# Fair Online Scheduling for Selfish Jobs on Heterogeneous Machines

Sungjin Im
University of California, Merced
Merced, CA 95343
sim3@ucmerced.edu

Janardhan Kulkarni
Microsoft Research
Redmond, WA 98052
jakul@microsoft.com

## ABSTRACT

Scheduling jobs on multiple machines has numerous applications and has been a central topic of research in the scheduling literature. Recently, much progress has been made particularly in online scheduling with the development of powerful analysis tools. In this line of wok a centralized scheduler typically dispatches jobs to machines to exploit the given resources the best to achieve the best system performance which is measured by a certain global scheduling objective. While this approach has been very successful in attacking scheduling problems of growing complexity, the underlying assumption that jobs follow a centralized scheduler may not be realistic in certain scheduling settings.

In this paper we initiate the study of online scheduling for selfish jobs in the presence of multiple machines. Selfish behavior of jobs is a common aspect observed in the absence of a centralized scheduler. We explore this question in the unrelated machines setting, arguably one of the most general multiple machine models. In this model each job can have a completely different processing time on each machine. Motivated by several practical scenarios, we assume that when a job arrives it *chooses the machine that completes the job the earliest* i.e. minimizes the flow time of the job. The goal is to design a local scheduling algorithm on each machine with the goal of minimizing the total (weighted) flow time. We show that the algorithm Smoothed Latest Arrival Processor Sharing, which was introduced in a recent work by Im et al. [27, 28], yields an $O(1/\epsilon^2)$-competitive schedule when given $(1+\epsilon)$ speed. We also extend our result to minimize total flow-time plus energy consumed. To show this result we establish several interesting properties of the algorithm which could be of potential use for other scheduling problems.

## 1. INTRODUCTION

Processing jobs using multiple resources is a fundamental scheduling problem that arises in various forms in many disciplines. Such settings are prevalent and found in server farms and data centers which consist of a large number of machines. At a smaller scale, chips are equipped with multiple heterogeneous cores, and the number of cores per chip is expected to grow exponentially to deliver high throughput with less energy consumption. Routers and middleboxes are common computing resources to process packets in network. When resources are viewed as abstract machines, numerous applications can be captured by this setting.

Not surprisingly, multiple machines scheduling has been a central topic of research in the theoretical scheduling community. One of the most general multiple machines settings is the unrelated machine model. In this model, there are $m$ machines and $n$ jobs, and each job $j$ arrives at time $r_j$. Each job $j$ may have an arbitrary processing requirement (size) of $p_{ij}$ on each machine $i$ and is associated with weight $w_j$ which stands for its importance. The unrelated machines setting captures and generalizes identical machines and related machines where machines run in different speeds. When multiple jobs compete to be processed earlier, one needs to prioritize between jobs according to a global scheduling objective. Perhaps the most popular objective in online setting is to minimize the average (or equivalently total) weighted flow time of jobs, where the flow time of a job $j$ is defined as the length of time a job waits before it is completed since its arrival, multiplied by its weight $w_j$.

The problem of minimizing the average weighted flow time of jobs on unrelated machines has been extensively studied both in online and offline settings; we give an extensive survey of known results in Section 1.4. Much of the approximation and online algorithms literature on the problem, however, has taken a system-centric and centralized view of the problem. In this view, the jobs arriving into the system are assumed to be altruistic, and are at the discretion of a centralized scheduler that attempts to achieve high overall system performance. Although this model is a fair abstraction of scheduling jobs within an operating system or across different cores of a single machine, it does not capture scheduling constraints arising in data centers or other cloud services [34]. In many applications, implementing centralized and monolithic scheduling algorithms is not always possible. Furthermore, in many scenarios jobs behave selfishly by choosing machines that give them the earliest completion time. The selfish behavior of jobs may arise because the jobs are submitted by self-interested clients or simply as a consequence of a dispatch rule which is decentralized and greedy. The examples for latter case include, webpage requests are usually dispatched to server farms that are least loaded[12], and DNS query requests are routed to DNS servers with the least service time. Selfish behavior may impact the performance of the system and has to be taken into consideration when designing scheduling policies.

Much of the work on selfish scheduling of jobs has been considered mainly in the setting of coordination games, first introduced in a seminal paper [16]. In this setting, each machine declares a strongly local scheduling policy. The strategy of each job consists of choosing the machine which minimizes its own completion time. Each job knows the processing lengths of other jobs and hence



*SPAA '16, July 11-13, 2016, Pacific Grove, CA, USA*



DOI: http://dx.doi.org/10.1145/2935764.2935773

this induces a game amongst the jobs. The objective in the coordination mechanisms is to design scheduling policies which have small Price of Anarchy (PoA) [33]. PoA measures the degradation in the social cost (objective function) at an equilibrium (typically Nash Equilibrium or Correlated Equilibrium) compared to the offline optimum which can assign each job to any machine. Coordination mechanisms have been extensively studied in the context of scheduling for the makespan objectives [4, 11, 10, 31]. Recently, Cole et al. [17] studied the coordination mechanisms for minimizing the weighted completion objective, and showed that many scheduling policies such as Shortest Job First, Round Robin and their weighted extensions to lead to games with constant PoA when all release dates are same (In this case, a job's completion time is equal to its flow time).

