

A New Analytical Technique for Designing Provably Efficient MapReduce Schedulers

Yousi Zheng*, Ness B. Shroff^{*†}, Prasun Sinha[†]

^{*}Department of Electrical and Computer Engineering

[†]Department of Computer Science and Engineering

The Ohio State University, Columbus, OH 43210

Abstract—With the rapid increase in size and number of jobs that are being processed in the MapReduce framework, efficiently scheduling jobs under this framework is becoming increasingly important. We consider the problem of minimizing the total flow-time of a sequence of jobs in the MapReduce framework, where the jobs arrive over time and need to be processed through both Map and Reduce procedures before leaving the system. We show that for this problem for non-preemptive tasks, no on-line algorithm can achieve a constant competitive ratio (defined as the ratio between the completion time of the online algorithm to the completion time of the optimal non-causal off-line algorithm). We then construct a slightly weaker metric of performance called the efficiency ratio. An online algorithm is said to achieve an efficiency ratio of γ when the flow-time incurred by that scheduler divided by the minimum flow-time achieved over all possible schedulers is *almost surely* less than or equal to γ . Under some weak assumptions, we then show a surprising property that, for the flow-time problem, any work-conserving scheduler has a constant efficiency ratio in both preemptive and non-preemptive scenarios. More importantly, we are able to develop an online scheduler with a very small efficiency ratio (2), and through simulations we show that it outperforms the state-of-the-art schedulers.

I. INTRODUCTION

MapReduce is a framework designed to process massive amounts of data in a cluster of machines [1]. Although it was first proposed by Google [1], today, many other companies including Microsoft, Yahoo, and Facebook also use this framework. Currently this framework is widely used for applications such as search indexing, distributed searching, web statistics generation, and data mining.

MapReduce has two elemental processes: *Map* and *Reduce*. For the Map tasks, the inputs are divided into several small sets, and processed by different machines in parallel. The output of Map tasks is a set of pairs in $\langle \text{key}, \text{value} \rangle$ format. The Reduce tasks then operate on this intermediate data, possibly running the operation on multiple machines in parallel to generate the final result.

Each arriving job consists of a set of Map tasks and Reduce tasks. The *scheduler* is centralized and responsible for making decisions on which task will be executed at what time and on which machine. The key metric considered in this paper is the *total delay in the system* per job, which includes the time it takes for a job, since it arrives, until it is fully processed. This includes both the delay in waiting before the first task in the

job begins to be processed, plus the time for processing all tasks in the job.

A critical consideration for the design of the scheduler is the *dependence* between the Map and Reduce tasks. For each job, the Map tasks need to be finished before starting any of its Reduce tasks¹ [1], [3]. Various scheduling solutions have been proposed within the MapReduce framework [2]–[6], but analytical bounds on performance have been derived only in some of these works [2], [3], [6]. However, rather than focusing directly on the flow-time, for deriving performance bounds, [2], [3] have considered a slightly different problem of minimizing the total completion time and [6] has assumed speed-up of the machines. Further discussion of these schedulers is given in Section II.

In this paper, we directly analyze the performance of total delay (flow-time) in the system. In an attempt to minimize this, we introduce a new metric to analyze the performance of schedulers called *efficiency ratio*. Based on this new metric, we analyze and design several schedulers, which can guarantee the performance of the MapReduce framework.

The contributions of this paper are as follows:

- For the problem of minimizing the total delay (flow-time) in the MapReduce framework, we show that no on-line algorithm can achieve a constant competitive ratio. (Sections III and IV)
- To directly analyze the total delay in the system, we propose a new metric to measure the performance of schedulers, which we call the *efficiency ratio*. (Section IV)
- Under some weak assumptions, we then show a surprising property that for the flow-time problem any work-conserving scheduler has a constant efficiency ratio in both preemptive and non-preemptive scenarios (precise definitions provided in Section III). (Section V)
- We present an online scheduling algorithm called ASRPT (Available Shortest Remaining Processing Time) with a very small (less than 2) efficiency ratio (Section VI), and show that it outperforms state-of-the-art schedulers through simulations (Section VII).

II. RELATED WORK

In Hadoop [7], the most widely used implementation, the default scheduling method is First In First Out (FIFO). FIFO

This work has been supported in part by the Army Research Office MURI Award W911NF-08-1-0238.

¹Here, we consider the most popular case in reality without the Shuffle phase. For discussion about the Shuffle phase, see Section II and [2].

suffers from the well known head-of-line blocking problem, which is mitigated in [4] by using the Fair scheduler.

In the case of the Fair scheduler [4], one problem is that jobs stick to the machines on which they are initially scheduled, which could result in significant performance degradation. The solution of this problem given by [4] is delayed scheduling. However, the fair scheduler could cause a starvation problem (please refer to [4], [5]). In [5], the authors propose a Coupling scheduler to mitigate this problem, and analyze its performance.

In [3], the authors assume that all Reduce tasks are non-preemptive. They design a scheduler in order to minimize the weighted sum of the job completion times by determining the ordering of the tasks on each processor. The authors show that this problem is NP-hard even in the offline case, and propose approximation algorithms that work within a factor of 3 of the optimal. However, as the authors point out in the article, they ignore the dependency between Map and Reduce tasks, which is a critical property of the MapReduce framework. Based on the work of [3], the authors in [2] add a precedence graph to describe the precedence between Map and Reduce tasks, and consider the effect of the Shuffle phase between the Map and Reduce tasks. They break the structure of the MapReduce framework from job level to task level using the Shuffle phase, and seek to minimize the total completion time of tasks instead of jobs. In both [3] and [2], the schedulers use an LP based lower bound which need to be recomputed frequently. However, the practical cost corresponding to the delay (or storage) of jobs are directly related to the total flow-time, not the completion time. Although the optimal solution is the same for these two optimization problems, the efficiency ratio obtained from minimizing the total flow-time will be much looser than the efficiency ratio obtained from minimizing the total completion time (details are shown in technical report [8]).

In [6], the authors study the problem of minimizing the total flow-time of all jobs. They propose an $O(1/\epsilon^5)$ competitive algorithm with $(1 + \epsilon)$ speed for the online case, where $0 < \epsilon \leq 1$. However, speed-up of machines is necessary in the algorithm; otherwise, there is no guarantee on the competitive ratio (if ϵ decreases to 0, the competitive ratio will increase to infinity correspondingly).

