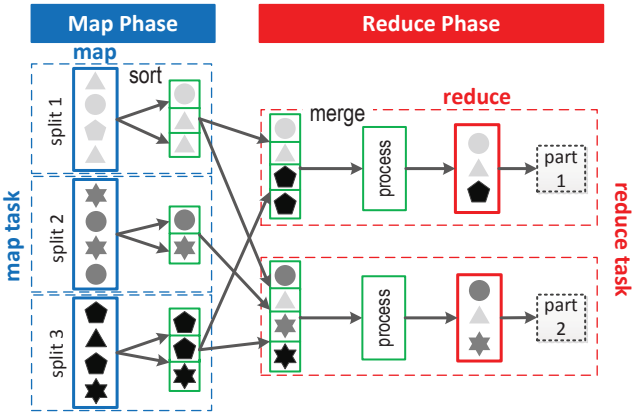


Figure 1: MR execution sequence

I. Introduction

MapReduce [1] (MR) is a popular batch data processing framework that enables massive amounts of such data to be organized and then analyzed on a distributed cluster of nodes preserving the principles of data locality and bringing computation closer to where the data resides.

MR processing is devided into two phases: the *map* phase and the *reduce* phase (Fig. 1). Correspondingly each job is divided into two types of tasks: map tasks that take the raw or processed data as the input and output key-value pairs, and reduce tasks that take the pairs output by the map tasks and compute the final results. In detail, the input to a job is divided into fixed-size *splits*, and for each split one map task is created. The system sorts the output of map tasks, and then merges them and passes them to the reduce tasks as the input. Thus, having many splits means the processing time for the map tasks is often shorter than that of the reduce tasks. Moreover, if the splits are moderately small, the processing is better load-balanced when the splits are processed in parallel.

Though "distributed" is one of MR's hallmarks, this framework has a centralized scheduler, i.e. *master node* that controls the execution and machine allocations for the tasks in arriving jobs. Besides FIFO, the classic yet simple approach, Fair Scheduler and Capacity Scheduler have been realized in MR implementation. The Fair Scheduler [2] aims to give a fair share of the cluster capacity to every user, on average the users will get similar cluster resources. It places jobs in pools and supports preemption. The Capacity Scheduler [3] uses queues like the pools using in Fair Scheduler, but jobs within each queue are scheduled using FIFO scheduling with priorities.

Beyond the pragmatic schedulers, some works viewed the MR scheduling in theoretic ways. The foundation of all the theoretic analysis is the *dependence* between the map and reduce tasks, i.e. for any job, none of its reduce tasks can be started unless all of its map tasks have been finished. Chen et al [4] treated the sort and merge operations as an independent and integrated phase in such theoretical analysis. However, a sort operation depends on its previous map output, while a merge operation proceeds the reducer's output. Thus, sort can be broadly a map component and merge can be part of reduce [5]. In this effort, we study MR as a two stage framework.

Probability based methods [6],[7],[8] and linear programming based methods [4],[9] are so far the hottest techniques for MR scheduling study. The effectiveness of probability based methods depends on the heavy-tailed property of job processing times, and this kind of methods have been used to mitigate the starvation problem. Linear programming based methods,

✉ Corresponding Author: Deying Li. Email: deyingli@ruc.edu.cn
This work was supported in part by NSF grants CNS-1016320 and CCF-0627233, and NSFC grants 61070191 and 91124001.
978-1-4799-3360-0/14/ $31.00 ©2014 IEEE

on the other hand, are used mainly on optimizing the weighted flow time (completion time). Compared to the previous two popular methods, combinatorial optimization based methods need more study. Note in MR a job consists of a batch of tasks, hence the scheduling has become a novel generalization of the classic two stage scheduling problem. Except the bounds on flow time obtained in [10], few theoretical guarantees generated by combinatorial optimization have been presented, it is urgent to study MR scheduling from this perspective.

In our model, stronger than existing ones, we assume that map tasks are parallelizable but reduce tasks are not, i.e. one map task can be executed on multiple machines at one time but one reduce task can only be executed on one machine at one time. Our model maintains the parallelism on job level but gives a constraint on the task level's parallelism. Under our assumption, many existing preemptive methods are inapplicable, since arbitrarily splitting reduce tasks to the machines often leads to some task being processed by many machines simultaneously. The performance analysis techniques for their schedulers are not suitable either. Hence it needs novel and careful consideration to solve our problem.

Our main contributions in the paper are:

1) We study the MR Scheduling problem of minimizing Makespan, i.e. the maximum completion time. We give the uniform lower bound for preemptive and non-preemptive reduce tasks. For the preemptive version we design an optimal algorithm. For the non-preemptive version we design an approximation algorithm and prove that it has the worst ratio of $\frac{3}{2}-\frac{1}{2h}$ where h is the number of machines.
2) We study the MR scheduling problem of minimizing total completion time of all jobs. For the non-preemptive version, we propose an approximation algorithm with worst case ratio of $2-\frac{1}{h}$. For the preemptive version, we extend the classic *McNaughton's Wrap-around Rule* and propose a heuristic.
3) Through simulations we confirm that on both makespan minimization and total completion time minimization, our algorithms outperform state-of-art schedulers.

II. Related Work

The makespan minimization problem of two stage (or machine) scheduling was first studied by Johnson in [11]. He assumed that each machine can process one job at a time, and each job must be finished through machine one before it is passed to machine two. A simple optimal decision rule was obtained under the assumption that the setup time plus work time for each job on each machine is known. He also discussed a restrict case of three machine problem. In [12], Ali et al. proposed a two-stage scheduling problem with bicriteria: makespan and mean completion time. They proved this problem is NP-hard, and proposed heuristics to solve it.

For minimizing the makespan for a batch of jobs in MR, several works have also been presented. [13] proposed a framerwork *ARIA* to meet the performance goals like deadline of Hadoop jobs. They estimated the makespan of a set of jobs and devised the methods of allocating resources according to the makespan estimation. In [14], Luo et al. presented a hierarchical MR framework to gather resources from clusters and run jobs accoss them. The framework splits the map tasks to different clusters and balances the workload in a naive way. They showed that the framework reduced the entire execution's makespan experimentally. Verma et al. in their work [15] mentioned that the order in which MR jobs are processed has a significant impact on the makespan, they proposed a heuristic named *BalancePools* that achieves $15\% - 38\%$ makespan improvements in their experiments.

