On Scheduling in Map-Reduce and Flow-Shops

Benjamin Moseley[*] Anirban Dasgupta Ravi Kumar Tamás Sarlós
University of Illinois
Urbana, IL
bmosele2@illinois.edu

Yahoo! Research
Sunnyvale, CA
{anirban,ravikumar,stamas}@yahoo-inc.com

ABSTRACT

The map-reduce paradigm is now standard in industry and academia for processing large-scale data. In this work, we formalize job scheduling in map-reduce as a novel generalization of the two-stage classical *flexible* flow shop (`FFS`) problem: instead of a single task at each stage, a job now consists of a set of tasks per stage. For this generalization, we consider the problem of minimizing the total *flowtime* and give an efficient 12-approximation in the offline setting and an online $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm.

Motivated by map-reduce, we revisit the two-stage flow shop problem, where we give a dynamic program for minimizing the total *flowtime* when all jobs arrive at the same time. If there are fixed number of job-types the dynamic program yields a PTAS; it is also a QPTAS when the processing times of jobs are polynomially bounded. This gives the first improvement in approximation of flowtime for the two-stage flow shop problem since the trivial 2-approximation algorithm of Gonzalez and Sahni [29] in 1978, and the first known approximation for the `FFS` problem. We then consider the generalization of the two-stage `FFS` problem to the unrelated machines case, where we give an offline 6-approximation and an online $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^4})$-competitive algorithm.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Theory

Keywords

Scheduling and resource allocation, Algorithm analysis, Approximation algorithms, On-line problems, Map-reduce, Flow-shops

1. INTRODUCTION

Map-reduce [9] has already established itself as the computing paradigm of choice to process massive data. The underlying idea in map-reduce is elegant. The input data is viewed as a stream of records comprising of key-value pairs. A map-reduce computation consists of a map phase, consisting of map tasks, followed by a reduce phase, consisting of reduce tasks. Each map task runs on a map machine and processes a portion of the input, outputting (possibly new) key-value pairs en route; the map tasks can be run in parallel. In the reduce phase, the key-value pairs output by the map machines are processed in parallel by reduce tasks, which run on the reduce machines, under the guarantee that all the records with the same key will be available together in one reduce machine. The reduce phase of a job therefore cannot begin until the map phase ends, i.e., all the map machines complete their work. Google's MapReduce and Apache Hadoop (`hadoop.apache.org`) are two existing implementations of map-reduce; both these implementations have useful enhancements such as job priorities, queues, batch processing, etc. The success of map-reduce as a parallel programming model can be attributed to its simplicity and its ability to hide low-level issues such as scheduling, failures, data locality, network bandwidth, and machine availability, from the end user.

Although map-reduce is a distributed computational model, the task-scheduling decisions are coordinated by a *centralized* job-tracker process that runs in a "master node." Designing new scheduling policies has been one of the active research topics in map-reduce because of the need to balance often contradictory needs, e.g., system utilization, fairness, and response times. There has been a great deal of empirical work demonstrating the value of a well-designed job-scheduling scheme in finding a good trade-off point among these different objectives [19, 34, 28, 35]. On the other hand, there has been very little work from a theoretical point of view. Wolf et al. [33] formalize the problem of allocation of slots among jobs by the Hadoop Fair scheduler (which is the default in many implementations), and present heuristic allocation schemes designed to optimize several scheduling metrics — these heuristics come with theoretical guarantees only in an idealized model that assumes infinitesimal task times, linear relation of processing times and allocated resources, no map-reduce dependencies, and one-shot allocation of resources. Fischer, Su, and Yin [11] show hardness results and present algorithms for the task-assignment problem with costs that reflect data locality. None of these papers presents theoretical guarantees for the underlying job-scheduling problem in terms of the commonly studied metrics in scheduling theory.

In this paper we study a scheduling model that captures the core challenges in map-reduce scheduling. The problem here is to assign jobs consisting of several map and reduce tasks to the available map and reduce machines in the best possible manner. However, it is tricky to adapt existing scheduling techniques to this setting due to the following non-negotiable reasons: (i) there are multi-

[*]Work done while visiting Yahoo! Labs. Partially supported by NSF grants CCF-0728782 and CCF-1016684.

ple map and reduce tasks in a job and multiple machines to which they can be assigned, (ii) map tasks have to be assigned to map machines and reduce tasks have to be assigned to reduce machines[1], (iii) no reduce task can be run before all map tasks from this job finish[2], (iv) the schedule can be preemptive but non-migratory, i.e., a task should be run on a single machine since it is wasteful to ship around partially processed data. It is also preferable to take data locality into account during the assignment of tasks to machines. Furthermore, both online and offline scenarios are relevant since map-reduce implementations typically permit batch as well online processing of jobs. Given that large map-reduce clusters are usually shared among several users, the most natural metric is to minimize the time between the arrival and the completion of a job, i.e., the *flowtime* [27].

Problem formulation. We formulate the problem of map-reduce scheduling by abstracting the above requirements and desiderata in scheduling terms. In particular, we focus on multiple-task multiple-machine two-stage non-migratory scheduling with precedence constraints; these constraints exist between each map task and reduce task for a job. This essentially captures the properties (i)–(iv) outlined above. We allow preemption in all our settings, unless noted, and focus on the objective of minimizing the total (equivalently, average) flowtime.

At a high-level, we consider two job scenarios, namely, the *offline* arrival of jobs and the *online* arrival. In the offline case, jobs arrive together; the algorithmic focus is on optimizing the approximation ratio. In the online case, jobs arrive over time and the scheduler makes decisions without knowing the jobs that are yet to arrive; the algorithmic focus is on optimizing the competitive ratio. Orthogonally, we consider two processing-time configurations. First, in the *identical machines* setting, all map machines have the same speed and all reduce machines have the same speed. Second, in the *unrelated machines* setting, the processing time for each task is a vector, specifying the task's running time on each of the machines; this is aimed at capturing the data locality desideratum. The unrelated machines model can also be used when different map machines have different amounts of memory and each task carries a minimum memory requirement to run. There is a large amount of literature on the unrelated machines model in scheduling theory and this is perhaps the most general machine model (see [14, 15, 4, 18, 32]).

Our results. Our main contribution is to model map-reduce scheduling as a generalization of the two-stage flexible flow-shop problem (FFS)[3]. The map-reduce scheduling problem generalizes FFS by having a *set* of map tasks per job that need to be scheduled on the map machines and a *set* of reduce tasks that are to be scheduled on the reduce machines. Our aim is to design schedules that minimize the total flowtime. Our main results are in the identical machines setting, where we obtain a 12-approximation algorithm

[1]This requirement does not follow immediately from the map-reduce programming model. However map and reduce tasks have typically different resource requirements and dividing a physical machine into multiple virtual map and reduce machines helps balance the load; Hadoop follows this model. Our results also extend to the case when a machine can execute both map and reduce tasks; see Section 3.

[2]We ignore in our model the data-aggregation (*shuffle*) phase of reduce that precedes the running of the user-code in reduce tasks, and can start before maps finish.

[3]In FFS each job consists of two tasks and the first task can be scheduled on a set of identical machines (say, map machines) and the second task can be scheduled on a different set of identical machines (say, reduce machines); the first task must be completed before the second can be started.

	Offline	Online
Map-reduce, Identical machines	NP-hard, 12-approx. (Corollary 3)	$\Omega(\min\{\log P, \log n/N\})$ [25], $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-comp. (Corollary 8)
FFS, Unrelated machines	NP-hard, 6-approx. (Corollary 15)	unbounded [14], $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^4})$-comp. (Corollary 20)

Table 1: Summary of results on minimizing total flow-time. P is the ratio of the largest task size to the smallest, N is the total number of machines, and n is the number of jobs. For the FFS identical machines case, we obtain a QPTAS for polynomial job sizes (Theorem 11) and a PTAS for fixed processing times (Theorem 12).

for the offline case and a $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm for the online case where $0 < \epsilon \leq 1$; the online result assumes resource augmentation [21], which is necessary to circumvent lower bounds. It is important to note that in the offline setting, we consider the case where all jobs arrive simultaneously; otherwise, if jobs arrive over time, a constant approximation cannot be achieved without resource augmentation [14].

Using the ideas developed for the identical machine case, we consider the unrelated machines case. However, it seems difficult to find good schedulers when there are multiple map and reduce tasks per job. In an effort to find such algorithms, we consider a natural generalization of FFS to the unrelated machines case while each job still has one map and one reduce task. We obtain a 6-approximation algorithm for the offline case and a $(1+\epsilon)$-speed $O(1/\epsilon^5)$-competitive algorithm for the online case where $0 < \epsilon \leq 1$; these results can be found in Section 5.

The two-stage flow (not flexible) shop problem F1S is an important special case of FFS, where there is only one map machine and one reduce machine. F1S is known to be strongly NP-hard [13]. We give the *first* non-trivial approximation algorithm for F1S and FFS offline. Specifically, we give a quasi-polynomial time approximation scheme (QPTAS) when the largest job is polynomial-sized. Our algorithm is also a polynomial time approximation scheme (PTAS) in the case that there are a fixed number of processing times for each job. This is the only approximation algorithm known for FFS or F1S problems besides the trivial 2-approximation for F1S shown in [29] over three decades ago.