While coordination games provide valuable insight on the effect of selfish jobs, the assumption that jobs reach a NE is not very realistic in many settings, especially for scheduling jobs that arrive online. In these applications, it is unlikely that an arriving job will have the complete knowledge of other jobs that will arrive in future, hence studying the cost at a Nash Equilibrium may not be the right solution concept. Also in many cases, jobs make one-time decision on which machine they are assigned to (as in the DNS example), and are not reassigned.

Unfortunately, there has been very little work on designing efficient scheduling algorithms for greedy jobs in online setting. Now that resources are growing both in complexity and quantity, we believe that developing scheduling algorithms for greedy jobs for various objectives is of increasing importance. In this paper we take the initiative in this direction. Following our discussion, we assume that upon the arrival each job *selfishly and irrevocably selects a machine which minimizes its own flow time looking at only the sets of jobs assigned to each machine*. We also require that each machine can run only a *strongly local* scheduling policy that makes its scheduling decisions based only on the set of jobs assigned to it (independent of the processing lengths of jobs on other machines). Local scheduling policies are more desirable since they can save communication cost between machines and are easier to implement. We will consider unrelated machines, and will refer to this setting as scheduling of greedy jobs on unrelated machines. In this paper we would like to explore the following concrete questions:

*Are there strongly local online scheduling policies for the unrelated machine setting that are constant competitive for total weighted flow objective? Can we design scheduling algorithms that align the best moves for the overall system with that of individual jobs?*

## 1.1 Our Results and Contributions

Our main result is that Smoothed Latest Arrival Processor Sharing (SLAPS), and its weighted version, recently introduced by Im et al [27, 28], answers the above questions in the affirmative in the resource augmentation model. In resource augmentation model, the online algorithm is allowed to run each machine $(1+\epsilon)$ times faster and is compared to an optimal offline scheduler that runs at a unit speed [32]. Without resource augmentation, the competitive ratio of any online algorithm can be unbounded [23]. An online algorithm is said to be *scalable* if it is $O(poly(1/\epsilon))$-competitive for all $\epsilon > 0$, and finding a scalable algorithm is in general considered to be the ultimate goal when there is a strong lower bound on competitive ratio without speed augmentation.

Our main result is the following.

- For the problem of minimizing the weighted flow time of greedy jobs on unrelated machines, we show that Weighted SLAPS is $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive for any $0 < \epsilon \leq 1$ (Theorem 3.1).

We prove several crucial properties of SLAPS that leads to the desired result. We believe that these properties may be of independent interest in analyzing SLAPS for other scheduling problems.

Recently, another important research direction in scheduling theory has been designing energy efficient algorithms. Dynamic speed scaling is a widely studied setting where machines can be run faster by consuming more energy. In this setting, one of the most widely studied models is where a machine consumes energy $s^\alpha$ when run at a speed of $s$. The goal of an energy efficient scheduler is to minimize total weighted flow time plus total energy consumed.

Our next result is an energy efficient algorithm in our setting.

- For the problem of minimizing the weighted flow time plus energy of greedy jobs on unrelated machines, we give a strongly local scheduling policy that is $O(\alpha^2)$-competitive (Theorem 4.1).

To the best of our knowledge, these are the first positive results for scheduling greedy jobs online in the multiple machines setting (even in the simplest identical machines case) for any scheduling objectives except minimizing the maximum flow time when all jobs arrive at time 0 (makespan). We note that when the scheduler can assign each job to any machine, the best known results for each of the above problems are a $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive algorithm and a $O(\alpha/\log\alpha)$-competitive algorithm, respectively [2, 35, 19]. Perhaps, it is surprising that one can obtain results that nearly match the best known results even when jobs behave selfishly.

## 1.2 Limitations of Previous Approaches

The first scalable algorithm for minimizing weighted flow time on unrelated machine was given by Chadha et al. in a breakthrough result [13]. The algorithm in [13] had two properties which are highly desirable in multiple machines scheduling – *immediate dispatch* and *non-migration*. That is, the scheduler immediately dispatches an incoming job to a machine, and the job never migrates to other machines. Furthermore, their algorithm consisted of two surprisingly simple components: each machine runs the Highest Density First (the density of a job is defined as the ratio of its weight to size), and the dispatch rule is to assign each incoming job to the machine that increases the objective the least. Later, Anand et al. [2] improved this result via a novel dual fitting argument to obtain a slightly better competitive ratio, and extended their result to dynamic speed scaling setting.