Schedulers that aim to reduce the (weighted) total completion time in MR framework has also been studied. In [9], the authors devised *OFFA*, a 3-approximation algorithm for offline scheduling. The author also devised the online version algorithm that achieved 30% improvements compared to FIFO. However, all their theoretical and experimental results were obtained by assuming that no precedences exists between tasks, which ignored a basic MR feature that the reduce tasks can only begin after the map tasks are done. In [4], based on [9] the authors added the precedence between map and reduce tasks, and they also claimed the shuffle phase should be considered independently. They devised offline approximation algorithms with guaranteed ratios. However frequent computations are required for linear programming based method.

In [6], Tan et al. observed the starvation problem in Fair Scheduler, and designed Coupling Scheduler that jointly schedules map and reduce tasks by coupling their progresses to mitigate this problem. [7] studied the capacity region and delay performance in MR, and presented a new queueing architecture. They proposed a scheduling algorithm joined by the Shortest Queue and Maxweight policies. The authors proved their algorithm is heavy-traffic optimal. In [8], based on a well-known fact that no online scheduling algorithm can achieve a constant competitive ratio on optimizing total completion time, the authors defined their criteria named efficiency ratio. They assumed that each map task has 1 unit workload. Based on *SRPT*(Shortest Remaining Processing Time), the authors designed algorithms and claimed that in preemptive scenario their algorithm reaches a small efficiency ratio.

Moseley et al. in [10] considered MR scheduling based on the classical two-stage flow shop problem, and studied the flow time minimization. When the machines are identical and each job consists of multiple tasks, they proposed an offline 12-approximation algorithm and an online $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm. When the machines are unrelated and each job consists of one map task and one reduce task, they proposed an offline 6-approximation algorithm and an online $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^4})$-competitive algorithm.

III. System Model and Problem statement

As mentioned in previous work that the map task has a very small size comparing to the reduce task, thus, we assume that the map task is *fractional*, which means that each job can be arbitrarily split between the machines, and parts of the same job can be processed on different machines in parallel.

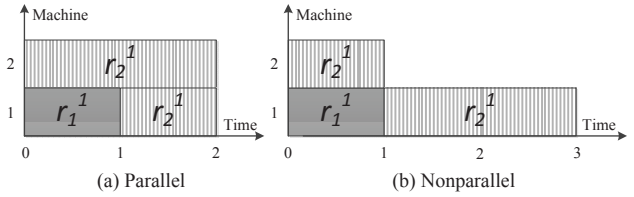


Figure 2: Parallel and nonparallel tasks

For the reduce task, we address both non-preemptive and preemptive version. If *preemption* is allowed, task may be preempted into a few pieces and these pieces are assigned to possibly different machines. Nevertheless, in reduce phase, the minimum unit for executing in parallel is a job. No matter if preemption is allowed or not, a single reduce task can not be executed in parallel, i.e. no reduce task can be executed on multiple machines at the same time. In Fig. 2, suppose r_1^1 and r_2^1 are two preemptive tasks of a job J_1. In Fig. 2(a), a task can be processed in parallel and J_1 is finished at time 2, in Fig. 2(b), no task is allowed to be processed in parallel thus J_1 is done until time 3. In job level J_1 is processed in parallel in both cases.

In existing works, to easily find the solution each reduce task is assumed to be processed in parallel freely, however this is not true. Because compared to map jobs, reduce jobs are much more complex, and there are many operations that must be processed in sequence. Hence in this paper we assume in reduce phase a job is the minimum level of parallel processing, and every reduce task cannot be processed on multiple machines simultaneously. Both objectives of minimizing the makespan and the total completion time of all jobs are considered.

The problem can be formally described as follows. Given a set of jobs $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$, which must be processed on h identical parallel machines $\{\sigma_1, ..., \sigma_h\}$. Let $\mathcal{M}_j$ and $\mathcal{R}_j$ be the sets of map tasks and reduce tasks of J_j for any $1 \le j \le n$ separatively, and denote $|\mathcal{M}_j|$ and $|\mathcal{R}_j|$ the numbers of tasks in $\mathcal{M}_j$ and $\mathcal{R}_j$ respectively. Let C_j be the completion time of job J_j, our goal is to minimize the makespan, i.e., maximum completion time $\max_{1\le j\le n}\{C_j\}$ and the total completion time $\sum_{j=1}^{n} C_j$. For the sake of conciseness, we denote makespan minimization on P machines by $P||C_{\max}$, and denote the total completion time minimization on P machines by $P||\sum C_j$.

The performance of an offline approximation algorithm A is measured by its *worst-case ratio*, which can be defined as the smallest number ρ such that for any $\mathcal{J}$, $Z_A(\mathcal{J}) \le \rho \cdot Z_{OPT}(\mathcal{J})$, where $Z_A(\mathcal{J})$ (or in short Z_A) denotes the objective function value produced by A and $Z_{OPT}(\mathcal{J})$ (or shortly Z_{OPT}) denotes the optimal objective function value.

Throughout the rest of this paper, the following notations are used. Denote by $p(\mathcal{S})$ the total size of all tasks in set $\mathcal{S}$. Let $r_i^{t_i}$ be the i-th largest reduce task in the union set $\cup_{1\le j\le n}\mathcal{R}_j$, where t_i denotes that the task $r_i^{t_i}$ belongs to $\mathcal{R}_{t_i}$, the reduce task set of J_{t_i}. For an arbitrary task r_k, we use $|r_k|$ to denotes its length.

• **Computation Complexity**: Under our model: 1) In the preemptive version, i.e. preemptions on reduce tasks are allowed, makespan minimization is in $\mathbf{P}$ when all jobs arrive in the beginning, and we will give a polynomial time optimal algorithm. The computation complexity of total completion time optimization, on the other hand, is open when all jobs arrive at the start. 2) In the non-preemptive version, both makespan minimization and total completion time minimization are $\mathbf{NP}$-hard. To prove the $\mathbf{NP}$-hardness of makespan minimization, simply assume 2 same machines exist and only one job J_1 exists, $\mathcal{M}_1 = \emptyset$ and $\mathcal{R}_1 = \{a_1, ..., a_n\}$ where a_i is an integer and $\sum_i^n a_i$ is even. A scheduling with makespan $\sum_i^n a_i/2$ exists if and only if $\{a_1, ..., a_n\}$ has a partition. Further, in this instance the total completion time equals to the makespan, thus total completion time minimization is also $\mathbf{NP}$-hard.

IV. MRSM

In this section, we consider the *MR Scheduling problem of minimizing Makespan* (MRSM). Both preemptive and non-preemptive versions are considered. We first give the lower bound of optimal objective value below.