III. SYSTEM MODEL

Consider a data center with N machines. We assume that each machine can only process one job at a time. A machine could represent a processor, a core in a multi-core processor or a virtual machine. Assume that there are n jobs arriving into the system. We assume the scheduler periodically collects the information on the state of jobs running on the machines, which is used to make scheduling decisions. Such time slot structure can efficiently reduce the performance degeneration caused by data locality (see [4] [9]). We assume that the number of job arrivals in each time slot is i.i.d., and the arrival rate is λ . Each job i brings M_i units of workload for its Map tasks and R_i units of workload for its Reduce

tasks. Each Map task has 1 unit of workload², however, each Reduce task can have multiple units of workload. Time is slotted and each machine can run one unit of workload in each time slot. Assume that $\{M_i\}$ are i.i.d. with expectation \overline{M} , and $\{R_i\}$ are i.i.d. with expectation \overline{R} . We assume that the traffic intensity $\rho < 1$, i.e., $\lambda < \frac{N}{\overline{M} + \overline{R}}$. Assume the moment generating function of workload of arriving jobs in a time slot has finite value in some neighborhood of 0. In time slot t for job i , $m_{i,t}$ and $r_{i,t}$ machines are scheduled for Map and Reduce tasks, respectively. As we know, each job contains several tasks. We assume that job i contains K_i tasks, and the workload of the Reduce task k of job i is $R_i^{(k)}$. Thus, for any job i , $\sum_{k=1}^{K_i} R_i^{(k)} = R_i$. In time slot t for job i , $r_{i,t}^{(k)}$ machines are scheduled for the Reduce task k . As each Reduce task may consist of multiple units of workload, it can be processed in either preemptive or non-preemptive fashion based on the type of scheduler.

Definition 1. A scheduler is called **preemptive** if Reduce tasks belonging to the same job can run in parallel on multiple machines, can be interrupted by any other task, and can be rescheduled to different machines in different time slots.

A scheduler is called **non-preemptive** if each Reduce task can only be scheduled on one machine and, once started, it must keep running without any interruption.

In any time slot t , the number of assigned machines must be less than or equal to the total number of machines N , i.e., $\sum_{i=1}^n (m_{i,t} + r_{i,t}) \leq N, \forall t$.

Let the arrival time of job i be a_i , the time slot in which the last Map task finishes execution be $f_i^{(m)}$, and the time slot in which all Reduce tasks are completed be $f_i^{(r)}$.

For any job i , the workload of its Map tasks should be processed by the assigned number of machines between time slot a_i and $f_i^{(m)}$, i.e., $\sum_{t=a_i}^{f_i^{(m)}} m_{i,t} = M_i, \forall i \in \{1, \dots, n\}$.

For any job i , if $t < a_i$ or $t > f_i^{(m)}$, then $m_{i,t} = 0$. Similarly, for any job i , the workload of its Reduce tasks should be processed by the assigned number of machines between time slot $f_i^{(m)} + 1$ and $f_i^{(r)}$, i.e., $\sum_{t=f_i^{(m)}+1}^{f_i^{(r)}} r_{i,t} = R_i, \forall i \in \{1, \dots, n\}$.

Since any Reduce task of job i cannot start before the finishing time slot of the Map tasks, if $t < f_i^{(m)} + 1$ or $t > f_i^{(r)}$, then $r_{i,t} = 0$.

The waiting and processing time $S_{i,t}$ of job i in time slot t is represented by the indicator function $\mathbf{1}_{\{a_i \leq t \leq f_i^{(r)}\}}$. Thus,

we define the delay cost as $\sum_{t=1}^{\infty} \sum_{i=1}^n S_{i,t}$, which is equal to $\sum_{i=1}^n (f_i^{(r)} - a_i + 1)$. The flow-time F_i of job i is equal to $f_i^{(r)} - a_i + 1$. The objective of the scheduler is to determine

²Because the Map tasks are independent and have small workload [5], such assumption is valid.

the assignment of jobs in each time slot, such that the cost of delaying the jobs or equivalently the flow-time is minimized.

For the preemptive scenario, the problem definition is as follows:

$$\begin{aligned}
\min_{m_{i,t}, r_{i,t}} \quad & \sum_{i=1}^n \left(f_i^{(r)} - a_i + 1 \right) \\
\text{s.t.} \quad & \sum_{i=1}^n (m_{i,t} + r_{i,t}) \leq N, \quad r_{i,t} \geq 0, \quad m_{i,t} \geq 0, \quad \forall t, \\
& \sum_{t=a_i}^{f_i^{(m)}} m_{i,t} = M_i, \quad \sum_{t=f_i^{(m)}+1}^{f_i^{(r)}} r_{i,t} = R_i, \quad \forall i \in \{1, \dots, n\}.
\end{aligned} \tag{1}$$

In the non-preemptive scenario, the Reduce tasks cannot be interrupted by other jobs. Once a Reduce task begins execution on a machine, it has to keep running on that machine without interruption until all its workload is finished. Also, the optimization problem in this scenario is similar to Eq. (1), with additional constraints representing the non-preemptive nature, as shown below:

$$\begin{aligned}
\min_{m_{i,t}, r_{i,t}^{(k)}} \quad & \sum_{i=1}^n \left(f_i^{(r)} - a_i + 1 \right) \\
\text{s.t.} \quad & \sum_{i=1}^n (m_{i,t} + \sum_{k=1}^{K_i} r_{i,t}^{(k)}) \leq N, \quad \forall t, \\
& \sum_{t=a_i}^{f_i^{(m)}} m_{i,t} = M_i, \quad m_{i,t} \geq 0, \quad \forall i \in \{1, \dots, n\}, \\
& \sum_{t=f_i^{(m)}+1}^{f_i^{(r)}} r_{i,t}^{(k)} = R_i^{(k)}, \quad \forall i \in \{1, \dots, n\}, \forall k \in \{1, \dots, K_i\}, \\
& r_{i,t}^{(k)} = 0 \text{ or } 1, \quad r_{i,t}^{(k)} = 1 \text{ if } 0 < \sum_{s=0}^{t-1} r_{i,s}^{(k)} < R_i^{(k)}.
\end{aligned} \tag{2}$$

Theorem 1. *The scheduling problem (both preemptive and non-preemptive) is NP-complete in the strong sense.*

Proof: The proof of NP-completeness follows a fairly standard reduction from 3-Partition and is described in technical report [8]. ■

IV. EFFICIENCY RATIO

The competitive ratio is often used as a measure of performance in a wide variety of scheduling problems. For our problem, the scheduling algorithm S has a competitive ratio of c , if for any total time T , any number of arrivals n in the T time slots, any arrival time a_i of each job i , any workload M_i and R_i of Map and Reduce tasks with respect to each arrival job i , the total flow-time $F^S(T, n, \{a_i, M_i, R_i; i = 1 \dots n\})$ of scheduling algorithm S satisfies the following:

$$\frac{F^S(T, n, \{a_i, M_i, R_i; i = 1 \dots n\})}{F^*(T, n, \{a_i, M_i, R_i; i = 1 \dots n\})} \leq c, \tag{3}$$

where

$$\begin{aligned}
& F^*(T, n, \{a_i, M_i, R_i; i = 1 \dots n\}) \\
& = \min_S F^S(T, n, \{a_i, M_i, R_i; i = 1 \dots n\}),
\end{aligned} \tag{4}$$

is the minimum flow time of an optimal off-line algorithm.