Related work. The two stage flexible flow shop problem (FFS) has been studied extensively; see [30, 23, 31, 16] for pointers to recent work. In the map-reduce case, on the other hand, we have multiple tasks per job per stage. We feel that beyond being practically important, the map-reduce scheduling problem is also a novel generalization of FFS that has not been previously studied.

Almost all previous work on FFS has focused on the case where all jobs arrive at the same time and the objective is to minimize the *maximum* completion time of any job. For this problem a PTAS was given in [16] for a fixed number of machines per stage and extended in [31] to a variable number of machines. Johnson's well-known algorithm [20] is also optimal for F1S when minimizing the maximum completion time. As mentioned, throughout this paper we focus on minimizing the *total flowtime*. Little is known about this objective for the FFS and F1S problems; there are no approximation algorithms known for FFS, while a trivial 2-approximation was shown for F1S [29]. It was suggested by Schuurman and Woeginger in the survey [31] that improving the 2-approximation for F1S is an important open question.

Most of the algorithmic work on map-reduce that has been done so far falls into one of the following three categories. The first is the development of computational models that faithfully capture the power and limitations of map-reduce, e.g., the work of Feldman et al. [10] and Karloff et al. [24]. The second is the development of map-reduce algorithms for several basic problems [24, 7]; problems such as MST, maximum cover, connectivity were shown to have efficient map-reduce algorithms. The third is the development of practical map-reduce-based heuristics to solve large-scale data problems, especially in text processing, graph analysis, and machine learning; see, for example, [22, 26, 8].

A critique. Since our goal is to provide a simple formalization of the scheduling problem in the map-reduce framework, we have deliberately ignored many issues in real systems that often have a large effect on the performance. We discuss some of these issues here. In the real system, intermediate data is transferred from the map machines to the reduce machines and thus the network bandwidth forms a significant bottleneck. Data locality — running the map tasks in the machines where data is located — is another important issue, as we mentioned earlier. Instead of modeling processing times as function of the network topology, we chose to model the effect of locality by the rather stylized unrelated machine setting. We also do not model task and machine failures, another important topic. In the map-reduce setting the dependence between map and reduce-tasks is more subtle than what we have described here due to the presence of the intermediate shuffle phase, which happens in parallel with the map tasks. We assume preemption, which is not yet a feature in Hadoop; interestingly, it is not hard to show that an online algorithm will have a large competitive ratio if preemption is not allowed without resource augmentation. Finally, we assume that the scheduler is aware of the job sizes. These may not be immediately available in practice, but in nearly all circumstances approximate job sizes can be determined based on historical data [12, 33].

2. PRELIMINARIES

Job and task. A *job* consists of sets of *tasks*, where tasks in a set can be run in parallel, but the sets themselves have to be run sequentially. In the map-reduce setting, we assume that each job has two sets of tasks, namely, a set of *map* tasks and a set of *reduce* tasks, where no reduce task can be started until all map tasks for the job are completed. Thus, the scheduling problem is precedence-constrained.

Let $\mathcal{J}$ be the set of jobs and let $J \in \mathcal{J}$ denote a generic job. Let $\{J_i^m\}$ and $\{J_i^r\}$ be the set of map and reduce tasks of J, respectively. When both these sets are singletons, we call this the *single task* case; otherwise, it is the *multiple task* case. Let the function $p(\cdot)$ be the *processing time* of a job or a task. In the case where the processing time depends on the machine assignment, let $p_x(\cdot)$ be the processing time of a job or a task on machine x. If a machine runs a task J at speed s, then it needs $p(J)/s$ time to complete the task; unless otherwise noted, we assume all machines run at unit speed. We also assume that the processing times of the tasks are known to the algorithms.

To avoid being repetitive, throughout the paper, let $b \in \{m, r\}$; this will be used to capture both map- and reduce-related statements for both machines and tasks, i.e., when b is used, it is fixed to either m or r. Let $J^{b,*} = \arg\max_i p(J_i^b)$ be the task with the maximum processing time in a set of tasks and let $J^* = \arg\max\{p(J^{m,*}), p(J^{r,*})\}$ be the task with the maximum processing time. Let a_J be the arrival time of job J.

Schedule. Let σ be a schedule of jobs. Given σ, for any job or task, let the function $\mathsf{s}_\sigma(\cdot)$ denote its starting time and the function $\mathsf{f}_\sigma(\cdot)$ denote its completion time, both with respect to the schedule σ. We also define $\mathsf{s}_\sigma^b(J) = \min_i s_\sigma(J_i^b)$ and $\mathsf{f}_\sigma^b(J) = \max_i \mathsf{f}(J_i^b)$, the starting and finishing times for a set of tasks; thus, $\mathsf{s}_\sigma(J) = \mathsf{s}_\sigma^m(J)$, $\mathsf{f}_\sigma(J) = \mathsf{f}_\sigma^r(J)$. Let N_m be the number of map machines, i.e., machines on which map tasks can be run and let N_r be the number of reduce machines, i.e., machines on which reduce tasks can be run. A schedule σ is called *viable* if for each job J: (i) all map tasks of J are scheduled only on the map machines, (ii) all reduce tasks of J are scheduled only on the reduce machines, and (iii) every reduce task for job J is scheduled only after all map tasks for job J are completed, i.e., $\mathsf{f}_\sigma^m(J) \le \mathsf{s}_\sigma^r(J)$. A schedule σ is called *non-migratory* if each task is run only on one machine.

The *flowtime* of a job J with respect to a schedule σ is $\texttt{flow}_\sigma(J) = \mathsf{f}_\sigma(J) - a_J$; let $\texttt{flow}_\sigma = \sum_J \texttt{flow}_\sigma(J)$ be the total flowtime. The total *completion time* of a schedule σ is $\sum_J \mathsf{f}_\sigma(J)$. When all jobs arrive at time 0, completion time is the same as flowtime. We will consider two different scheduling metrics: minimizing the total flowtime (equivalently, the average flowtime) and minimizing the total completion time. For a time interval I, let $|I|$ denote its length.

3. MAP-REDUCE: IDENTICAL MACHINES CASE

Consider the map-reduce scheduling problem when all the map machines are identical, all the reduce machines are identical, and each job can have multiple map tasks and multiple reduce tasks. We will construct the map-reduce schedule out of two individual schedules for the map and the reduce tasks. Let σ_m denote some schedule on a single map machine of speed N_m for just the map tasks. Likewise, ignoring the precedence constraints between map and reduce, let σ_r denote some schedule for all the reduce tasks on a single reduce machine of speed N_r.

3.1 Offline scheduling

We first construct a non-migratory schedule σ for the offline setting where all jobs arrive at time 0. Our goal is to reduce the precedence constrained map-reduce scheduling problem to simpler scheduling problems. To do this, we will use σ_m and σ_r to assign priorities to tasks and construct the final schedule σ using these priorities.

THEOREM 1. *Given schedules σ_m and σ_r, there is a viable non-migratory schedule σ such that for all job J it holds that* $\mathsf{f}_\sigma(J) \le 4\max\{\mathsf{f}_{\sigma_m}^m(J), \mathsf{f}_{\sigma_r}^r(J), p(J^*)\}$.

PROOF. We now describe the algorithm to construct σ. Define w_J, the *width* of job J, as the maximum of map and reduce finish times of the job and the maximum task length, i.e., $w_J = \max\{\mathsf{f}_{\sigma_m}^m(J), \mathsf{f}_{\sigma_r}^r(J), p(J^*)\}$. Note that while width incorporates the maximum flowtime that J incurs in σ_r or σ_m, it also incorporates the processing time of the largest task of job J; this will later be used to ensure that a unit speed scheduler can finish the largest task of J in time w_J. The width of a job is its priority and a smaller width means higher priority.