Theorem 1. *No matter if preemption is allowed or not, the optimal objective value of problem MRSM is at least*

$$LB = \max\{\frac{\sum_{j=1}^{n} p(\mathcal{M}_j \cup \mathcal{R}_j)}{h}, \max_{1\le i\le h-1} f_i\},$$

where

$$f_i = \frac{p(\cup_{1\le j\le i}\mathcal{M}_{t_j}) + \sum_{j=1}^{i} r_j^{t_j} + (h-i)r_i^{t_i}}{h}.$$

Proof: It is trivial that the objective value is at least $\frac{\sum_{j=1}^{n} p(\mathcal{M}_j \cup \mathcal{R}_j)}{h}$ according to the average load of h machines for all map and reduce tasks. We next show that it is at least f_i for any $1 \le i \le h-1$.

For any $1 \le i \le h-1$, let us consider the optimal schedules for tasks in $\mathcal{J}_i = \{r_1^{t_1}, r_2^{t_2}, \cdots, r_i^{t_i}\} \cup (\cup_{1\le j\le i}\mathcal{M}_{t_j})$. Denote by Z_i^* the makespan of optimal schedule for $\mathcal{J}_i$. We first show that there exists an optimal schedule σ^* in which each reduce task $r_j^{t_j}, 1 \le j \le i$ is non-preemptively processed on j-th machine during time interval $I_j = [Z_i^* - r_j^{t_j}, Z_i^*]$ as shown in Fig. 3. Note that the length of the interval I_j is $|I_j| = r_j^{t_j}$.

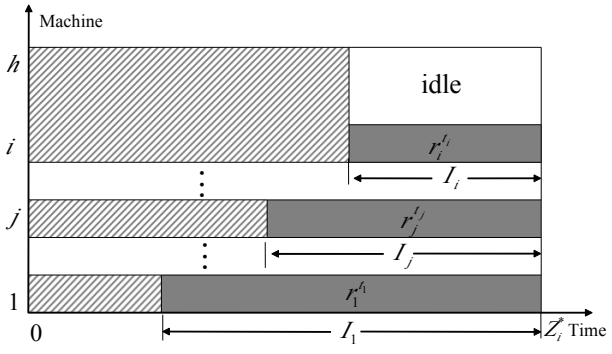


Figure 3: An optimal schedule σ^* for $\mathcal{J}_i$

In fact, for any optimal schedule σ for $\mathcal{J}_i$, we can transform it into σ^* by the following steps.

1) For any $r_j^{t_j}$, suppose that some of it is processed on k-th machine during time interval I, $k \neq j$. It is easy to obtain that there must not exist any part of $r_j^{t_j}$ on j-th machine during I because of non-overlapping rule of preemption: each task can not be processed on different machines at the same time. Thus, we can exchange the tasks on two machines in the same interval I. By this kind of exchange, all parts of $r_j^{t_j}$ can be moved to j-th machine.
2) For each j-th machine, $1 \leq j \leq i$, there might be some map tasks between two different parts of $r_j^{t_j}$. All parts of $r_j^{t_j}$ can be moved into the interval I_j via exchanging each part and its neighboring map tasks. This exchange is feasible because the reduce task $r_j^{t_j}$ is processed after finishing all map tasks on the same machine.

In the optimal schedule σ^* as shown in Fig. 3, the reduce task $r_j^{t_j}$ is processed on time interval I_j on the j-th machine and all map tasks in $\cup_{1\leq j\leq i}\mathcal{M}_{t_j}$ must be finished before the reduce task $r_i^{t_i}$. It implies that there is no task in the interval $I_i = \min_{1\leq j\leq i} I_j$ on any one of the last $h-i$ machines. Hence, we conclude that the total size of all tasks in $\mathcal{J}_i$ is

$$\begin{aligned} & p(\cup_{1\leq j\leq i}\mathcal{M}_{t_j}) + \sum_{j=1}^{i} r_j^{t_j} \\ \leq\ & hZ_i^* - (h-i)|I_i| \\ =\ & hZ_i^* - (h-i)r_i^{t_i}, \end{aligned}$$

that is,

$$\begin{aligned} Z_i^* &\geq \frac{p(\cup_{1\leq j\leq i}\mathcal{M}_{t_j}) + \sum_{j=1}^{i} r_j^{t_j} + (h-i)r_i^{t_i}}{h} \\ &= f_i. \end{aligned}$$

Note that $\mathcal{J}_i \subseteq \mathcal{J}$ for any $1 \leq i \leq h-1$, we obtain that the optimal objective value is at least $Z_i^* \geq f_i$. ■

A. McNaughton's wrap-around rule

Before we further introduce our algorithm, we give a simple algorithm named *McNaughton's wrap-around rule* [16]. Given a set of tasks $\{r_1, r_2, ..., r_m\}$ and at most h same machines, this algorithm creates an optimal schedule for makespan minimization with preemptions, which has a length of $D = \max\{\sum_i^m |r_i|/h, \max_i r_i\}$. The tasks are sorted and then placed on the machines. The rule fills machine σ_i up until load D before it starts machine σ_{i+1}. A task r_i may be split into two parts, the first part is executed on the last e units of time of σ_i, while the second part is executed on the first $|r_i|-e$ units of time on σ_{i+1}, for some e. Because there are no more than $h \cdot D$ units of processing, and since $D - e \geq |r_i| - e$ for any e, a task is scheduled on at most one machine at any time.

Note that *McNaughton's wrap-around rule* creates the schedule machine by machine, rather than over time. Although machine $\{1, .., h\}$ can be used for the schedule, when $\max_i r_i > \sum_i^m |r_i|/h$, the load on the last scheduled machine may be smaller than D, and some machine may have no task assigned on it, i.e. its load is 0.

B. An optimal preemptive algorithm

In this subsection, we consider the optimal preemptive algorithm for problem MRSM. Note that our goal is to minimize the maximum completion time of all jobs, one obvious ideal is to schedule all map tasks first then all reduce tasks, which can dodge the dependence between the map and reduce tasks. Thus, the problem becomes classic parallel machine scheduling problem of minimizing makespan[16]. It is easy to obtain the optimal makespans for the map and reduce tasks, respectively. However, the sum of two optimal makespans may not be the optimal makespan for MRSM. Let us consider an instance $\mathcal{J}$ with two jobs where $\mathcal{M}_1$ contains N tasks, $\mathcal{R}_1 = \emptyset$, $\mathcal{M}_2 = \emptyset$ and $\mathcal{R}_2$ only contains one task with size of $\frac{N}{h-1}$. Clearly, the makespan produced is $\frac{N}{h} + \frac{N}{h-1}$ if we scheduling all map tasks first and followed by all reduce task, while the optimal makespan is $\frac{N}{h-1}$ by scheduling all map tasks on $h-1$ machines and the reduce task on one machine independently.