For our problem, we construct an arrival pattern such that no scheduler can achieve a constant competitive ratio c in the following example.

Consider the simplest example of non-preemptive scenario in which the data center only has one machine, i.e., $N = 1$. There is a job with 1 Map task and 1 Reduce task. The workload of Map and Reduce tasks are 1 and $2Q + 1$, respectively. Without loss of generality, we assume that this job arrives in time slot 1. We can show that, for any given constant c_0 , and any scheduling algorithm, there exists a special sequence of future arrivals and workload, such that the competitive ratio c is greater than c_0 .

For any scheduler S , we assume that the Map task of the job is scheduled in time slot $H + 1$, and the Reduce task is scheduled in the $(H + L + 1)^{st}$ time slot ($H, L \geq 0$). Then the Reduce task will last for $2Q + 1$ time slots because the Reduce task cannot be interrupted. This scheduler's operation is given in Fig. 1(a). Observe that any arbitrary scheduler's operation can be represented by choosing appropriate values for H and L possibilities: $H + L > 2(c_0 - 1)(Q + 1) + 1$ or $H + L \leq 2(c_0 - 1)(Q + 1) + 1$.

Case I: $H + L > 2(c_0 - 1)(Q + 1) + 1$.

Consider the arrival pattern in which only this unique job arrives in the system. Under this arrival pattern, the flow-time of S is $L + 2Q + 1$. Now, consider another scheduler S_1 , which schedules the Map task in the first time slot, and schedules the Reduce task in the second time slot. The operation of the scheduler S_1 is depicted in Fig. 1(b), and the flow-time of S_1 is $2Q + 2$. So, the competitive ratio c of S must satisfy the following:

$$c \geq \frac{H + L + 2Q + 1}{2Q + 2} > c_0. \tag{5}$$

Case II: $H + L \leq 2(c_0 - 1)(Q + 1) + 1$.

Let us consider an arrival pattern in which Q additional jobs arrive in time slots $H + L + 2, H + L + 4, \dots, H + L + 2Q$. The Map and Reduce workload of each arrival is 1. Then for the scheduler S , the total flow-time is greater than $(H + L + 2Q + 1) + (2Q + 1)Q$, no matter how the last Q jobs are scheduled. The scheduler S is shown in Fig. 1(c).

Now, we construct a scheduler S_2 , which schedules the Map function of the first arrival in the same time slot as S , and schedules the last Q arrivals before scheduling the Reduce task of the first arrival. The scheduler S_2 is shown in Fig. 1(d).

Then, for the scheduler S , the competitive ratio c satisfies the following:

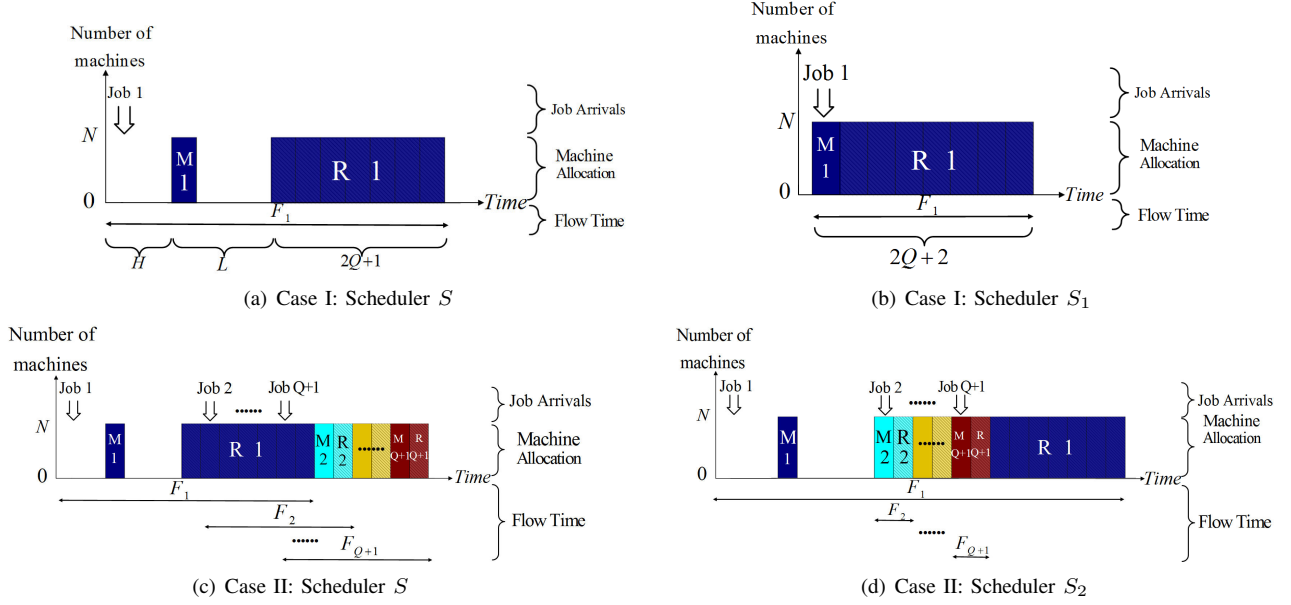


Fig. 1. The Quick Example

$$\begin{aligned}
c &\geq \frac{(H + L + 2Q + 1) + (2Q + 1)Q}{2Q + (H + L + 2Q + 2Q + 1)} \\
&\geq \frac{(2Q + 1)(Q + 1)}{6Q + 1 + (2(c_0 - 1)(Q + 1) + 1)} \\
&> \frac{(2Q + 1)(Q + 1)}{2(c_0 + 2)(Q + 1)} > \frac{Q}{c_0 + 2}.
\end{aligned} \tag{6}$$

By selecting $Q > c_0^2 + 2c_0$, using Eq. 6, we can get $c > c_0$.

Thus, we show that for any constant c_0 and scheduler S , there are sequences of arrivals and workloads, such that the competitive ratio c is greater than c_0 . In other words, in this scenario, there does not exist a constant competitive ratio c . This is because the scheduler does not know the information of future arrivals, i.e., it only makes causal decisions. In fact, even if the scheduler only knows a limited amount of future information, it can still be shown that no constant competitive ratio will hold by increasing the value of Q .