We first show a generic bound on the finish time of the task in terms of when it is available for scheduling and its width. Let $\mathsf{a}_\sigma(J_i^b)$ be the earliest time that task J_i^b is available to schedule by our algorithm. As we will see later, if $b = m$, then $\mathsf{a}_\sigma(J_i^b) = 0$ and if $b = r$, then the task will be available at time $2w_J$.

```
Algorithm: Offline Schedule
Simulate the schedules $\sigma_m$ on single $N_m$-speed and $\sigma_r$ on a single $N_r$-speed machine respectively
$w_J \leftarrow \max\{\mathsf{f}^m_{\sigma_m}(J), \mathsf{f}^r_{\sigma_r}(J), p(J^*)\}$
for each job $J$ by $w_J$ increasing do
    for each map task $J^m_i$ of job $J$ do
        Assign $J^m_i$ to the least loaded map machine
    end for
    for each reduce task $J^r_i$ of job $J$ do
        Let $x$ be the earliest available reduce machine
        if $x$ is available before time $w_J$ then
            Idle $x$ till time $2w_J$
        end if
        Assign $J^r_i$ to $x$
    end for
end for
```

LEMMA 2. *For any task J^b_i, it is the case that $\mathsf{f}_\sigma(J^b_i) \leq \mathsf{a}_\sigma(J^b_i) + 2w_J$.*

PROOF. Assume that the statement is false and consider a task J^b_i where $\mathsf{f}_\sigma(J^b_i) > \mathsf{a}_\sigma(J^b_i) + 2w_J$. By definition, this task was available from time $\mathsf{a}_\sigma(J^b_i)$ and the hence schedule σ must have been working on tasks with width at most w_J in the time interval $[\mathsf{a}_\sigma(J^b_i), \mathsf{f}_\sigma(J^b_i) - p(J^*)]$. By definition of width and the assumption, we know $\mathsf{f}_\sigma(J^b_i) - p(J^*) - \mathsf{a}_\sigma(J^b_i) > 2w_J - p(J^*) \geq w_J$. Note that also by definition, σ uses N_b machines for this interval and is busy. Therefore, the above tasks that have width at most w_J represent strictly more than $N_b \cdot w_J$ volume of work. However, the width of a job is at least the completion time of the job in σ_b. This implies that σ_b must complete strictly more than a $N_b \cdot w_J$ volume of work by time w_J. But this is a contradiction since σ_b has a single machine of speed N_b. $\square$

To schedule the map tasks, the algorithm runs the N_m map tasks with the smallest width across the identical machines, breaking ties arbitrarily but consistently. Notice that any map task is scheduled on a single machine since no task will be preempted. Furthermore, map tasks are scheduled only on map machines. By setting $\mathsf{a}_\sigma(J^m_i) = 0$ for all tasks, Lemma 2 yields $\mathsf{f}_\sigma(J^m_i) \leq 2w_J$. Scheduling reduce tasks is less obvious since we have to ensure that each reduce task is processed by only one machine. Consider a reduce task J^r_i. Our algorithm will not consider scheduling this task until time $2w_J$. The algorithm then runs the set of at most N_r reduce tasks that are available to schedule with minimum width. By Lemma 2, it must be the case that all map tasks for a job J are finished by time $2w_J$. Hence, the reduce tasks of a job are scheduled after the map tasks. Further, by definition of this algorithm, after time $2w_J$ the only reduce tasks that become available to schedule have width greater than w_J. This implies that the algorithm will never preempt a reduce task. Thus this schedule assigns each reduce task to only one machine. By once again appealing to Lemma 2 with $\mathsf{a}_\sigma(J^r_i) = 2w_J$ yields $\mathsf{f}_\sigma(J^r_i) \leq 4w_J$. Combining the bounds completes the proof.

We now show an application of the theorem.

COROLLARY 3. *There exists a non-migratory 12-approximation algorithm for flowtime (completion time) in the offline, identical machines, multiple task, map-reduce setting.*

PROOF. It is known that the algorithm Shortest Remaining Processing Time (SRPT) is optimal for average flowtime on a single machine where there is one task per job and no precedence constraints. Knowing that on a single machine having more than one task per job is irrelevant, we can use SRPT to generate the two schedules σ_m and σ_r. Let $\texttt{flow}_{\sigma_m}$ denote SRPT's flowtime for the schedule σ_m and let $\texttt{flow}_{\sigma_r}$ denote SRPT's flowtime for the schedule σ_r. Let OPT be the optimal schedule. Notice that $\texttt{flow}_{\text{OPT}} \geq \max\{\texttt{flow}_{\sigma_m}, \texttt{flow}_{\sigma_r}\}$ and that $\texttt{flow}_{\text{OPT}} \geq \sum_J p(J^*)$. Theorem 1 implies that $\texttt{flow}_\sigma \leq 4(\texttt{flow}_{\sigma_m} + \texttt{flow}_{\sigma_r} + \sum_J p(J^*)) \leq 12\texttt{flow}_{\text{OPT}}$. $\square$

This analysis can be extended to the case when map and reduce machines are indistinguishable.

COROLLARY 4. *There exists a non-migratory 12-approximation algorithm for total flowtime (completion time) in the offline, identical machines, multiple task, map-reduce setting when tasks can be assigned to any machine.*

3.2 Online scheduling

In this section, we consider a similar scheduling instance as in Section 3.1 except, now jobs can arrive over time and the scheduler must be online. Consider a fixed sequence of jobs. As before, our plan is to construct a schedule σ by using σ_m and σ_r, which are schedules of the map and reduce tasks on N_m and N_r machines respectively.

In the online scheduling case, when there are no precedence constraints (no map-reduce phases), there are N identical machines, each of the n jobs has only one task, and the ratio of the maximum job size to the minimum job size is P, it is known that there is an $\Omega(\min\{\log P, \log n/N\})$ lower bound on the competitive ratio for flowtime [25]. Our scheduling model strictly generalizes this setting, therefore this is also a lower bound on flowtime in our setting. Thus, for an algorithm to be $O(1)$-competitive, *resource augmentation* [21] is necessary. I.e., we assume that the schedule σ is given N_m map machines each of speed $(1+\epsilon)$ and N_r reduce machine each of speed $(1+\epsilon)$ where $0 < \epsilon \leq 1$.

THEOREM 5. *Given online schedules σ_r and σ_m, there is a viable online non-migratory $(1+\epsilon)$-resource augmented schedule σ such that $\mathsf{f}_\sigma(J) \leq a_J + \frac{128}{\epsilon^2}\max\{(\max\{\mathsf{f}^m_{\sigma_m}(J), \mathsf{f}^r_{\sigma_r}(J)\} - a_J), p(J^*)\}$.*

To construct the schedule σ, we will use the following algorithm, which employs ideas from [3, 2, 6, 5]. The algorithm simulates the schedules σ_m and σ_r, but needs to be more sophisticated than the offline case, since online load balancing between the machines will be necessary. For a job J, we will define its *width* to be $w_J = \max\{(\max\{\mathsf{f}^m_{\sigma_m}(J), \mathsf{f}^r_{\sigma_r}(J)\} - a_J), p(J^*)\}$. Our algorithm will group tasks according to their width. A job J together with its tasks is said to be in *class* k if $w_J \in [2^k, 2^{k+1})$. The algorithm will maintain the total processing time (volume) of map jobs assigned to a map machine x for each class k. Let $U^{m,x}_{=k}(t)$ denote the total processing time of tasks in class k assigned to map machine x by time t. Likewise, let $U^{r,x}_{=k}(t)$ denote the total processing time of tasks in class k assigned to reduce machine x by time t.

The idea behind the algorithm is to use the schedules σ_r and σ_m to give priorities to the jobs, where the priority of a job is captured by its width. We group tasks geometrically according to their width to balance the volume of work for a specific width across the machines. Notice that the assignment is not based on the current volume of unfinished work, but is based on the total volume of jobs that were assigned to machines up until now. This algorithm is online if σ_m and σ_r are online, since no task for some job J is scheduled by the algorithm unless all tasks for job J are

```
Algorithm: Online Schedule($t$)
Simulate the schedules $\sigma_m$ and $\sigma_r$
if time $t$ is the first time all map tasks for job $J$ are finished in $\sigma_m$ and all reduce tasks for job $J$ are finished in $\sigma_r$ then
    Let $k$ be $J$'s class
    for each map task $J_i^m$ of job $J$ do
        Assign $J_i^m$ to the map machine $x$ where $U_{=k}^{m,x}(t) = \min_y U_{=k}^{m,y}(t)$
        $U_{=k}^{m,x}(t) \leftarrow U_{=k}^{m,x}(t) + p(J_i^m)$
    end for
end if
if time $t$ is the first time that all map tasks for job $J$ are finished in the new schedule $\sigma$ then
    Let $k$ be $J$'s class
    for each reduce task $J_i^r$ of job $J$ do
        Assign $J_i^r$ to the reduce machine $x$ where $U_{=k}^{r,x}(t) = \min_y U_{=k}^{r,y}(t)$
        $U_{=k}^{r,x}(t) \leftarrow U_{=k}^{r,x}(t) + p(J_i^r)$
    end for
end if
On each map and reduce machine run the task assigned to that machine such that the job associated with the task has minimum width.
```

completed in σ_m and σ_r. It can also been seen that the algorithm is non-migratory, since each task is assigned to a single machine, and viable. Thus, we only need to show the guarantee on the job completion time.

Before we begin the analysis, we will introduce a fair bit of notation. As before, let $b \in \{m, r\}$. Since we deal with viable schedules, when we mean machine or task, it will be clear from the context if it is map-related or reduce-related. For each time t, a machine x, and class k we define several quantities. The notation "$\leq k$" will indicate classes 1 to k. Thus, $U_{\leq k}^{b,x}(t)$ is the total volume of tasks in classes 1 to k assigned to machine x. Let $R_{=k}^{b,x}(t)$ denote the remaining processing time of tasks in class k on machine x. Let $P_{=k}^{b,x}(t)$ denote the total volume of tasks in class k machine x has processed up to time t. It can be noted that $P_{=k}^{b,x}(t) = U_{=k}^{b,x}(t) - R_{=k}^{b,x}(t)$. Each of the previously discussed quantities refers to our algorithm. Let $V_{=k}^{*b}(t)$ be the total remaining volume of unsatisfied tasks in class k in the optimal solution's schedule at time t. Let $V_{=k}^{b}(t) = \sum_x R_{=k}^{b,x}(t)$ be the total remaining volume of tasks in class k in our algorithm solution's schedule at time t. We now state some basic facts about these quantities, which will be used to show that our algorithm properly load balanced jobs in each class; the proofs are an extension of those in [2, 6].