We propose an optimal linear time algorithm named *PreeMPTive*, and we call it *PMPT* for short in the following. In MR scheduling, all map tasks of each job must be finished before processing any of its reduce tasks. Conversely however, for ease of description, in our algorithm, we schedule the reduce tasks first then its corresponding map tasks. Finally, we output the schedule in reverse-chronological order and thus it becomes a feasible solution. Our algorithm can be described in detail as follows.

Algorithm *PMPT* can be illustrated in Fig. 4.

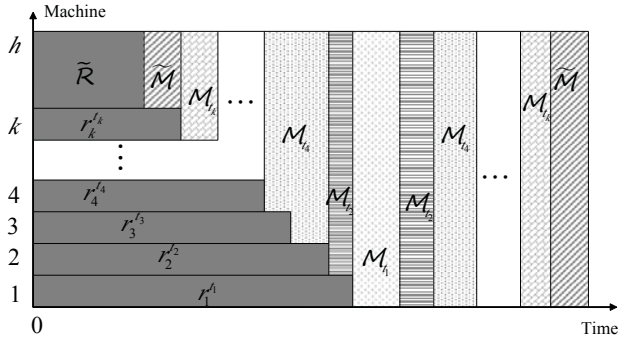


Figure 4: Optimal schedule produced by *PMPT* (before reversing the execution sequence)

Remark. Note that there is no $\mathcal{M}_{t_3}$ in Fig. 4, which represents that reduce task $r_3^{t_3}$ and one of $r_1^{t_1}$ and $r_2^{t_2}$ belong to the same job, that is, $t_3 = t_1$ or $t_3 = t_2$.

Let Z_{PMPT} be the makespan produced by algorithm *PMPT*, then we have the following theorem.

Theorem 2. *$Z_{PMPT} = LB$ and thus Algorithm 1 is optimal.*

Proof: Suppose the algorithm stops at step 3, by our algorithm, all reduce tasks are assigned to h machines by *McNaughton's Wrap-around rule*, which can generate a schedule such that the loads of all machines are the same, because the largest task is not greater than the average value of the total

size of all tasks, i.e., $r_1^{t_1} \leq \frac{p(\cup_{1\leq j\leq n}\mathcal{R}_j)}{h}$. For all map tasks, it is easy to evenly assign them on h machines. Thus, we obtain

$$\begin{aligned} Z_{PMPT} &= \frac{p(\cup_{1\leq j\leq n}\mathcal{R}_j)}{h} + \frac{p(\cup_{1\leq j\leq n}\mathcal{M}_j)}{h} \\ &= \frac{\sum_{j=1}^{n} p(\mathcal{M}_j \cup \mathcal{R}_j)}{h}. \end{aligned}$$

We next consider the case where our algorithm stops at step 24. Denote by c_i the completion time produced by our algorithm after scheduling the map tasks in $\mathcal{M}_{t_i}$, $1 \leq i \leq k$. We claim that

$$c_i = \max_{1\leq j\leq i} f_j. \tag{1}$$

We use induction to prove (1). For $i = 1$, by step 13, it is easy to obtain that $c_1 = r_1^{t_1} + \frac{p(\mathcal{M}_{t_1})}{h} = f_1$. Suppose (1) holds for $i-1$. We then consider the case of i. If $\mathcal{M}_{t_i} \subseteq \mathcal{S}_{i-1}$, i.e., the map tasks in $\mathcal{M}_{t_i}$ have been assigned before, then we have $c_i = c_{i-1}$ and thus (1) holds trivially. On the other hand, we need to schedule the map tasks $\mathcal{M}_{t_i}$ by step 13. If all these tasks can be assigned in the idle time slots during interval $[r_i^{t_i}, r_1^{t_1}]$, we still have $c_i = c_{i-1}$. Otherwise, we can obtain that

$$\begin{aligned} c_i &= r_i^{t_i} + \frac{p(\cup_{1\leq j\leq i}\mathcal{M}_{t_j}) + \sum_{j=1}^{i} r_j^{t_j} - i r_i^{t_i}}{h} \\ &= f_i, \end{aligned}$$

which implies (1) holds.

Finally, we consider the makespan of our algorithm after scheduling all tasks by step 23. If all the tasks in $\cup_{1\leq j\leq n}\mathcal{M}_j \setminus \mathcal{S}_k$ can be processed in the idle time slots before $r_1^{t_1}$, then we have

$$Z_{PMPT} = c_k.$$

Otherwise, it is not hard to get that

$$Z_{PMPT} = \frac{\sum_{j=1}^{n} p(\mathcal{M}_j \cup \mathcal{R}_j)}{h}.$$

Hence, we conclude that $Z_{PMPT} = LB$ by Theorem 1 and thus the algorithm is optimal. ■

C. A non-preemptive algorithm

In this subsection, we consider the non-preemptive algorithm for problem MRSM. Based on algorithm *PMPT*, we obtain an algorithm named *Non-PreeMPTive* (*NPMPT*) with worst case of $\frac{3}{2} - \frac{1}{2h}$.

Remark Note that there is no preemption in the schedule for $\mathcal{J}_h$ by algorithm *PMPT*, then our algorithm is non-preemptive.

Theorem 3. *The worst case ratio of Algorithm 2 is at most* $\frac{3}{2} - \frac{1}{2h}$.

Proof: Let r be the last finished reduce task and its start time is s, then the makespan produced by algorithm *NPMPT* is $s + r$. If $r \in \mathcal{J}_h$, that is, it is scheduled by step 2, then we obtain optimal makespan by algorithm PMPT. Thus we assume that r is not in $\mathcal{J}_h$, which implies that the optimal makespan is at least $r_h^{t_h} + r \geq 2r$. In addition, there is no idle time in the schedule by the algorithm, we can conclude that the optimal makespan is at least $\frac{hs+r}{h} = s + \frac{r}{h}$. Hence

$$\frac{s+r}{\max\{2r, s + r/h\}} \leq \frac{3h-1}{2h}.$$

The proof is complete. ■

As mentioned in previous subsection, it is easy to obtain an algorithm which schedule all map tasks first then reduce tasks, denoted by *FMFR*. The map tasks can be assigned evenly to h machines. The reduce tasks can be scheduled by any algorithm