We now introduce a slightly weaker notion of performance, called the **efficiency ratio**.

Definition 2. We say that the scheduling algorithm S has an efficiency ratio γ , if the total flow-time $F^S(T, n, \{a_i, M_i, R_i; i = 1 \dots n\})$ of scheduling algorithm S satisfies the following:

$$\lim_{T \rightarrow \infty} \frac{F^S(T, n, \{a_i, M_i, R_i; i = 1 \dots n\})}{F^*(T, n, \{a_i, M_i, R_i; i = 1 \dots n\})} \leq \gamma, \text{ with probability } 1. \tag{7}$$

Later, we will show that for the quick example, a constant efficiency ratio γ still can exist (e.g., the non-preemptive scenario with light-tailed distributed Reduce workload in Section V).

V. WORK-CONSERVING SCHEDULERS

In this section, we analyze the performance of work-conserving schedulers in both preemptive and non-preemptive

scenarios.

A. Preemptive Scenario

We first study the case in which the workload of Reduce tasks of each job is bounded by a constant, i.e., there exists a constant R_{max} , s.t., $R_i \leq R_{max}$, $\forall i$.

Theorem 2. In the preemptive scenario, any work-conserving scheduler has a constant efficiency ratio $\frac{B_2 + B_1^2}{\max\{2, \frac{1-p_0}{\lambda}, \frac{p_0}{\lambda}\} \max\{1, \frac{1}{N(1-p_0)}\}}$, where p_0 is the probability that no job arrives in a time slot, and B_1 and B_2 are given in Eqs. (8) and (9).

$$B_1 = \min_{\epsilon \in (0, N - \lambda(\overline{M} + \overline{R}))} \left\{ 1 + \left\lceil \frac{(N-1)R_{max}}{\epsilon} \right\rceil + \frac{2e^{l(N-\epsilon)} - 1}{e^{l(N-\epsilon)}(e^{l(N-\epsilon)} - 1)^2} \right\}, \tag{8}$$

$$B_2 = \min_{\epsilon \in (0, N - \lambda(\overline{M} + \overline{R}))} \left\{ \left(1 + \left\lceil \frac{(N-1)R_{max}}{\epsilon} \right\rceil \right)^2 + \frac{4e^{2l(N-\epsilon)} - 3e^{l(N-\epsilon)} + 1}{e^{l(N-\epsilon)}(e^{l(N-\epsilon)} - 1)^3} \right\}, \tag{9}$$

where the rate function $l(a)$ is defined as

$$l(a) = \sup_{\theta \geq 0} (\theta a - \log(E[e^{\theta W_s}])). \tag{10}$$

Proof: (Proof Sketch) We briefly outline the basic idea of the proof. The full details are given in technical report [8].

Consider the total scheduled number of machines over all the time slots. If all the N machines are scheduled in a time slot, we call this time slot a “developed” time slot; otherwise, we call this time slot a “developing” time slot. We define the j^{th} “interval” to be the interval between the $(j-1)^{st}$ developing time slot and the j^{th} developing time slot. (We

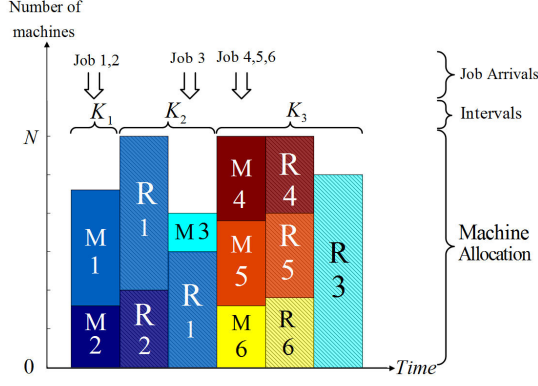


Fig. 2. An example schedule of a work-conserving scheduler

define the first interval as the interval from the first time slot to the first developing time slot.) Thus, the last time slot of each interval is the only developing time slot in the interval. Let K_j be the length of the j^{th} interval, as shown in Fig. 2.

Observe that in each interval, all the job arrivals in this interval must finish their Map tasks in this interval, and all their Reduce tasks will be finished before the end of the next interval. In other words, for all the arrivals in the j^{th} interval, the Map tasks are finished in K_j time slots, and the Reduce tasks are finished in at most $K_j + K_{j+1}$ time slots. For example, job 3 arriving in interval K_2 finishes the Map task in K_2 but the Reduce task is finished in K_3 .

If the scheduler is work-conserving, then for any given number H , there exists a constant B_H , such that $E[K_j^H] < B_H$, $\forall j$, where B_H is given by:

$$B_H \triangleq \min_{\epsilon \in (0, N - \lambda(\bar{M} + \bar{R}))} \left\{ \left(1 + \left\lceil \frac{(N-1)R_{max}}{\epsilon} \right\rceil \right)^H + \sum_{k=2}^{\infty} k^H e^{-kl(N-\epsilon)} \right\}. \quad (11)$$

Thus, $E[K_j]$ is bounded by a constant B_1 , $E[K_j^2]$ is bounded by a constant B_2 , for any j . By adding some dummy Reduce workload in the beginning slot of each interval, the constant efficiency ratio can be achieved by using Strong Law of Large Number (SLLN). ■

B. Non-preemptive Scenario

Theorem 3. *In the non-preemptive scenario, any work-conserving scheduler has a constant efficiency ratio $\frac{B_2 + B_1^2 + B_1(\bar{R}-1)}{\max\{2, \frac{1-p_0}{\lambda}, \frac{\bar{p}}{\lambda}\} \max\{1, \frac{1}{N(1-p)}\}}$, where p_0 is the probability that no job arrives in a time slot, and B_1 and B_2 are given in Eqs. (8) and (9).*

Proof: The outline of the proof is similar to the proof of Theorem 2. The key difference in this scenario is that for job i , which arrives in the j^{th} interval, its Map tasks are finished in K_j time slots, and its Reduce tasks are finished in $K_j + K_{j+1} + R_i - 1$ time slots, as shown in Fig. 3. More details are given in technical report [8]. ■

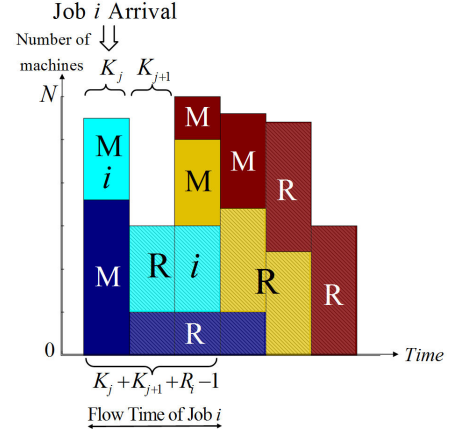


Fig. 3. The job i , which arrives in the j^{th} interval, finishes its Map tasks in K_j time slots and Reduce tasks in $K_j + K_{j+1} + R_i - 1$ time slots.