LEMMA 6. *At any time t and any two machines x and y, we have the following: (i) $|U_{=k}^{b,x}(t) - U_{=k}^{b,y}(t)| \leq 2^{k+1}$ and $|U_{\leq k}^{b,x}(t) - U_{\leq k}^{b,y}(t)| \leq 2^{k+2}$; (ii) $|P_{\leq k}^{b,x}(t) - P_{\leq k}^{b,y}(t)| \leq 2^{k+2}$; and (iii) $|R_{\leq k}^{b,x}(t) - R_{\leq k}^{b,y}(t)| \leq 2^{k+3}$.*

PROOF. (i) The first inequality is true because the size of a task that belongs to some job J has processing at most $w_J \leq 2^{k+1}$. The second inequality is immediate given the first.

(ii) For the sake of contradiction assume the statement is false. Let t_0 be the first time when $|P_{\leq k}^{m,x}(t_0) - P_{\leq k}^{m,y}(t_0)| = 2^{k+2}$ and a small constant δ such that $|P_{\leq k}^{m,x}(t_0+\delta) - P_{\leq k}^{m,y}(t_0+\delta)| > 2^{k+2}$. This can only occur if machine x processes a task of class $\leq k$ during $I = [t_0, t_0 + \delta]$ while y processes some task of class $> k$. Knowing that each machine always processes the task of minimum width, the machine y must have no tasks in class $\leq k$ during I. This shows that $U_{\leq k}^{m,y}(t_0 + \delta) = P^{m,y}(t_0 + \delta)$. Thus we have,

$$\begin{aligned} & U_{\leq k}^{m,y}(t_0+\delta) = P^{m,y}(t_0+\delta) \\ < \quad & P^{m,x}(t_0+\delta) - 2^{k+2} \leq U_{\leq k}^{m,x}(t_0+\delta) - 2^{k+2}, \end{aligned}$$

knowing that $P_{\leq k}^{m,x}(t_0+\delta) \leq U_{\leq k}^{m,x}(t_0+\delta)$. However, then we have that $U_{\leq k}^{m,y}(t_0+\delta) < U_{\leq k}^{m,x}(t_0+\delta) - 2^{k+2}$, but this is a contradiction to (i). The proof is similar for any two reduce machines. (iii) We know that $R(t) = U(t) - P(t)$. Combining this with (ii), we have that

$$\begin{aligned} & |R_{\leq k}^{m,x}(t) - R_{\leq k}^{m,y}(t)| \\ \leq \quad & |U_{\leq k}^{m,x}(t) - U_{\leq k}^{m,y}(t)| + |P_{\leq k}^{m,x}(t) - P_{\leq k}^{m,y}(t)| \\ \leq \quad & 2 \cdot 2^{k+2} = 2^{k+3}. \end{aligned}$$

The proof is similar for a reduce machine. □

The remainder of the analysis differs from [2, 6]. We first concentrate on showing that each task for each job J is not completed too long after $a_J + w_J$. To do this, our analysis will use the fact that our algorithm is given resource augmentation over the schedules σ_b. We now prove a generic bound on the time gap between the dispatching of a task by our algorithm and its completion. Fix k to be some class and fix $b \in \{m, r\}$. Let job J be the job in class k such that $\mathsf{f}_\sigma^b(J) - a_J$ is maximized. Let task J_i^b be the task for job J that was finished last by our algorithm and let machine x be the machine to which the task J_i^b was assigned. Let the time t_b be the last time before time $\mathsf{f}_\sigma^b(J)$ that our algorithm processed a task of class greater than k on machine x; this implies that machine x is busy processing tasks of class $\leq k$ during $[t_b, \mathsf{f}_\sigma^b(J))$.

LEMMA 7. *For any job J that is in some class k and arrived after time $t_b - \beta_J$, it is the case that $\mathsf{f}_\sigma^b(J) - t_b \leq (2^{k+4} + \beta_J)/\epsilon$ and therefore $(\mathsf{f}_\sigma^b(J) - a_J) \leq (2^{k+4} + \beta_J)/\epsilon$, where $\beta_J > 0$.*

PROOF. By definition of our algorithm, machine x processes a total volume of $(1+\epsilon)(\mathsf{f}_\sigma^b(J) - t_b)$ of work on tasks of class $\leq k$ during $[t_b, \mathsf{f}_\sigma^b(J))$. This and Lemma 6(ii) show that any other machine y also processes a volume of $(1+\epsilon)(\mathsf{f}_\sigma^b(J) - t_b) - 2^{k+3}$ on tasks of class $\leq k$ during $[t_b, \mathsf{f}_\sigma^b(J))$. Further, by definition of time t_b and our algorithm, the machine x has no tasks of class $\leq k$ at time t_b. Thus by Lemma 6(iii) for any machine y we have that $R_{\leq k}^{b,y} \leq 2^{k+3}$. Together this shows that the total volume processed by our algorithm on machines during $[t_b, \mathsf{f}_\sigma^b(J))$ of jobs of class $\leq k$ that were dispatched to machines *after* time t_b is at most $N_b(1+\epsilon)(\mathsf{f}_\sigma^b(J) - t_b) - N_b 2^{k+4}$.

Note that our algorithm does not process any task until the schedule σ_b completes the task. Thus the schedule σ_b must process this volume of work during the interval $[t_b - \beta_J, \mathsf{f}_\sigma^b(J)]$. This implies that $N_b(1+\epsilon)(\mathsf{f}_\sigma^b(J) - t_b) - N_b 2^{k+4} \leq N_b(\mathsf{f}_\sigma^b(J) - t_b + \beta_J)$, since the schedule σ_b has a single machine of speed N_b. However, this implies that $\epsilon(\mathsf{f}_\sigma^b(J) - t_b) \leq 2^{k+4} + \beta_J$, completing the proof. □

Now we apply Lemma 7 to the map tasks and show that our algorithm completes all map tasks in a relatively short amount of time when compared to σ_m. To do this, recall that a map task associated with some job J is dispatched by our algorithm by time $a_J + w_J$. For this case, set $b = m$. The definition of t_m implies $a_J + w_J \geq t_m$. It must be the case that job J arrived after time $t_m - 2^{k+1}$ since the job is of class $\leq k$ and therefore has width $\leq 2^{k+1}$. Hence, Lemma 7 with $\beta_J = 2^{k+1}$ yields that $(\mathsf{f}_\sigma^m(J) - a_J) \leq 2^{k+5}/\epsilon \leq (32/\epsilon)w_J$.

Next, we would like to to show the same thing about reduce tasks. First set $b = r$. Recall the reduce task is dispatched by our algorithm at time $\mathsf{f}_\sigma^m(J)$, the time that all map tasks of J are completed. From the above argument we have $(\mathsf{f}_\sigma^m(J) - a_J) \leq (32/\epsilon)w_J \leq 2^{k+6}/\epsilon$. Thus $a_J \geq \mathsf{f}_\sigma^m(J) - 2^{k+6}/\epsilon \geq t_r - 2^{k+6}/\epsilon$. Appealing to Lemma 7 with $\beta_J = 2^{k+6}/\epsilon$ yields $(\mathsf{f}_\sigma^r(J) - a_J) \leq \frac{2^{k+4}+2^{k+6}/\epsilon}{\epsilon} \leq 2^{k+7}/\epsilon^2 \leq (128/\epsilon^2)w_J$. This completes the proof of Theorem 5.

3.2.1 An application of Theorem 5

Using SRPT to generate the schedules σ_m and σ_r we can show the following.

COROLLARY 8. *There exists a non-migratory $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm for average flowtime in the online, identical machines, multiple task, map-reduce setting where $0 < \epsilon \leq 1$.*

PROOF. It is well know that the online algorithm SRPT is optimal for average flowtime in a standard scheduling instance when there is a single machine. We use SRPT to generate the two schedules σ_m and σ_r. The rest of the proof follows from Theorem 5 and the proof of Corollary 3. □

Considering a simple extension of the previous analysis gives a scheduler that is competitive with resource augmentation when there is no separation between map and reduce machines. When considering this setting, simple extensions of the previous analysis lose a factor of 2 in this speed. This is because it is difficult for the scheduler to decide how to prioritize between map and reduce tasks on a single machine.

REMARK 1. *There exists a non-migratory $(2+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm for average flowtime in the online, identical machines, multiple task, map-reduce setting when tasks can be scheduled on any machine where $0 < \epsilon \leq 1$.*

Chekuri et al. [5] introduce another model of resource augmentation where the online algorithm is provided with $(1+\epsilon)$ as many 1-speed machines. It is not hard to see that section's results hold in this setting as well.