Algorithm 1 PMPT

input: a list of jobs $J_1, J_2, ..., J_n$;
output: the schedule.
Phase one:
1: Select the first h largest reduce tasks $r_1^{t_1}, r_2^{t_2}, \cdots, r_h^{t_h}$ in $\cup_{1\leq j\leq n}\mathcal{R}_j$.
2: **if** $r_1^{t_1} \leq \frac{p(\cup_{1\leq j\leq n}\mathcal{R}_j)}{h}$ **then**
3: Schedule all reduce tasks by *McNaughton's Wrap-around Rule*, assign all map tasks evenly to h machines, **goto** step 24.
4: **else**
5: Let k ($1 \leq k \leq h-1$) be the largest number such that $r_k^{t_k} > \frac{p(\cup_{1\leq j\leq n}\mathcal{R}_j) - \sum_{j=1}^{k} r_j^{t_j}}{h-k}$;
6: **for** $1 \leq i \leq k$ **do**
7: Schedule the task $r_i^{t_i}$ on i-th machine from time 0;
8: **end for**
9: Use *McNaughton's wrap-around rule* to schedule all the remainder reduce tasks, i.e. $\cup_{1\leq j\leq n}\mathcal{R}_j \setminus \{r_1^{t_1}, \cdots, r_k^{t_k}\} \doteq \widetilde{\mathcal{R}}$ on the left $h-k$ machines;
10: **end if**
Phase two:
11: $i \leftarrow 1$, $\mathcal{S}_i \leftarrow \mathcal{M}_{t_i}$;
12: **while** true **do**
13: Assign map tasks in $\mathcal{M}_{t_i}$ to the idle time slots in interval $[r_i^{t_i}, r_1^{t_1}]$, once all these idle time slots are occupied, assign the remainder map tasks to the time slots as early as possible from time $r_1^{t_1}$;
14: **do**
15: $i \leftarrow i + 1$;
16: **while** $\mathcal{M}_{t_i} \subseteq \mathcal{S}_{i-1}$;
17: **if** $i > k$ **then**
18: **break**;
19: **else**
20: $\mathcal{S}_i \leftarrow \mathcal{S}_{i-1} \cup \mathcal{M}_{t_i}$;
21: **end if**
22: **end while**
23: Assign all remainder map tasks in $\cup_{1\leq j\leq n}\mathcal{M}_j \setminus \mathcal{S}_k \doteq \widetilde{\mathcal{M}}$ to the idle time slots before $r_1^{t_1}$, once all idle time slots are occupied, assign the remainder map tasks to the time slots as early as possible from time $r_1^{t_1}$;
24: Reverse the executed sequence of obtained schedule while keep the executed machines the same, **stop**.

Algorithm 2 NPMPT

input: a list of jobs $J_1, J_2, ..., J_n$;
output: the schedule.

1: Select the first h largest reduce tasks $r_1^{t_1}, r_2^{t_2}, \cdots, r_h^{t_h}$ in $\cup_{1\le j\le n}\mathcal{R}_j$;
2: For $\mathcal{J}_h = \{r_1^{t_1}, r_2^{t_2}, \cdots, r_h^{t_h}\} \cup (\cup_{1\le j\le h}\mathcal{M}_{t_j})$, schedule r_i on machine σ_i, and schedule $\cup_{1\le j\le h}\mathcal{M}_{t_j}$ by PMPT (Alg. 1). If there are some idle times between map and reduce tasks, then move the reduce tasks earlier such that no idle time exists;
3: For the remainder map tasks, assign them to h machines evenly before the tasks in $\cup_{1\le j\le h}\mathcal{M}_{t_j}$;
4: For the remainder reduce tasks, schedule them to the least load machine one by one after the first h tasks.

for problem $P||C_{\max}$[17]. However, the worst case ratio of *FMFR* is at least $2-\frac{1}{h}$. We still take the instance $\mathcal{J}$ mentioned in previous subsection as an example, we can obtain that the makespan produced by *FMFR* is $\frac{N}{h}+\frac{N}{h-1}$ while the optimal makespan is $\frac{N}{h-1}$ as long as N is an sufficient large number, which follows that the worst case of *FMFR* is $2-\frac{1}{h}$.

V. MRST

In this section, we consider the *MR Scheduling problem of minimizing Total completion time of all jobs* (MRST).

A. A non-preemptive algorithm

For the variant where the reduce tasks are not allowed to be preempted, based on *Shortest Processing Time* (*SPT*)[18] we propose an algorithm named *Non-preemptive Shortest Processing Time* (*NSPT*), with worst case ratio of $2-\frac{1}{h}$.

Note that algorithm SPT is optimal for the single machine scheduling problem $1||\sum C_j$, where each job has one task and no precedence constraints. If, like the map task, each reduce task has a unit size, then we can evenly assign both map and reduce tasks to h machines, which is equivalent to the problem $1||\sum C_j$ on a h-speed machine. Hence, we can use SPT to generate an optimal schedule σ_{SPT} for the problem $1||\sum C_j$ on a h-speed machine, where each job J_i has a size of $p(\mathcal{M}_i\cup\mathcal{R}_i)$.

Algorithm 3 NSPT

input: a list of jobs $J_1, J_2, ..., J_n$;
output: the schedule.

1: Sort jobs J_i in non-decreasing order of $p(\mathcal{M}_i\cup\mathcal{R}_i)$;
2: **for** $\forall J_i, 1\le i\le n$ **do**
3: Assign all map tasks in $\mathcal{M}_i$ so that they can be finished as early as possible;
4: Assign each reduce task in $\mathcal{R}_i$ to the least loaded machine;
5: **end for**

Let $C_i^* = \frac{1}{h}\sum_{j=1}^{i} p(\mathcal{M}_i\cup\mathcal{R}_i)$. Denote $r_i^{\max}$ and C_i^{NSPT} be the largest task in $\mathcal{R}_i$ and the completion time of job J_i in *NSPT*, i.e. Algorithm 3. Let $Z_{\sigma_{SPT}}$ denote the objective value of schedule σ_{SPT}. Denoted by Z_{NSPT} and Z_{OPT} the objective values optimal schedule and the schedule produced by algorithm *NSPT* for problem MRST, respectively. It is easy to obtain the following results.

Lemma 1. *(i)* $Z_{OPT}\ge Z_{\sigma_{SPT}} = \sum_{i=1}^{n} C_i^*$*; (ii)* $C_i^{NSPT}\le C_i^* + \frac{h-1}{h}r_i^{\max}$ *for any* $1\le i\le n$.

Proof: (i) Since all jobs are sorted in non-decreasing order of $p(\mathcal{M}_i\cup\mathcal{R}_i)$, we have the completion time of each job is C_i^* and thus the total completion time of schedule σ_{SPT} is $\sum_{i=1}^{n} C_i^*$. Clearly, the objective value of problem MRST is not less than that of problem $1||\sum C_j$ on a h-speed machine, i.e., $Z_{OPT}\ge Z_{\sigma_{SPT}}$.