Remark 1. *In Theorems 2 and 3, we can relax the assumption of boundedness for each Reduce job and allow them to follow a light-tailed distribution, i.e., a distribution on R_i such that $\exists r_0$, such that $P(R_i \geq r) \leq \alpha \exp(-\beta r)$, $\forall r \geq r_0$, $\forall i$, where $\alpha, \beta > 0$ are two constants. We obtain similar results to Theorem 2 and 3 with different expressions. More details are given in technical report [8].*

Remark 2. *Although any work-conserving scheduler has a constant efficiency ratio, the constant efficiency ratio may be large (because the result is true for “any” work-conserving scheduler). We further discuss algorithms to tighten the constant efficiency ratio in Section VI.*

VI.

AVAILABLE-SHORTEST-REMAINING-PROCESSING-TIME (ASRPT) ALGORITHM AND ANALYSIS

In the previous sections we have shown that any arbitrary work-conserving algorithm has a constant efficiency ratio, but the constant can be large as it is related to the size of the jobs. In this section, we design an algorithm with much tighter bounds that does not depend on the size of jobs. Although, the tight bound is provided in the case of preemptive jobs, we also show via numerical results that our algorithm works well in the non-preemptive case.

Before presenting our solution, we first describe a known algorithm called SRPT (Shortest Remaining Processing Time) [10]. SRPT assumes that Map and Reduce tasks from the same job can be scheduled simultaneously in the same slot. In each slot, SRPT picks up the job with the minimum total remaining workload, i.e., including Map and Reduce tasks, to schedule. Observe that the SRPT scheduler may come up with an infeasible solution as it ignores the dependency between Map and Reduce tasks.

Lemma 1. *Without considering the requirement that Reduce tasks can be processed only if the corresponding Map tasks are finished, the total flow-time F_S of Shortest-Remaining-Processing-Time (SRPT) algorithm is a lower bound on the*

total flow-time of MapReduce framework.

Proof: Without the requirement that Reduce tasks can be processed only if the corresponding Map tasks are finished, the optimization problem in the preemptive scenario will be as follows:

$$\begin{aligned} \min_{m_{i,t}, r_{i,t}} \quad & \sum_{i=1}^n (f_i^{(r)} - a_i + 1) \\ \text{s.t.} \quad & \sum_{i=1}^n (m_{i,t} + r_{i,t}) \leq N, \quad r_{i,t} \geq 0, \quad m_{i,t} \geq 0, \quad \forall t, \quad (12) \\ & \sum_{t=a_i}^{f_i^{(r)}} (m_{i,t} + r_{i,t}) = M_i + R_i, \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$

The readers can easily check that

$$\begin{aligned} \bigcup_{i=1}^n \left\{ \sum_{t=a_i}^{f_i^{(r)}} m_{i,t} + r_{i,t} = M_i + R_i \right\} \supset \\ \bigcup_{i=1}^n \left\{ \left\{ \sum_{t=a_i}^{f_i^{(m)}} m_{i,t} = M_i \right\} \cap \left\{ \sum_{t=f_i^{(m)}+1}^{f_i^{(r)}} r_{i,t} = R_i \right\} \right\}. \end{aligned} \quad (13)$$

Thus, the constraints in Eq. (12) are weaker than constraints in Eq. (1). Hence, the optimal solution of Eq. (12) is less than the optimal solution of Eq. (1). Since we know that the SRPT algorithm can achieve the optimal solution of Eq. (12) [10], then its total flow-time F_S is a lower bound of any scheduling method in the MapReduce framework. Since the optimization problem in the non-preemptive scenario has more constraints, the lower bound also holds for the non-preemptive scenario. ■

A. ASRPT Algorithm

Based on the SRPT scheduler, we present our greedy scheme called ASRPT. We base our design on a very simple intuition that by ensuring that the Map tasks of ASRPT finish as early as SRPT, and assigning Reduce tasks with smallest amount of remaining workload, we can hope to reduce the overall flow-time. However, care must be taken to ensure that if Map tasks are scheduled in this slot, then the Reduce tasks are scheduled after this slot.

ASRPT works as follows. ASRPT uses the schedule computed by SRPT to determine its schedule. In other words, it runs SRPT in a virtual fashion and keeps track of how SRPT would have scheduled jobs in any given slot. In the beginning of each slot, the list of unfinished jobs J , and the scheduling list S of the SRPT scheduler are updated. The scheduled jobs in the previous time slot are updated and the new arriving jobs are added to the list J in non-decreasing order of available workload in this slot. In the list J , we also keep the number of schedulable tasks in this time-slot. For a job that has unscheduled Map tasks, its available workload in this slot is the units of unfinished workload of both Map and Reduce tasks, while its schedulable tasks are

the unfinished Map tasks. Otherwise, its available workload is the unfinished Reduce workload, while its schedulable tasks are the unfinished workload of the Reduce tasks (preemptive scenario) or the number of unfinished Reduce tasks (non-preemptive scenario), respectively. Then, the algorithm assigns machines to the tasks in the following priority order (from high to low): the previously scheduled Reduce tasks which are not finished yet (only in the non-preemptive scenario), the Map tasks which are scheduled in the list S , the available Reduce tasks, and the available Map tasks. For each priority group, the algorithm greedily assign machines to the corresponding tasks through the sorted available workload list J . The pseudo-code of the algorithm is shown in Algorithm 1.

B. Efficiency Ratio Analysis of ASRPT Algorithm

We first prove a lower bound on its performance. The lower bound is based on the SRPT. For the SRPT algorithm, we assume that in time slot t , the number of machines which are scheduled to Map and Reduce tasks are M_t^S and R_t^S , respectively.

Now we construct a scheduling method called Delayed-Shortest-Remaining-Processing-Time (DSRPT) for MapReduce framework based on the SRPT algorithm. We keep the scheduling method for the Map tasks exactly same as in SRPT. For the Reduce tasks, they are scheduled in the same order from the *next time slot* compared to SRPT scheduling. An example showing the operation of SRPT, DSRPT and ASRPT is shown in Fig. 4.