4. FLOW SHOP: PTAS AND QPTAS

In the section we describe our approximation scheme when each job consists of one map task and one reduce task. For the ease of presentation, we assume that there is only one map machine and only one reduce machine; later, we will show how to extend this to the multiple machine case. Our dynamic programming algorithm and proof follow the ideas presented in [1]. We begin by assuming that the processing time of a task is polynomially bounded and we give a quasi-PTAS in this case; this problem is NP-hard even under this assumption [13]. The analysis extends to the case where there is a fixed number of jobs types; our algorithm yields a PTAS in this case. Since there is only one map and one reduce task per job, we let J^b denote job J's map or reduce task, where $b \in \{m, r\}$. We assume preemption is not allowed and unlike the other offline settings we consider, we will allow jobs to arrive over time, but we focus on the objective of total *completion time*. Recall that this is the same as flowtime if jobs arrive at time 0.

The approximation schema that we present is a dynamic program. We first apply a number of structural modifications to the input — these are all inspired by [1], where they have shown to be useful in the problem instance before applying a dynamic program. The core intuition in the design of the dynamic program is the use of Johnson's celebrated algorithm [20] that optimizes makespan as a subroutine to verify feasibility.

Structural modifications. In this section we give an informal description of the various structural modifications applied to the problem in preprocessing. Appendix A outlines the formal lemma statements and proofs corresponding to these modifications.

Let $0 < \epsilon \leq 1/2$. For an arbitrary integer x, define $R_x = (1+\epsilon)^x$. We partition the time interval $(0, \infty)$ into disjoint intervals of the form $I_x = [R_x, R_{x+1})$; we will use I_x to refer to both the interval and the size, $R_{x+1} - R_x$, of the interval. We will often use the fact that $I_x = \epsilon R_x$, i.e., the length of the interval is ϵ times its start time.

We apply the preprocessing in a number of steps, while causing only a $(1+\epsilon)$ factor loss to the optimal value for each step. Each arrival time is rounded to be of the form R_x for some x (Lemma 21). Each task processing time is also rounded to be a power of $(1+\epsilon)$ (Lemma 21). With another $(1+\epsilon)$ factor loss (Lemma 22), we ensure that the task J^m is not started before $a_J + \epsilon p(J^m)$, and task J^r before $\max(\mathsf{f}^m(J^m), \epsilon p(J^r))$. As a result of these modifications, Lemma 24 shows that no task crosses too many intervals. We define time to be stretched by a factor of $1+\epsilon$ when we multiply each interval endpoint by a factor of $1+\epsilon$; this again causes the OPT to degrade by another factor of at most $1+\epsilon$.

We say that a job with its map task running in interval I_J^m and reduce task in interval I_J^r is *small* if $p(J^m) \leq \epsilon I_J^m$ and $\max(p(J^m), p(J^r)) \leq \epsilon I_J^r$. Otherwise the task is large. If a task has a processing time $(1+\epsilon)^x$ we say it is of type x. We call a job J to be of type (x, y) if J^m is of type x and J^r is of type y. Sets of jobs are denoted as vectors, e.g., a vector $\mathbf{s}$ of counts s_{xy} corresponding to each of the types (x, y). For any two sequences $\mathbf{a}$ and $\mathbf{b}$, define $\mathbf{a} \preceq \mathbf{b}$ if $a_{xy} \leq b_{xy}$ for all (x, y). All sequences $\mathbf{s}$ that we consider will satisfy $\mathbf{0} \preceq \mathbf{s}$.

Algorithm. Our algorithm is a dynamic program that creates the schedule interval-wise. The crucial observation is that given a set of map-reduce tasks, we can test the feasibility of scheduling these jobs in an interval by using Johnson's algorithm [20] for minimizing makespan in a two-stage two-machine flowshop. Our algorithm also guesses the last $\frac{25}{\epsilon^{10}}$ tasks that will be completed in the optimal schedule and does not consider these tasks in the dynamic program.

At time R_t, let $\mathbf{n}$ be the set of all jobs that have arrived up until now and let $\mathbf{c}$ denote the set of completed jobs. Define a job to be *partially done* if the map task is completed. Let $\mathbf{p}$ denote the set of partially done jobs at R_t. The dynamic program at iteration t will select a set of tasks to be scheduled in some interval I_{t+1}. This set could consist of both the tasks of a job, only the map task of a job, or a reduce task whose corresponding map task has been completed. If a reduce task is scheduled in an interval and the map task for this job was scheduled in an earlier interval, we will implicitly incorporate a map task of length zero to precede this reduce. We now describe the algorithm.

(1) Assume the arrival time of each job to be $\max(a_J, \frac{p(J^m)}{\epsilon^2})$. Also assume that the reduce task of J is not available before $p(J^r)/\epsilon^2$ (and of course, before completing the corresponding map task).

(2) If $\mathbf{n}$ jobs have arrived up until time interval I_t, define $C[\mathbf{c}, \mathbf{p}, \mathbf{n}, t]$ be the total completion time when scheduling all tasks in $\mathbf{c}$ completely and just the map tasks for jobs in $\mathbf{p} - \mathbf{c}$ during $[0, R_{t+1})$. Since the tasks in $\mathbf{p} - \mathbf{c}$ are not completed, we do not include their completion time in the total. If $\mathbf{n}$ does not correspond to the number of arrived jobs, or if $\mathbf{c} \preceq \mathbf{p} \preceq \mathbf{n}$ is not satisfied, then

$C[\mathbf{c}, \mathbf{p}, \mathbf{n}, t] = \infty$. We also define $C[\mathbf{0}, \mathbf{0}, \mathbf{0}, -1] = 0$. We use the following dynamic program for all but the last $\frac{25}{\epsilon^{10}}$ that we guessed.

(3) Suppose $I_{t+1} = [R_{t+1}, R_{t+2})$ be the current interval. Let $\mathbf{v}$ be the sequence of new jobs that arrive at the beginning of this interval. Let $\mathbf{q}, \mathbf{m}$, and $\mathbf{r}$ be such that $\mathbf{m} + \mathbf{q} \preceq \mathbf{n} + \mathbf{v} - \mathbf{p}$ and $\mathbf{r} \preceq \mathbf{p} - \mathbf{c}$. Intuitively, $\mathbf{q}$ denotes the set of jobs for which we are going to schedule both the map and the reduce tasks in I_{t+1}, $\mathbf{m}$ the set of jobs for which we are going to schedule only the map task, and $\mathbf{r}$ the set of jobs for which we schedule only the reduce. Given these sets our algorithm would like to schedule the tasks in $\mathbf{q} + \mathbf{m} + \mathbf{r}$ during the interval I_{t+1}. We verify the feasibility of this schedule using Johnson's algorithm [20] for optimal makespan in two-stage two machine setting. Define the scheduling cost for this interval to be $Q(\mathbf{q}, \mathbf{m}, \mathbf{r}, I_{t+1}) = R_{t+2} \sum_{ij}(q_{ij} + r_{ij})$ if Johnson's algorithm returns a feasible schedule and $Q(\mathbf{q}, \mathbf{m}, \mathbf{r}, I_{t+1}) = \infty$ else. Notice that this cost function assumes that completion time of the jobs scheduled in interval I_{t+1} is R_{t+2}. This is because each task completed during I_{t+1} is finished before time R_{t+2}. We justify this in the analysis by showing that this increases the schedule's cost by at most a $(1 + \epsilon)$ factor. Thus, the dynamic program computes

$$C[\mathbf{c}, \mathbf{p}, \mathbf{n} + \mathbf{v}, t + 1] = \min_{\mathbf{q},\mathbf{m},\mathbf{r}} C[\mathbf{c} - \mathbf{q} - \mathbf{r}, \mathbf{p} - \mathbf{m} - \mathbf{q} - \mathbf{r}, \mathbf{n}, t] + Q(\mathbf{q}, \mathbf{m}, \mathbf{r}, I_{t+1}).$$

(4) After all jobs are scheduled using the dynamic program, enumerate the possible ways to finish the final $\frac{25}{\epsilon^{10}}$ jobs that were guessed.

Let $P = \max_J p(J^*)$ be the processing time of the largest task and $T = \sum_J p(J^m) + p(J^r)$ be an upper-bound on the schedule length. The time taken by the above procedure is given by $n^{O(\log_{1+\epsilon}(P))} \log T$ for fixed ϵ. As a first step in establishing our claims, we motivate our algorithm with the following lemma whose proof follows the proof of Lemma 3.1 in [1].