(ii) For any $1\le i\le n$, let r_i be the last finished reduce task in $\mathcal{R}_i$ and its start time is s_i, then we have $C_i^{NSPT} = s_i + r_i$. According to the rule of algorithm, we can conclude that there is no idle time on each machine before time s_i, then we have $s_i \le \frac{\sum_{j=1}^{i} p(\mathcal{M}_i\cup\mathcal{R}_i)-r_i}{h}$. Together with the fact that $r_i\le r_i^{\max}$, we have

$$\begin{aligned} C_i^{NSPT} &= s_i + r_i \\ &\le \frac{\sum_{j=1}^{i} p(\mathcal{M}_i\cup\mathcal{R}_i)-r_i}{h} + r_i \\ &\le C_i^* + \frac{h-1}{h}r_i^{\max}. \end{aligned}$$

■

Theorem 4. $Z_{NSPT}\le(2-\frac{1}{h})Z_{OPT}$.

Proof: By Lemma 1(ii), we have

$$\begin{aligned} Z_{NSPT} &\le \sum_{i=1}^{n} C_i^{NSPT} \\ &\le \sum_{i=1}^{n} C_i^* + \frac{h-1}{h}\sum_{i=1}^{n} r_i^{\max}. \end{aligned}$$

It is clear that $Z_{OPT}\ge\sum_{i=1}^{n} r_i^{\max}$ and $Z_{OPT}\ge\sum_{i=1}^{n} C_i^*$ from Lemma 1(i). Hence, Theorem 4 stands. ■

B. A preemptive heuristic

Note that McNaughton[16] has shown that preemption cannot reduce $\sum C_j$ for problem $P||\sum C_j$, however, it is not true in our problem. In fact, we can give a very simple instance $\mathcal{J}'$ including only one job with $\mathcal{M}_1=\emptyset$ and $\mathcal{R}_1=\{1,1,1\}$. Clearly the completion time is 2 if preemption is not allowed while the objective value is $3/2$ if preemption is allowed. Hence, we design a preemptive algorithm to minimize the total completion time.

Synchronic start is the situation where all the machines start synchronically. Fig. 5 plots that all machines start at time 0, an scenario of *synchronic start*. We present another definition *step start*, which means there exist at least two machines that start (become available) in different times. Fig. 6 plots an instance of *step start*.

Since *McNaughton's Wrap-around Rule* only deals with the *Synchronic start*, we design the *Extented Wrap-around Rule*

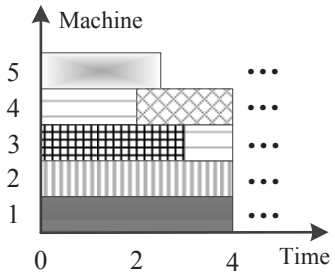


Figure 5: Synchronic start

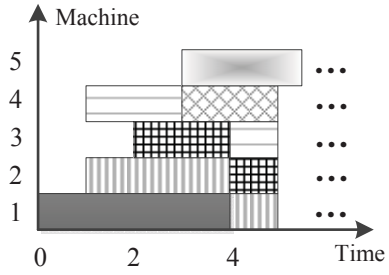


Figure 6: Step start

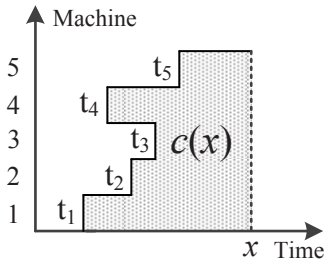


Figure 7: $c(x)$

that can also deal with *step start*. To start with, we define a function $c(x)$ on a given sequence of positive real numbers $(t_1, ..., t_h)$:

$$c(x) = \sum_{i=1}^{h} c_i(x), \quad \text{where}$$

$$c_i(x) = \begin{cases} 0 & x \le t_i \\ x - t_i & x > t_i \end{cases}.$$

Theorem 5. *Given h machines $\{\sigma_1, \sigma_2, ..., \sigma_h\}$ such that σ_i starts at time t_i $(1 \le i \le h)$, and $\mathcal{R}$, a set of $|\mathcal{R}|$ reduce tasks $\{r_1, r_2, ..., r_{|\mathcal{R}|}\}$. Without loss of generality suppose $t_1 \le t_2 \le ... \le t_h$ and $|r_1| \ge |r_2| \ge ... \ge |r_{|\mathcal{R}|}|$. Let*

$$\begin{aligned} x^* &= \arg\min_x \{x | c(x) \ge \sum_{i=1}^{|\mathcal{R}|} |r_i|\} \\ k^* &= \arg\max_k \{k | t_k < x^*\} \\ L &= \max\{x^*, \max_{1 \le i \le k^*} (t_i + r_i)\} \end{aligned} \tag{2}$$

then there is a schedule that assigns $\mathcal{R}$ on $\{\sigma_1, ..., \sigma_{k^*}\}$ which finishes $\mathcal{R}$ before time L on each machine.

We omit the proof of Theorem 5 here. The computation of $\min_x\{x|c(x) \ge \sum_{i=1}^{|\mathcal{R}|} |r_i|\}$ is not straightforward but it can be done in $O(h)$ time, since $c(x) = ix - \sum_{j=1}^{i} r_j$ if $t_i < x \le t_{i+1}$. L in (2) gives us an estimation of the time it needs to schedule the reduce tasks in $\mathcal{R}$ on the machines so that no task is processed on different machines at the same time. Note that Theorem 5 stands for both *synchronic start* and *step start*, however, L may not be optimal for *step start*. In Alg. 4 the tasks are assigned on the machines from time L to each machine's starting time. For example, check a task u with work load 1 and machine σ_1 (starts at t_1), we place u in the interval $[L-1, L]$, while in *McNaughtons wrap-around rule*, u is placed in $[t_1, t_1 + 1]$.