Theorem 4. *In the preemptive scenario, DSRPT has an efficiency ratio 2.*

Proof: We construct a queueing system to represent the DSRPT algorithm. In each time slot $t \geq 2$, there is an arrival with workload R_{t-1}^S , which is the total workload of the delayed Reduce tasks in time slot $t-1$. The service capacity of the Reduce tasks is $N - M_t^S$, and the service policy of the Reduce tasks is First-Come-First-Served (FCFS). The Reduce tasks which are scheduled in previous time slots by the SRPT algorithm are delayed in the DSRPT algorithm up to the time of finishing the remaining workload of delayed tasks. Also, the remaining workload W_t in the DSRPT algorithm will be processed first, because the scheduling policy is FCFS. Let D_t be the largest delay time of the Reduce tasks which are scheduled in the time slot $t-1$ in SRPT.

In the construction of the DSRPT algorithm, the remaining workload W_t is from the delayed tasks which are scheduled in previous time slots by the SRPT algorithm, and $M_{s \geq t}^S$ is the workload of Map tasks which are scheduled in the future time slots by the SRPT algorithm. Hence, they are independent. We assume that D'_t is the first time such that $W_t \leq N - M_s$, where $s \geq t$. Then $D_t \leq D'_t$.

If a time slot $t_0 - 1$ has no workload (except the R_{t_0-1} which is delayed to the next time slot) left to the future, we call t_0 the first time slot of the interval it belongs to. (Note that the definition of interval is different from Section V.) Assume that $P(W_{t_0} \leq N - M_s) = P(R_{t_0-1} \leq N - M_s) = p$, where

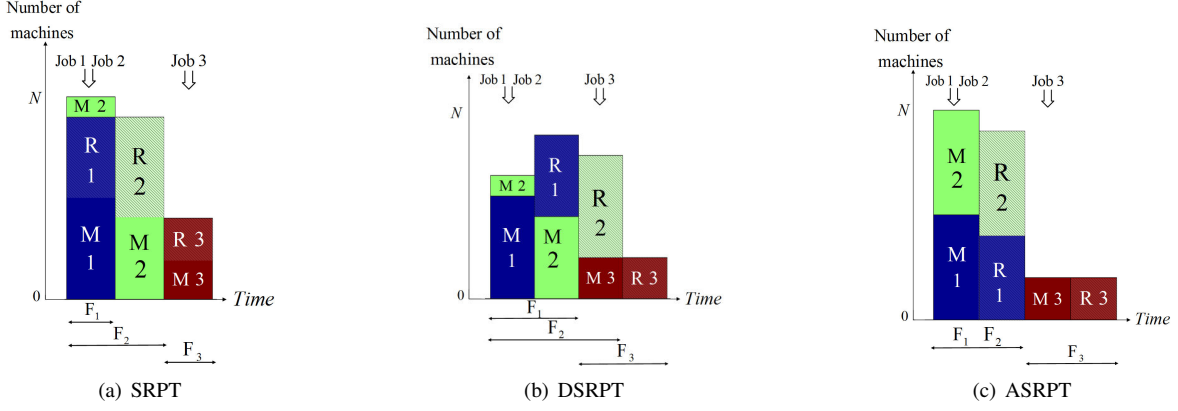


Fig. 4. The construction of schedule for ASRPT and DSRPT based on the schedule for SRPT. Observe that the Map tasks are scheduled in the same way. But, roughly speaking, the Reduce tasks are delayed by one slot.

$s \geq t_0$. Since $R_s \leq N - M_s$, then $p \geq P(R_{t_0-1} \leq R_s) \geq 1/2$. Thus, we can get that

$$E[D_{t_0}] \leq E[D'_{t_0}] = \sum_{k=1}^{\infty} kp(1-p)^{k-1} = \frac{1}{p} \leq 2. \quad (14)$$

Note that $N - M_s \geq R_s$ for all s , then the current remaining workload $W_t \leq W_{t_0}$. Then, $E[D_t] \leq E[D_{t_0}] \leq 2$.

Then, for all the Reduce tasks, the expectation of delay compared to the SRPT algorithm is not greater than 2. Let \mathbb{D} has the same distribution with D_t . Thus, the total flow-time F_D of DSRPT algorithm satisfies

$$\begin{aligned} \lim_{T \rightarrow \infty} \frac{F_D}{F_S} &\leq \frac{\lim_{n \rightarrow \infty} \frac{F_S}{n} + E[\mathbb{D}]}{\lim_{n \rightarrow \infty} \frac{F_S}{n}} \text{ w.p.1} \\ &= 1 + \frac{E[\mathbb{D}]}{\lim_{n \rightarrow \infty} \frac{F_S}{n}} \leq 1 + E[\mathbb{D}] \leq 3. \end{aligned} \quad (15)$$

For the flow time F of any feasible scheduler in the MapReduce framework, we have

$$\begin{aligned} \lim_{T \rightarrow \infty} \frac{F_D}{F} &\leq \frac{\lim_{n \rightarrow \infty} \frac{F_S}{n} + E[\mathbb{D}]}{\lim_{n \rightarrow \infty} \frac{F}{n}} \text{ w.p.1} \\ &\leq 1 + \frac{E[\mathbb{D}]}{\lim_{n \rightarrow \infty} \frac{F}{n}} \leq 1 + \frac{E[\mathbb{D}]}{2} \leq 2. \end{aligned} \quad (16)$$

Corollary 1. *From the proof of Theorem 4, the total flow-time of DSRPT is not greater than 3 times the lower bound given by Lemma 1 with probability 1, when n goes to infinity.*

Corollary 2. *In the preemptive scenario, the ASRPT scheduler has an efficiency ratio 2.*

Proof: For each time slot, all the Map tasks finished by DSRPT are also finished by ASRPT. For the Reduce tasks of each job, the jobs with smaller remaining workload will be finished earlier than the jobs with larger remaining workload. Hence, based on the optimality of SRPT, ASRPT can be

viewed as an improvement of DSRPT. Thus, the total flow-time of ASRPT will not be greater than DSRPT. So, the efficiency ratio of DSRPT also holds for ASRPT. ■

Note that, the total flow-time of ASRPT is not greater than 3 times of the lower bound given by Lemma 1 with probability 1, when n goes to infinity. We will show this performance in the next section. Also, the performance of ASRPT is analyzed in the preemptive scenario in Corollary 2. We will show that ASRPT performs better than other schedulers in both preemptive and non-preemptive scenarios via simulations in the next section.

VII. SIMULATION RESULTS

A. Simulation Setting

We evaluate the efficacy of our algorithm ASRPT for both preemptive and non-preemptive scenarios. We consider a data center with $N = 100$ machines, and choose Poisson process with arrival rate $\lambda = 2$ jobs per time slot as the job arrival process. We choose uniform distribution and exponential distribution as examples of bounded workload and light-tailed distributed workload, respectively. For short, we use $Exp(\mu)$ to represent an exponential distribution with mean μ , and use $U[a, b]$ to represent a uniform distribution on $\{a, a+1, \dots, b-1, b\}$. We choose the total time slots to be $T = 500$, and the number of tasks in each job is up to 10.