LEMMA 9. *If in the optimal solution, all jobs are small, then the above algorithm gives a $(1 + 3\epsilon)$-approximate solution.*

PROOF. Let $R(I)$ denote the starting point of interval I. If the jobs are small in the optimal solution, then from Lemma 22 it follows that $s^m_*(J) \geq R(I^m_J) = \frac{I^m_J}{\epsilon} \geq p(J^m)/\epsilon^2$. Also, similarly, $s^r_*(J) \geq R(I^r_J) \geq p(J^r)/\epsilon^2$. Hence, changing the arrival times does not matter for the optimal solution. Also, since each job is small for the interval, by stretching each interval by a factor of $(1 + \epsilon)$ we ensure that all jobs finish completely inside their interval (Lemma 25). Given this condition, we first claim that the dynamic program computes a $(1 + \epsilon)$ approximation to the optimal solution. Consider the $C[\mathbf{c}, \mathbf{p}, \mathbf{m}, \mathbf{n}, t - 1]$ that corresponds to the optimal algorithm's choice of $\mathbf{c}, \mathbf{p}, \mathbf{m}$ up until interval I_{t-1} — by inductive hypothesis this is a $1 + \epsilon$ approximation. If the optimal solution chooses the task sets from the jobs in $\mathbf{q}^* + \mathbf{m}^* + \mathbf{r}^*$ to schedule in this interval $I_t = [R_t, R_{t+1})$, then it incurs at least $R_t \sum_{ij}(q^*_{ij} + r^*_{ij})$. By looking at all possible assignments, our algorithm makes a choice such that the resulting minimum value of C is at most $(1 + \epsilon)$ of the value incurred by the OPT at the end of I_t. This is because Johnson's algorithm ensures that the tasks are scheduled within the interval I_t. Further, a task's completion time is at most R_{t+1} if the task was scheduled during I_t. Hence at the end of the dynamic program, we still have a $(1 + \epsilon)$ approximation.

Since stretching the time by a factor of $(1 + \epsilon)$ increases the total completion time by the same factor, and we pay another $(1 + \epsilon)$ in the dynamic program. Overall we have a $(1 + 3\epsilon)$ factor approximation. □

Let $t_h = 2\epsilon^{10}\text{OPT}$ be a threshold time. Notice that in the optimal solution there are at most $\frac{1}{2\epsilon^{10}}$ jobs that are completed after time t_h. The goal of the next lemma is to show that large jobs can be postponed. To do this, we show that there is an approximate optimal solution where either a job is small or it is done after time t_h. To prove the lemma, we consider shifting large jobs in the optimal solution to intervals where the jobs are small. We create room for these jobs by expanding the interval lengths by a factor of $(1 + O(\epsilon))$. We also show that shifting the large jobs does not effect the value of the optimal solution by more than a factor of $(1 + O(\epsilon))$. The proof of the lemma is quite technical since we have to be careful on how map and reduce tasks for the same job are shifted.

LEMMA 10. *There exists a $(1 + 13\epsilon)$-approximate optimal schedule in which, for each job J, $\mathsf{s}^m_*(J) \geq \min(\frac{p(J^m)}{\epsilon^2}, t_h)$ and $\mathsf{s}^r_*(J) \geq \min(\max(\mathsf{f}^m_*(J), p(J^r)/\epsilon^2), t_h)$.*

PROOF. The proof argument is similar to that of Lemma 3.2 in [1]. Fix some job J. Let $I^m = [R^m, S^m)$ be the interval the J^m is processed during and $I^r = [R^r, S^r)$ be the interval J^r is processed during. If J is small, then $p(J^m) \leq \epsilon I^m \leq \epsilon^2 R^m \leq \epsilon^2 s^m_*(J)$. Also, for the reduce job, $s^r_*(J) \geq f^m_*(J)$. Furthermore, by the smallness of J, $p(J^r) \leq \epsilon I^r \leq \epsilon^2 R^r \leq \epsilon^2 s^r_*(J)$. Hence, if J is small both the conditions are satisfied.

The two cases we need to worry about are i) $\max(p(J^r), p(J^m)) > \epsilon I^r$ and ii) $p(J^m) > \epsilon I^m$. To handle both of these cases, we will show how jobs in the optimal solution can be shifted to satisfy the lemma.

We handle case (i) first. The first subcase (i.A) is $p(J^r) > \epsilon I^r$. Let $s = \log_{1+\epsilon}(\frac{1}{\epsilon^6})$. We move the map and reduce task of J to the interval $I' = I^r + s = [R', R'')$. In this case, if $s^m(J)$ and $s^r(J)$ denotes the new starting point of the map and reduce tasks, then similar to the argument in Lemma 3.2 of [1], we have that $p(J^m) \leq s^m_*(J)/\epsilon \leq \epsilon^5 R' \leq \epsilon^5 s^m(J)$. We can show similarly $p(J^r) < \epsilon^5 s^r(J)$. We now need to show that we can fit the shifted jobs in this interval. Since $p(J^r) > \epsilon I_r$, there are at most $\frac{1}{\epsilon}$ such jobs from interval I_r that move into interval I', where each of then requires a total time of $p(J^m) + p(J^r) < 2\epsilon^4 I'$. Hence, the total time required by these shifted jobs in interval I' is $2\epsilon^3 I'$. By stretching time by a $(1 + 2\epsilon)$ factor, we can easily accommodate these jobs. Hence the condition is fulfilled.

The only case where the above construction will not work is when there is a single task (either in map or reduce machine) that spans the entire interval I'. Then, we use Lemma 25 to say that we could as well insert this $\epsilon I'$ space at the beginning of the crossing job. That is, shift the single crossing task and place the space before the task. If the new interval is I'', as at most $\log_{1+\epsilon} \frac{1}{\epsilon}$ intervals are crossed by the job, $I'' = \epsilon I$, and thus each of $p(J^m)$ and $p(J^r)$ are at most $\epsilon^2 I''$ and are small.

For the second subcase (i.B), $p(J^m) > \epsilon I^r$. We again move the entire job to $I' = I^r + s$. Now, we need to justify as before that there are small number of such jobs being shifted to I'. In this case, since both the map and reduce happen by the interval I^r and the interval lengths are geometrically increasing, the total time taken by such jobs is at most I^r/ϵ and thus there can be at most $\frac{1}{\epsilon^2}$ of such jobs. After shifting to the interval I', such jobs take up at most $\frac{1}{\epsilon^2} \cdot 2\epsilon^3 I' \leq 2\epsilon I'$. The case where there is a single task covering I' can be handled similar as before.

Next, we handle case (ii). The first subcase (ii.A) is when $I' = I^m + s \leq I^r$. In this case, only the map task is moved to the interval I'.

In case (ii.B), $I' = I^m + s > I^r$. In this case, we move both the map and the reduce to the interval I'. The number of jobs shifted to interval I' can be bounded now by $\frac{1}{\epsilon}$ by the fact that $p(J^m) > \epsilon I^m$. Hence, again by stretching time by a $1 + 2\epsilon$ factor, we can accommodate all jobs.

Now we bound the cost of the solution after performing these shifting operations. We might need to expand by schedule twice by factors of $1 + 2\epsilon$ because of the two cases – this increases the cost by a factor of $1 + 2\epsilon$. By doing the shift, we increase the completion time of any job ending in R_x to at most $R_{x+s+1} \leq \frac{(1+\epsilon)R_x}{\epsilon^6}$ factor, and there are at most $\frac{2}{\epsilon^2}$ jobs being shifted overall. The last interval from which jobs are being shifted ends at t_h. Thus, the total completion time of the shifted jobs is

$$\begin{aligned}\sum_{R_x < t_h} \frac{2}{\epsilon^2} \frac{(1+\epsilon)R_x}{\epsilon^6} &\leq \frac{2t_h}{\epsilon^8} \sum_{i \geq 0} \frac{1}{(1+\epsilon)^i} \\ &\leq \frac{2t_h}{\epsilon^8} \frac{(1+\epsilon)^2}{\epsilon} \\ &\leq \epsilon(1+\epsilon)^2 \text{OPT} < 2\epsilon\text{OPT},\end{aligned}$$

since $t_h = 2\epsilon^{10}\text{OPT}$. Since we stretched time by a $1 + \epsilon$ factor for rounding processing times, $1 + \epsilon$ factor for Lemma 22, and $1 + 2\epsilon$ factor for this lemma, and added a 2ϵ cost, we have a $1 + 13\epsilon$ factor approximation overall. □

By combining the Lemmas 9 and 10, we now show how to get a $(1 + O(\epsilon))$ approximation. Note that if $\mathsf{s}_*^m(J) \geq \frac{p(J^m)}{\epsilon^2}$ and $\mathsf{s}_*^r(J) \geq \max(\mathsf{f}_*^m(J), p(J^r)/\epsilon^2)$, then the job J is small when run. By the Lemma 10, we have a $(1 + 13\epsilon)$ approximate solution for these jobs. Now suppose we fix the positions of the last $\frac{25}{\epsilon^{10}}$ tasks. Given the fixed position of these non-small tasks, the dynamic program will still find a $(1 + 3\epsilon)$ approximate solution to the current OPT, and hence a $(1 + 13\epsilon)(1 + 3\epsilon) \leq (1 + 50\epsilon)$ approximate solution overall assuming $\epsilon \leq 1/2$. Hence, if we run this dynamic program, at the end of time $t_h = 2\epsilon^{10}\text{OPT}$, the number of jobs left is at most $(1 + 50\epsilon)\frac{1}{2\epsilon^{10}} \leq \frac{25}{\epsilon^{10}}$. Hence, all tasks scheduled by the dynamic program finish by time t_h and the last jobs were enumerated.