The idea of the algorithm for preemptive reduce tasks is also based on *SPT*, yet its process is like playing jigsaw puzzles. Hence we name it *Jigsaw Shortest Processing Time*, which is *JSPT* in short. *JSPT* assigns each job J_i one by one to finish them in sequence. When $\mathcal{M}_1$, the map tasks in job J_1, whose size $p(\mathcal{M}_1 \cup \mathcal{R}_1)$ is the smallest, has been assigned, the machines can still start synchronically. However, after the assignment of $\mathcal{R}_1$, if $\max\{|r_i^1|\} > \sum_i |r_i^1|/h$, some machines may have no task in $\mathcal{R}_1$ assigned on, and they can start

Algorithm 4 Extended Wrap-around Rule

input: h machines $\{\sigma_1, \sigma_2, ..., \sigma_h\}$, σ_i is available at time t_i for $1 \le i \le h$. A set of reduce tasks $\mathcal{R} = \{r_1, r_2, ..., r_{|\mathcal{R}|}\}$.
output: the schedule.

```
Sort the tasks in R, obtain a new sequence of tasks (r'_1, r'_2, ..., r'_|R|) such that |r'_1| ≥ |r'_2| ≥ ... ≥ |r'_|R||;
Sort the machines, obtain a new sequence of machines (σ'_1, σ'_2, ..., σ'_|R|) such that t'_1 ≤ t'_2 ≤ ... ≤ t'_h;
Obtain L according to formula (2);
i ← 1, j ← 1;
while i ≤ |J| do
    if |r'_i| + the load of σ'_j ≤ L − t'_j then
        place r'_i on σ'_j from the right, i.e. time L, as late as possible;
    else
        assign r'_i the earliest available e load of σ'_j;
        j ← j + 1;
        assign r'_i the latest |r'_i| − e load of σ'_j;
    end if
    i ← i + 1;
end while
```

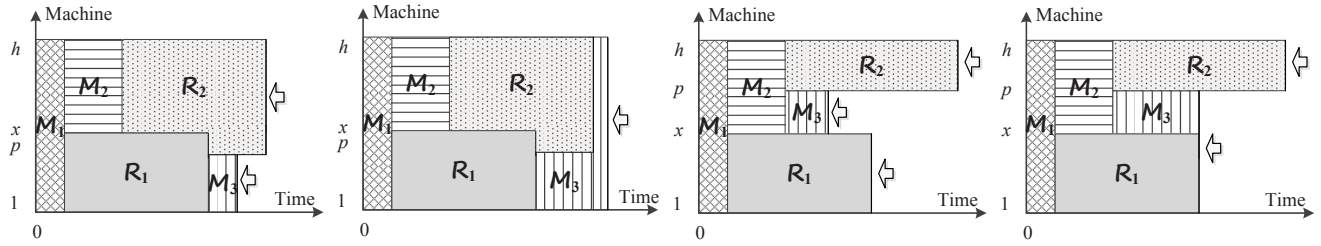
Algorithm 5 JSPT

input: a list of jobs $J_1, J_2, ..., J_n$;
output: the schedule.

```
Sort jobs J_i in non-decreasing order of p(M_i ∪ R_i);
for ∀J_i, 1 ≤ i ≤ n do
    Assign all map tasks in M_i so that they can be finished as early as possible;
    Assign the reduce tasks in R_i using Alg. 4;
end for
```

earlier for scheduling the next job J_2, for example, suppose $\mathcal{R}_1$ is assigned on $\sigma_1 - \sigma_x$, then $\mathcal{M}_2$ should be assigned on $\sigma_{x+1} - \sigma_h$. After $\mathcal{R}_2$ has been assigned by Alg. 4 there are two possible situations: **1**) Its tasks have been placed on more than or equal to $h - x$ machines, i.e. $\sigma_p - \sigma_h$ where $p \le x$, as shown in Fig. 8(a) and Fig. 8(b). **2**) Its tasks have been placed on less than x machines, i.e. $\sigma_p - \sigma_h$ where $p > x$, as shown in Fig. 8(c) and Fig. 8(d).

After $\mathcal{M}_3$ is assigned, 4 typical scenarios may appear



(a) Finish $\mathcal{M}_3$ before $\mathcal{R}_2$ (b) Finish $\mathcal{M}_3$ after $\mathcal{R}_2$ (c) Finish $\mathcal{M}_3$ before $\mathcal{R}_1$ (d) Finish $\mathcal{M}_3$ after $\mathcal{R}_1$

Figure 8: Possible situations before scheduling $\mathcal{R}_3$

according to its load. In 8(c), we see that machines will be available at 3 different times for scheduling $\mathcal{R}_3$. When more jobs are assigned, the available times of different machines may vary more, which *McNaughtons Wrap-around Rule* can no longer handle. That is the reason why we design Alg. 4 and use it in our heuristic *JSPT*.

VI. Simulation

In this section, we present the simulation results of our algorithms. The efficacy of each of our algorithms is compared to several candidate approaches. To begin with we introduce the schedulers and experimental setting in our evaluation.

A. Simulation Setup

We compare 6 typical schedulers with our algorithms:

• **FIFO**: First in First out. Jobs are served in their orders of arrivals. To compare with our offline algorithms, the jobs arrive at the same time will be executed in random sequence. This strategy is the basis of our comparison throughout this section.

• **Capacity**: The latest scheduler in Hadoop. To simulate this scheduler, we group jobs into several queues and machines into several clusters, and allocate queues to clusters. Within each cluster FIFO is used to process the jobs in a queue.

• **SJF**: Shortest Job First. The tasks in the job with shortest size are served first. The tasks in the same job are executed in random sequence.

• **STF**: Shortest Task First. Different from SJF, STF uses the individual size of a task instead of the total size of tasks in a job to decide the execution sequence.

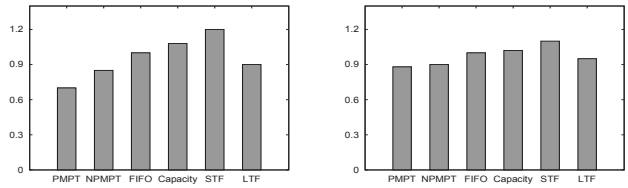
• **LJF** and **LTF**: Longest Job First and Longest Task First. Similar to SJF and STF respectively, except that the longest job and the longest task are severed first respectively.

We test following two types of distributions of the task's length:

Uniform distribution: $X \sim \mathrm{U}(a, b)$, the *Probability Density Function* (PDF) $f(x) = \frac{1}{b-a}$.

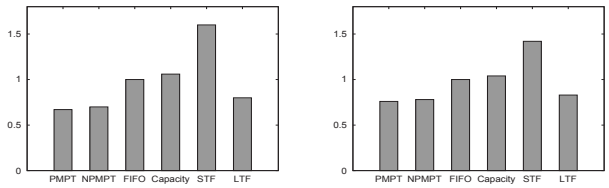
Exponential distribution: $X \sim \mathrm{E}(\lambda)$, PDF $f(x) = \lambda e^{-\lambda x}, x \geq 0$. We fix $\lambda = 1$. To generate the random variable in $[a, b]$, we add a to the sample when each sample is first generated, and if the new sample is greater than b, it is abandoned until new qualified sample is obtained.