We compare 3 typical schedulers to the ASRPT scheduler:

The **FIFO** scheduler: It is the default scheduler in Hadoop. All the jobs are scheduled in their order of arrival.

The **Fair** scheduler: It is a widely used scheduler in Hadoop. The assignment of machines are scheduled to all the waiting jobs in a fair manner. However, if some jobs need fewer machines than others in each time slot, then the remaining machines are scheduled to the other jobs, to avoid resource wastage and to keep the scheduler work-conserving.

The **LRPT** scheduler: Jobs with larger unfinished workload are always scheduled first. Roughly speaking, the performance of this scheduler represents in a sense how poorly even some work-conserving schedulers can perform.

Algorithm 1 Available-Shortest-Remaining-Processing-Time (ASRPT) Algorithm for MapReduce Framework

Input: List of unfinished jobs (including new arrivals in this slot) J
Output: Scheduled machines for Map tasks, scheduled machines for Reduce tasks, updated remaining jobs J

```

1: Update the scheduling list  $S$  of SRPT algorithm without task
   dependency. For job  $i$ ,  $S(i).MapLoad$  is the corresponding
   scheduling of the Map tasks in SRPT.
2: Update the list of jobs with new arrivals and machine assignments
   in the previous slot to maintain a non-decreasing order of the
   available total workload  $J(i).AvailableWorkload$ . Also, keep
   the number of schedulable tasks  $J(i).SchedulableTasksNum$ 
   for each job in the list.
3:  $d \leftarrow N$ ;
4: if The scheduler is a non-preemptive scheduler then
5:   Keep the assignment of machines which are already scheduled
   to the Reduce tasks previously and not finished yet;
6:   Update  $d$ , the idle number of machines;
7: end if
8: for  $i = 1 \rightarrow |J|$  do
9:   if  $i$  has Map tasks which is scheduled in  $S$  then
10:    if  $J(i).SchedulableTasksNum \geq S(i).MapLoad$  then
11:      if  $S(i).MapLoad \leq d$  then
12:        Assign  $S(i).MapLoad$  machines to the Map tasks of
        job  $i$ ;
13:      else
14:        Assign  $d$  machines to the Map tasks of job  $i$ ;
15:      end if
16:    else
17:      if  $J(i).SchedulableTasksNum \leq d$  then
18:        Assign  $J(i).SchedulableTasksNum$  machines to the
        Map tasks of job  $i$ ;
19:      else
20:        Assign  $d$  machines to the Map tasks of job  $i$ ;
21:      end if
22:    end if
23:    Update status of job  $i$  in the list of jobs  $J$ ;
24:    Update  $d$ , the idle number of machines;
25:  end if
26: end for
27: for  $i = 1 \rightarrow |J|$  do
28:   if  $i^{th}$  job still has unfinished Reduce tasks then
29:     if  $J(i).SchedulableTasksNum \leq d$  then
30:       Assign  $J(i).SchedulableTasksNum$  machines to the
       Reduce tasks of job  $i$ ;
31:     else
32:       Assign  $d$  machines to the Reduce tasks of job  $i$ ;
33:     end if
34:     Update  $d$ , the idle number of machines;
35:   end if
36:   if  $d = 0$  then
37:     RETURN;
38:   end if
39: end for
40: for  $i = 1 \rightarrow |J|$  do
41:   if  $i^{th}$  job still has unfinished Map tasks then
42:     if  $J(i).SchedulableTasksNum \leq d$  then
43:       Assign  $J(i).SchedulableTasksNum$  machines to the
       Map tasks of job  $i$ ;
44:     else
45:       Assign  $d$  machines to the Map tasks of job  $i$ ;
46:     end if
47:     Update  $d$ , the idle number of machines;
48:   end if
49:   if  $d = 0$  then
50:     RETURN;
51:   end if
52: end for

```

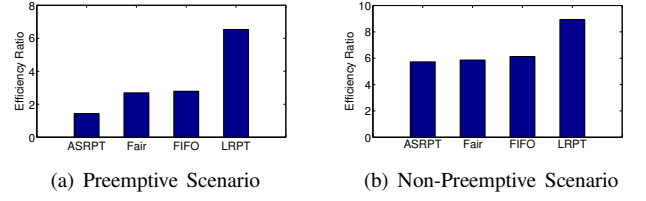


Fig. 5. Efficiency Ratio (Exponential Distribution, Large Reduce)

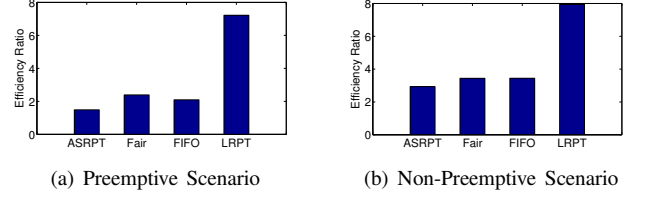


Fig. 6. Efficiency Ratio (Exponential Distribution, Small Reduce)

B. Efficiency Ratio

In the simulations, the efficiency ratio of a scheduler is obtained by the total flow-time of the scheduler over the lower bound of the total flow-time in T time slots. Thus, the real efficiency ratio should be smaller than the efficiency ratio given in the simulations. However, the proportion of all schedulers would remain the same.

First, we evaluate the exponentially distributed workload. We choose the workload distribution of Map for each job as $Exp(5)$ and the workload distribution of Reduce for each job as $Exp(40)$. The efficiency ratios of different schedulers are shown in Fig. 5. For different workload, we choose workload distribution of Map as $Exp(30)$ and the workload distribution of Reduce as $Exp(15)$. The efficiency ratios of schedulers are shown in Fig. 6.

Then, we evaluate the uniformly distributed workload. We choose the workload distribution of Map for each job as $U[1, 9]$ and the workload distribution of Reduce for each job as $U[10, 70]$. The efficiency ratios of different schedulers are shown in Fig. 7. To evaluate for a smaller Reduce workload, we choose workload distribution of Map as $U[10, 50]$ and the workload distribution of Reduce as $U[10, 20]$. The efficiency ratios of different schedulers are shown in Fig. 8.

As an example, we show the convergence of efficiency ratios in Fig. 9, where the workload distribution of Map as $U[10, 50]$ and the workload distribution of Reduce as $U[10, 20]$. More simulations of convergence are shown in technical report [8].