THEOREM 11. *For the offline case with arrival times, and one map and one reduce task per job on identical machines, there exists a $1 + O(\epsilon)$ approximate algorithm that runs in time $n^{O(\frac{1}{\epsilon^{10}})}(n^{\log_{1+\epsilon}(P)} \log T + \frac{25}{\epsilon^{10}}!)$.*

Notice that this theorem gives a quasi-polynomial time algorithm when maximum processing time a task is polynomially bounded. Now consider the case where there are a constant δ number of tasks types. In this case, the dynamic program needs to enumerate over each of the task types. Thus, for this case we have the following theorem.

THEOREM 12. *For the offline case with arrival times, one map and one reduce task per job on identical machines and there are δ task types, there exists a $1 + O(\epsilon)$ approximate algorithm that runs in time $n^{O(\frac{1}{\epsilon^{10}})}(n^\delta \log T + \frac{25}{\epsilon^{10}}!)$.*

The final case we consider is when there are multiple map and reduce machines. Notice that in the dynamic program, Johnson's algorithm was used to actually schedule the tasks assigned to an interval. This is the only part of the analysis where we used the fact that there was a single machine at each stage. We can consider the case where there are multiple map and reduce machines by using the PTAS for maximum completion time in the two stage flexible flow shop problem to determine how to schedule tasks within an interval [31]. By stretching time by another factor of $(1 + \epsilon)$ it can be ensured that this PTAS is able to fit all jobs into an interval.

Lastly we remark that although the run-time of the (Q)PTAS might seem daunting at first sight, Hepner and Stein [17] have already demonstrated how a related algorithm can be implemented in practice.

5. FLEXIBLE FLOW SHOP: UNRELATED MACHINES CASE

In this section we consider the most general multiple machine scheduling model known as the *unrelated* machines model. In the unrelated machines model, the processing time of a task depends on the machine to which the task is assigned. In general, for $x \neq y$, $p_x(\cdot)$ and $p_y(\cdot)$ may be uncorrelated. In fact, the processing time of a task may be ∞ on some machines, capturing the case where a task cannot be assigned to a specific machine, e.g., when there is not enough memory on a machine to run a specific task.

Due to the generality of the unrelated machines model, it seems difficult to find an algorithm that performs well when there are multiple map and reduce tasks per job. Working towards the goal of finding good algorithms for multiple task instances, we consider the single task case in this section. This is the FFS problem generalized to unrelated machines.

Let σ_m be a non-migratory schedule on the unrelated map machines for only map jobs and let σ_r be a non-migratory schedule on the unrelated reduce machines for only reduce jobs. Unlike the identical machine cases, these two schedules are *not* on a single machine, but rather they are on the original set of machines.

We assign each job J width $w_J = \max\{\mathsf{f}_{\sigma_m}(J), \mathsf{f}_{\sigma_r}(J)\}$. Notice that in this case, the width of a job only depends on the simulated schedules and does not include the maximum processing time of task.

5.1 Offline scheduling

First we address the case where the scheduler is offline and all jobs arrive at time 0.

THEOREM 13. *Given σ_m and σ_r, there is a non-migratory viable schedule σ such that all tasks for job J are completed by time $2\max\{\mathsf{f}_{\sigma_m}(J), \mathsf{f}_{\sigma_r}(J)\}$.*

Our algorithm simulates the schedules σ_m and σ_r. The algorithm assigns each map (reduce) task to the same machine it was processed on in the schedule σ_m (σ_r). A map machine runs the task with shortest width assigned to it. At any time, a reduce machine only runs a reduce task whose corresponding map task is complete. The algorithm always runs the reduce task with smallest width amongst the reduce tasks which are available to schedule. It is easy to check that this schedule is non-migratory and viable. To bound the completion time of the tasks, first we consider the map tasks. Since there is only one map and one reduce task per job, we drop the index of the tasks. Thus, J^b denotes the task for job J and $\mathsf{f}_\sigma(J^b)$ denote the time the task of job J is completed in σ. Again, we give a generic bound on the completion times of tasks based on their earliest availability and width. Recall that for a task J^b, $b \in \{m, r\}$, $\mathsf{a}_\sigma(J^b)$ is the earliest time when the task is available to the schedule σ.

LEMMA 14. *For any task J^b, $\mathsf{f}_\sigma(J^b) \leq \mathsf{a}_\sigma(J^b) + w_J$.*

PROOF. For the sake of contradiction, assume that the lemma is false. Consider a task J^b where $\mathsf{f}_\sigma(J^b) > \mathsf{a}_\sigma(J^b) + w_J$. We know that this task has been available to schedule since time $\mathsf{a}_\sigma(J^b)$. By definition of our algorithm, this implies that the machine task J^b is assigned to has been busy processing jobs with width at most w_J during $[\mathsf{a}_\sigma(J^b), \mathsf{f}_\sigma(J^b)]$. By definition of our algorithm and width, the schedule σ_b must processes strictly more than a w_J volume of work on this machine by time w_J, a contradiction. $\Box$

By using the fact $\mathsf{a}_\sigma(J^m) = 0$ for all map tasks, we have that for any map task J^m, $\mathsf{f}_\sigma(J^m) \leq w_J$. Similarly, the completion time of a reduce task is bounded by using the above lemma that $\mathsf{a}_\sigma(J^r) = \max_{J^m \in J} \mathsf{f}_\sigma(J^m) \leq w_J$. We have now bounded the completion times of the jobs. Using Theorem 13, we can construct an approximation algorithm for average flowtime in the scheduling setting.

COROLLARY 15. *There exists a non-migratory 6-approximation algorithm for flowtime (completion time) in the offline, unrelated machines, single task, map-reduce setting.*

PROOF. Skutella in [32] gave a $\frac{3}{2}$-approximation algorithm for minimizing the total completion time on unrelated machines where there is one task per job, no precedence constraints and all jobs arrive at time 0. Since there are only one map and one reduce task per job in our scheduling instance, in the scheduling instances the schedules σ_m and σ_r consider there is one task per job. Thus, the algorithm of Skutella can be used to construct the schedules σ_m and σ_r. Since $\mathtt{flow}_{\text{OPT}} \geq \frac{2}{3}\max\{\mathtt{flow}_{\sigma_m}, \mathtt{flow}_{\sigma_r}\}$, Let F_σ denote the total flow Theorem 13 implies that $\mathtt{flow}_\sigma \leq 2(\mathtt{flow}_{\sigma_m} + \mathtt{flow}_{\sigma_r}) \leq 6\mathtt{flow}_{\text{OPT}}$. $\Box$

5.2 Online scheduling

We now consider the case when jobs arrive over time in an online fashion. In the online unrelated machines setting, even when there are no precedence constraints and all jobs consist of one task, it is known that no online algorithm has bounded competitive ratio for the objective of flowtime [14]. Thus, like in the identical machines setting, we resort to resource augmentation.

THEOREM 16. *Given online non-migratory schedules σ_m and σ_r, there is a viable online non-migratory $(1+\epsilon)$-resource-augmented schedule σ such that all tasks for job J are completed by time $a_J + \frac{4}{\epsilon^2}(\max\{\mathsf{f}_{\sigma_m}(J), \mathsf{f}_{\sigma_r}(J)\} - a_J)$.*

Our algorithm simulates the schedules σ_m and σ_r similarly to the offline algorithm. It is easy to check that the scheduler is online, non-migratory, and viable.

We first present a common lemma that we will use in bounding both the map and the reduce finish times.

LEMMA 17. *Let $\alpha > 0$. Suppose that task J^b, $b \in \{m, r\}$, is available for scheduling by our schedule σ at time $a_J + \alpha w_J$. Then it is the case that $\mathsf{f}_\sigma(J^b) \leq a_J + \frac{2\alpha}{\epsilon} w_J$.*

PROOF. Let x be the machine (map or reduce) that the task J^b is assigned to. Let time t_b be the earliest time such that every task processed during $[t_b, \mathsf{f}_\sigma(J^b)]$ has width at most w_J on machine x. By the given condition, we know that task J^b is available to schedule at time $a_J + \alpha w_J$. Knowing that our algorithm always schedules the task with minimum width on each machine, we have that $t_b \leq a_J + \alpha w_J$.

We also claim that any task scheduled during $[t_b, \mathsf{f}_\sigma(J^b)]$ arrived at earliest $t_b - \alpha w_J$. This is because any task J' scheduled during $[t_b, \mathsf{f}_\sigma(J^b)]$ has width $w_{J'} \leq w_J$, and hence by given assumption has arrival time $a_{J'} \geq t_b - \alpha w_{J'} \geq t_b - \alpha w_J$.