We simulate a data center with 50 machines, and check the performances of algorithms on two batches of MapRuduce jobs, one contains 10 jobs and the other contains 100 jobs. To



(a) 10 jobs (b) 100 jobs

Figure 9: Makespan (Uniform Distribution)



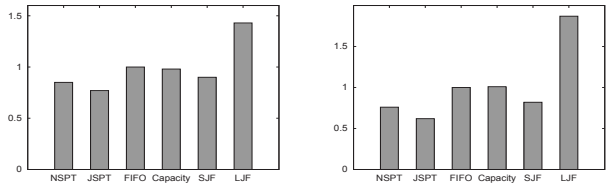
(a) 10 jobs (b) 100 jobs

Figure 10: Makespan (Exponential Distribution)

generate the reduce and map tasks for each job J, we do the following: 1) Generate $|\mathcal{M}|$, the number of map tasks uniform randomly from $[20, 40]$. Considering that several map tasks will merge into one reduce task, we set the number of reduce tasks in the same job, i.e. $|\mathcal{R}| = \lceil |\mathcal{M}|/a \rceil$, where a is a real number uniform distributed in $[2, 4]$. 2) Decide a distribution (Uniform or Exponential), and use the same distribution to generate the loads of all tasks. For a map task, generate its load randomly from $[1, 10]$. For a reduce task, generate its load randomly from $[10, 100]$. For all the simulation results, we normalize them to make the result of FIFO equal to 1.

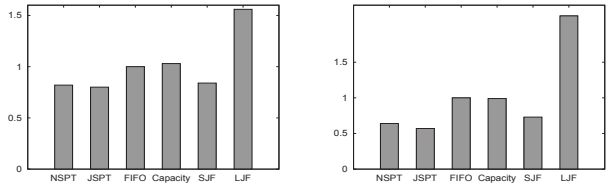
B. Makespan

Fig. 9 and Fig. 10 plot the comparison of our algorithms and other schedulers on makepan minimization. SJF and LJF are not compared since their performances has no difference with FIFO. From the results we see that, when there are more jobs, the difference between preemptive and non-preemptive algorithms gets smaller. The reason is because more jobs leads to more tasks, and more tasks means the ratio between one task and the sum of all tasks are smaller, which means the effect of breaking tasks into factions which reduces the makespan in preemptive scenario gets weak. We also see that when the length of reduce task satisfies exponential distribution, the



(a) 10 jobs (b) 100 jobs

Figure 11: Total completion time (Uniform Distribution)



(a) 10 jobs (b) 100 jobs

Figure 12: Total completion time (Exponential Distribution)

difference between preemptive and non-preemptive algorithms are smaller, comparing the difference when the length of reduce task satisfies uniform distribution. This happens because in exponential distribution, there are more small tasks, and since the big task can not be processed in parallel whether preemption is allowed or not, preemptions in small tasks bring limited reduction on decreasing makespan. However, our algorithms consistently perform better than other popular schedulers.

C. Total Completion Time

Fig. 11 and Fig. 12 plot the results of total completion time minimization of our algorithms and others. In this scenario STF and LTF are not compared, because these two schedulers break job into tasks and ignore the completion time of a job, which will lead to extremely bad performance on reducing total completion time. The difference between the worst and the best schedulers increases when comparing with the situation of makespan. Some different trends appear. With the increment of number of jobs, the gap between our preemptive algorithm *JSPT* and non-preemptive algorithm grows, and the gap between the worst algorithm *LJF* and other algorithms also increases. Our algorithms are still the best ones.

VII. Conclusion and Discussion

In this paper, besides the precedence between the map and reduce phases in MR framework, we also consider the fundamental distinguishing properties between map and reduce tasks: while most map tasks are simple and easy to parallelize, due to constraints of data locality and task complexity most reduce tasks are not easily decomposed. Hence, we assume that map tasks are preemptive and parallelizable, but none reduce task can be processed in parallel. Both preemptive and non-preemptive reduce tasks are considered, with the goals of minimizing the makespan and total completion time of a batch of MR jobs. On makespan minimization, for preemptive version, we design *PMPT* algorithm and prove it is optimal, for non-preemptive version we design *NPMPT* algorithm and prove that its worst ratio is $\frac{3}{2} - \frac{1}{2h}$, where h is the number of machines. On total complete time minimization, for non-preemptive version we devise *NSPT* algorithm whose worst case ratio is $2 - \frac{1}{h}$, and for preemptive version we devise a heuristic named *JSPT* based on extending *McNaughton's wrap-around rule*. Our algorithms' superior over state-of-art schedulers is verified through experiments.

References

[1] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[2] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.

[3] http://hadoop.apache.org/docs/stable/capacity_scheduler.html.

[4] F. Chen, M. Kodialam, and T. V. Lakshman. Joint Scheduling of Processing and Shuffle Phases in MapReduce Systems. In *IEEE INFOCOM*, 2012.

[5] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[6] J. Tan, X. Meng, and L. Zhang. Performance analysis of coupling scheduler for mapreduce/hadoop. In Albert G. Greenberg and Kazem Sohraby, editors, *INFOCOM*, pages 2586–2590. IEEE, 2012.

[7] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang. Map task scheduling in mapreduce with data locality throughput and heavy-traffic optimality. In *INFOCOM*. IEEE, 2013.

[8] Y. Zheng, N.B. Shroff, and P. Sinha. A new analytical technique for designing provably effcient mapreduce schedulers. In *INFOCOM*. IEEE, 2013.

[9] H. Chang, M. S. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee. Scheduling in mapreduce-like systems for fast completion time. In *INFOCOM*, pages 3074–3082. IEEE, 2011.

[10] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós. On scheduling in map-reduce and flow-shops. In *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures*, SPAA '11, pages 289–298, New York, NY, USA, 2011. ACM.

[11] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:61–68, 1953.

[12] A. Allahverdi and F. S. Al-Anzi. The two-stage assembly flowshop scheduling problem with bicriteria of makespan and mean completion time. *The International Journal of Advanced Manufacturing Technology*, 37(1-2):166–177, 2008.

[13] A. Verma, L. Cherkasova, and R. H. Campbell. Aria: Automatic resource inference and allocation for mapreduce environments. In *International Conference on Autonomic Computing (ICAC)*, Karlsruhe, Germany, 06/2011 2011. IEEE/ACM, IEEE/ACM.

[14] *A Hierarchical Framework for Cross-Domain MapReduce Execution*, San Jose, CA, 06/2011 2011. ACM Press.

[15] A. Verma, L. Cherkasova, and R. H. Campbell. Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance. In *Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '12, pages 11–18, Washington, DC, USA, 2012. IEEE Computer Society.

[16] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[17] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[18] K. R. Baker and D. Trietsch. *Principles of Sequencing and Scheduling*. Wiley Publishing, 2009.