From Figures. 5-8, we can see that the total flow-time of

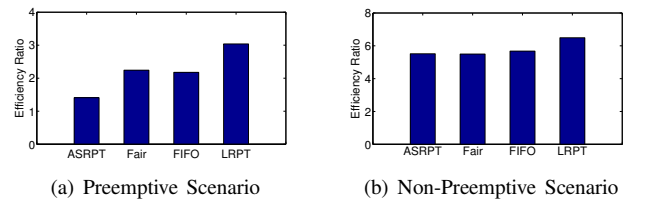


Fig. 7. Efficiency Ratio (Uniform Distribution, Large Reduce)

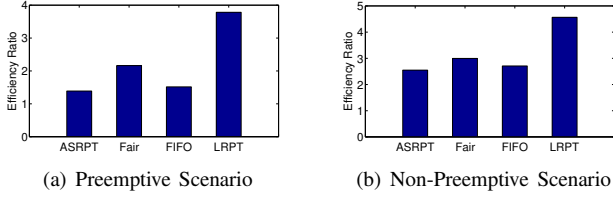


Fig. 8. Efficiency Ratio (Uniform Distribution, Small Reduce)

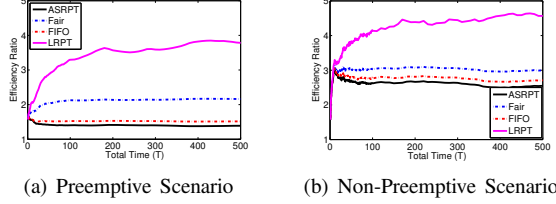


Fig. 9. Convergence of Efficiency Ratios (Uniform Distribution, Small Reduce)

ASRPT is much smaller than all the other schedulers. Also, as a “bad” work-conserving, the LRPT scheduler also has a constant (maybe relative large) efficiency ratio, from Fig. 9.

C. Cumulative Distribution Function (CDF)

For the same setting and parameters, the CDFs of flow-times are shown in Fig. 10-13. We plot the CDF only for flow-time up to 100 units. From these figures, we can see that the ASRPT scheduler has a very light tail in the CDF of flow-time, compared to the FIFO and Fair schedulers. In other words, the fairness of the ASRPT scheduler is similar to the FIFO and Fair schedulers. However, the LRPT scheduler has a long tail in the CDF of flow time. In the other words, the fairness of the LRPT scheduler is not as good as the other schedulers.

VIII. CONCLUSION

In this paper, we study the problem of minimizing the total flow-time of a sequence of jobs in the MapReduce framework, where the jobs arrive over time and need to be

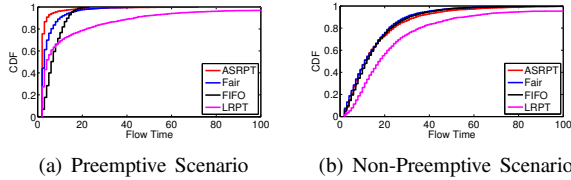


Fig. 10. CDF of Schedulers (Exponential Distribution, Large Reduce)

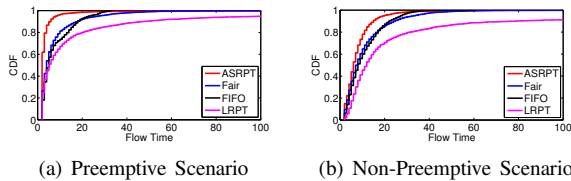


Fig. 11. CDF of Schedulers (Exponential Distribution, Small Reduce)

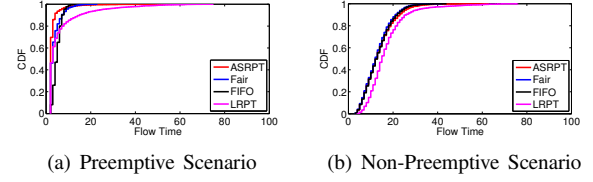


Fig. 12. CDF of Schedulers (Uniform Distribution, Large Reduce)

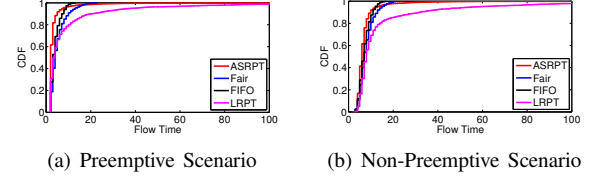


Fig. 13. CDF of Schedulers (Uniform Distribution, Small Reduce)

processed through Map and Reduce procedures before leaving the system. We show that no on-line algorithm can achieve a constant competitive ratio for non-preemptive tasks. We define weaker metric of performance called the efficiency ratio and propose a corresponding technique to analyze on-line schedulers. Under some weak assumptions, we then show a surprising property that for the flow-time problem any work-conserving scheduler has a constant efficiency ratio in both preemptive and non-preemptive scenarios. More importantly, we are able to find an online scheduler ASRPT with a very small efficiency ratio. The simulation results show that the efficiency ratio of ASRPT is much smaller than the other schedulers, while the fairness of ASRPT is as good as others.

REFERENCES

- [1] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” in *Proceedings of Sixth Symposium on Operating System Design and Implementation, OSDI*, pp. 137–150, December 2004.
- [2] F. Chen, M. Kodialam, and T. Lakshman, “Joint scheduling of processing and shuffle phases in mapreduce systems,” in *Proceedings of IEEE Infocom*, March 2012.
- [3] H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee, “Scheduling in mapreduce-like systems for fast completion time,” in *Proceedings of IEEE Infocom*, pp. 3074–3082, March 2011.
- [4] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Job scheduling for multi-user mapreduce clusters,” tech. rep., University of California, Berkeley, April 2009.
- [5] J. Tan, X. Meng, and L. Zhang, “Performance analysis of coupling scheduler for mapreduce/hadoop,” in *Proceedings of IEEE Infocom*, March 2012.
- [6] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlos, “On scheduling in map-reduce and flow-shops,” in *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures, SPAA*, pp. 289–298, June 2011.
- [7] “Hadoop, <http://hadoop.apache.org/>.”
- [8] Y. Zheng, P. Sinha, and N. Shroff, “Performance Analysis of Work-Conserving Scheduler in Minimizing Flowtime Problem with Two-Stages Precedence,” tech. rep., Ohio State University, June 2012. <http://www2.ece.ohio-state.edu/~zhengy/TR3.pdf>.
- [9] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling,” in *Proceedings of the 5th European conference on Computer systems, EuroSys*, pp. 265–278, April 2010.
- [10] K. R. Baker and D. Trietsch, *Principles of Sequencing and Scheduling*. Hoboken, NJ, USA: John Wiley & Sons, 2009.