Our algorithm has speed $1+\epsilon$, thus the algorithm processes $(1+\epsilon)(\mathsf{f}_\sigma(J^b) - t_b)$ volume of work in total during $[t_b, \mathsf{f}_\sigma(J^b)]$. All of the tasks processed by our algorithm during $[t_b, \mathsf{f}_\sigma(J^b)]$ must be processed on the interval $[t_b - \alpha w_J, \mathsf{f}_\sigma(J^b)]$ by the schedule σ_b on machine x itself. This is because all of the tasks processed by σ during $[t_b, \mathsf{f}_\sigma(J^b)]$ arrived no earlier than $t_b - \alpha w_J$ by the previous argument. Further, our algorithm assigns any task to the same machine the schedule σ_b processed the task on. Therefore it must be the case that $(1+\epsilon)(\mathsf{f}_\sigma(J^b) - t_b) \leq \mathsf{f}_\sigma(J^b) - t_b + \alpha w_J$. This implies that $\mathsf{f}_\sigma(J^b) \leq t_b + \frac{\alpha}{\epsilon} w_J \leq a_J + \alpha w_J + \frac{\alpha}{\epsilon} w_J \leq a_J + \frac{2\alpha}{\epsilon} w_J$ since $\epsilon < 1$. $\Box$

We now use the above Lemma to deduct the following two corollaries. The first corollary is obtained from the above Lemma, combined with the fact that by construction of our algorithm, the map task J_m is available to σ at time $a_J + w_J$.

COROLLARY 18. *For any map task J^m it is the case that $\mathsf{f}_\sigma(J^m) \leq a_J + \frac{2}{\epsilon} w_J$.*

Similarly, using the above corollary, the reduce task J^r is available when all the corresponding maps are finished, and hence at time $a_J + \frac{2}{\epsilon} w_J$. Using this bound in Lemma 17, we have the following corollary.

COROLLARY 19. *For any reduce task J^r it is the case that $\mathsf{f}_\sigma(J^r) \leq a_J + \frac{4}{\epsilon^2} w_J$.*

5.2.1 Application of Theorem 16

First we consider the objective of total flowtime. As mentioned, it is known that no algorithm has bounded competitive ratio without resource augmentation [14]. Since our algorithm only uses ϵ resource augmentation, our algorithm is constant competitive when given the minimum advantage over the adversary.

COROLLARY 20. *There exists a non-migratory $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^4})$-competitive online algorithm for average flowtime in the online, unrelated machines, single task, map-reduce setting.*

PROOF. In a recent breakthrough result Chadha et al. showed a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive online non-migratory algorithm for average flowtime in the unrelated machine setting when there is one task per job and no precedence constraints [4]. Using this algorithm we can generate the schedules σ_m and σ_r. The result of Chadha et al. implies that $\text{OPT} \geq \Omega(\epsilon^2)\max\{\mathtt{flow}_{\sigma_m}, \mathtt{flow}_{\sigma_r}\}$. Theorem 16 shows that $\mathtt{flow}_\sigma \leq \frac{4}{\epsilon^2}(\mathtt{flow}_{\sigma_m} + \mathtt{flow}_{\sigma_r})$. Thus, $\mathtt{flow}_\sigma \leq O(\frac{1}{\epsilon^4})\mathtt{flow}_{\text{OPT}}$. $\Box$

6. REFERENCES

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, and C. Stein. Approximation schemes for minimizing average weighted completion time with release dates. In *Proc. 40th FOCS*, pages 32–44, 1999.

[2] N. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. *Algorithmica*, 47(3):253–268, 2007.

[3] N. Bansal, R. Krishnaswamy, and V. Nagarajan. Better scalable algorithms for broadcast scheduling. In *Proc. 37th ICALP*, pages 324–335, 2010.

[4] J. S. Chadha, N. Garg, A. Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In *Proc. 41st STOC*, pages 679–684, 2009.

[5] C. Chekuri, A. Goel, S. Khanna, and A. Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *Proc. 36th STOC*, pages 363–372, 2004.

[6] C. Chekuri and B. Moseley. Online scheduling to minimize the maximum delay factor. In *Proc. 19th SODA*, pages 1116–1125, 2009.

[7] F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in map-reduce. In *Proc. 19th WWW*, pages 231–240, 2010.

[8] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Proc. 20th NIPS*, pages 281–288, 2006.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *C. ACM*, 51:107–113, 2008.

[10] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. In *Proc. 19th SODA*, pages 710–719, 2008.

[11] M. J. Fischer, X. Su, and Y. Yin. Assigning tasks for efficiency in hadoop: Extended abstract. In *Proc. 22nd SPAA*, pages 30–39, 2010.

[12] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson. Statistics-driven workload modeling for the Cloud. In *Proc. Data Engineering Workshops at 26th ICDE*, pages 87–92, 2010.

[13] M. R. D. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1:1171–129, 1976.

[14] N. Garg and A. Kumar. Minimizing average flow-time : Upper and lower bounds. In *FOCS*, pages 603–613, 2007.

[15] N. Garg, A. Kumar, and V. N. Muralidhara. Minimizing total flow-time: The unrelated case. In *Proc. 19th ISAAC*, pages 424–435, 2008.

[16] L. A. Hall. Approximability of flow shop scheduling. In *Proc. 36th FOCS*, pages 82–91, 1995.

[17] C. Hepner and C. Stein. Implementation of a PTAS for scheduling with release dates. *Algorithm Engineering and Experimentation*, pages 202–215, 2001.

[18] S. Im and B. Moseley. An online scalable algorithm for minimizing ℓ_k-norms of weighted flow time on unrelated machines. In *Proc. 21st SODA*, pages 95–108, 2011.

[19] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *Proc. 22nd SOSP*, pages 261–276, 2009.

[20] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:69–81, 1954.

[21] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[22] U. Kang, C. E. Tsourakakis, A. Appel, C. Faloutsos, and J. Leskovec. HADI: Fast diameter estimation and mining in massive graphs with Hadoop. Technical Report CMU-ML-08-117, CMU, 2008.

[23] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In M. Atallah, editor, *Handbook on Algorithms and Theory of Computation*, chapter 34. Chapman and Hall/CRC, 1999.

[24] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Proc. 20th SODA*, pages 938–948, 2010.

[25] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. *JCSS*, 73(6):875–891, 2007.

[26] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Number 7 in Synthesis Lectures on Human Language Technologies. Morgan and Claypool, 2010.

[27] K. Pruhs, J. Sgall, and E. Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. CRC Press, 2004.

[28] T. Sandholm and K. Lai. MapReduce optimization using regulated dynamic prioritization. In *Proc. 11th SIGMETRICS*, pages 299–310, 2009.

[29] P. Schuurman and G. J. Woeginger. Flowshop and jobshop schedules: Complexity and approximation. *Operations Research*, 26:136–152, 1978.

[30] P. Schuurman and G. J. Woeginger. Polynomial time approximation algorithms for machine scheduling: ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999.

[31] P. Schuurman and G. J. Woeginger. A polynomial time approximation scheme for the two-stage multiprocessor flow shop problem. *TCS*, 237(1-2):105–122, 2000.

[32] M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *J. ACM*, 48(2):206–242, 2001.

[33] J. L. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin. FLEX: A slot allocation scheduling optimizer for MapReduce workloads. In *Middleware*, pages 1–20, 2010.

[34] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proc. 5th EuroSys*, pages 265–278, 2010.

[35] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I. Stoica. Improving MapReduce performance in heterogeneous environments. In *Proc. USENIX OSDI*, 2008.

APPENDIX

A. FLOW SHOP: PTAS AND QPTAS STRUCTURAL LEMMAS

In this section we state a few structural lemmas for our PTAS. These lemmas are adopted from [1] and the proofs are almost identical.

LEMMA 21 ([1]). *With $1+\epsilon$ loss we can assume that all processing and arrival times are integer powers of $1+\epsilon$.*

LEMMA 22 ([1]). *With $1+O(\epsilon)$ loss, we can ensure that a job arrives after $\epsilon p(J^m)$ and a reduce task starts later than $p(J^m)+\epsilon p(J^r)$. Further, the arrival of jobs occur only at R_x for some x.*

PROOF. The same as in [1]. The second part follows by the $1+\epsilon$ stretch and since the reduce task cannot start earlier than map completion. □

DEFINITION 23. *We say that a task* crosses *an interval I_x if its execution overlaps with I_x but it is not contained in I_x completely.*

LEMMA 24 ([1]). *Each* task *crosses at most $s = \lceil \log_{1+\epsilon}\left(1+\frac{1}{\epsilon}\right)\rceil$ intervals.*

PROOF. The proof follows the same idea as in [1]. Suppose that a task of job J starts in interval $I_x = [R_x, R_{x+1})$. Since $R_x \geq s^b(J)$ for both $b = \{m, r\}$, i.e., both map and reduce tasks and $s^b(J) \geq \epsilon p^b(J)$ by Lemma 22, we have $I_x = \epsilon R_x \geq \epsilon^2 p^b(J)$. The s intervals following x sum in size to $I_x/\epsilon^2 \geq p^b(J)$. □

LEMMA 25 ([1]). *With $1+\epsilon$ loss we can restrict our attention to schedules in which no small task crosses an interval.*