The scheduling algorithms Highest Density First (HDF) or Highest Residual Density First (HRDF) that were used to get a (centralized) scalable algorithm by [13, 2], however, result in a competitive ratio as high as $\Omega(n)$ even when given any constant speed augmentation in our setting (We defer the proof to the full version.) The main reason for this catastrophic degradation in the performance is because the best move for individual jobs can be very different from the best move for the overall system. In particular, the greedy dispatch rule – assign incoming job to the machine that increase the objective by the least– is very crucial and a common ingredient in the works of [13, 26, 2]. Hence, it would be natural to ask the question: Is the greedy dispatch rule robust? How critical is it to minimize the increase in the objective when assigning jobs to machines?

Indeed, one can show that if the dispatch rule repeatedly makes a non-optimal decision in dispatching jobs, even if it is a constant factor off from the optimal decision, the resulting schedule could be

far from the optimum without substantial speed augmentation. This is the case no matter what algorithm each machines uses. Thus, it is imperative that any algorithm in our setting has to unify the best moves for the overall system and for individual jobs.

Then probably the first algorithm one would try will be Round Robin (RR) – for a while let us assume that jobs are unweighted to simplify our discussion. Intuitively, RR processes all jobs at the same rate, so the overall system performance will be less susceptible to jobs selfish behavior. This intuition turns out to be true, and one can show that RR achieves a constant competitive ratio for average flow time when given $(2+\epsilon)$-speed. However, RR is known to be not scalable [32]. Similarly, it is easy to show a lower bound of $\Omega(n)$ on Shortest Elapsed Time First (SETF) even with constant speed augmentation. The next natural candidate algorithm will be then Latest Arrival Processor Sharing (LAPS) which was first introduced by Edmonds and Pruhs in a beautiful result and was shown to be scalable for the average flow time objective in a fairly general scheduling setting [21]. The algorithm LAPS generalizes RR, and round-robins among a fraction of the most recently arriving jobs; the fraction is given as a parameter. Although LAPS was developed for scheduling for jobs of different parallelizability [21], it has been proven to be very useful in other scheduling contexts, in particular broadcast scheduling [7, 20]. However, when it comes to multiple (heterogeneous) machines scheduling, LAPS has a serious weakness that the increase of the objective due to a new job's arrival cannot be easily bounded. In a nutshell, it is because LAPS processes only jobs within a 'boundary' and this makes it very hard to measure the increase. Perhaps this is the reason why there is no work that used LAPS in non-identical machines settings [1].

SLAPS **and its properties.** We use SLAPS (and its weighted version), recently introduced by Im et al [27, 28], to overcome this issue. What SLAPS distinguishes itself from LAPS the most is its *proportional invariance*. The proportional invariance property states that the relative speed of two jobs does not change upon the arrival of new jobs into the machine. This simple property turns out to be very important in the analysis of SLAPS. In particular, using the proportional invariance property we establish that the flow time of a newly arriving job *closely* approximates the increase in the objective. We call this crucial property of SLAPS as *bounded externality*. Hence, when machines run SLAPS, individual job's greedy behavior leads to (approximately) globally good behavior. This is our key idea in obtaining scalable algorithm when jobs are greedy. This also answers the question (iii).

SLAPS seems to inherit many important properties of LAPS which has been very useful for various scheduling problems, for example see [7, 20, 14, 22]. One of those properties is that the scheduling decision is made only based on the *ordering of the arrival time* of currently existing jobs. This is a very unique property that LAPS has (and its special case, RR). In some cases, this counter intuitive scheduling decision makes the analysis a lot easier compared to other algorithms that use time-varying feature such as remaining processing times. SLAPS also makes its scheduling decisions only using the relative arrival ordering of jobs, and it seems that it can be used in all the places where LAPS was used to obtain the same results.

Finally, in terms of the service provided to individual jobs, we think SLAPS is more closer to Round Robin, which can be thought as a fair scheduling algorithm, than any other scheduling algorithm. One intuitive way of defining fairness is to let a job wait for at most a period of time in proportion to its size. The idea is to distinguish between jobs asking for different amount of resources, and this was the key intuition behind the total stretch objective which is the sum of each job's flow time divided by its size [9, 6]. In SLAPS, when a new job arrives, each existing job gets delayed by an amount up to a constant times the job's size, unlike LAPS (See Section 2). For these reasons, we believe that SLAPS is perhaps a more useful generalization of RR compared to LAPS.

[1]The only exception is [25] that used LAPS to show a scalable algorithm for average flow time on related machines. But the algorithm in [25] is migratory, which is very different from our setting.

## 1.3 Our technical contributions

Our analysis uses the elegant dual fitting framework introduced recently by Anand et al. [2]. So far our discussion was mainly on the difficulty arising due to the gap between the best moves for individual jobs and for entire system performance. Another challenge in obtaining a scalable algorithm in our setting is that we have to *directly argue with integral flow time* of jobs. Fractional flow time is a very useful relaxation of flow time and simplifies the analysis substantially, especially when potential functions are used [13, 30]. Another view of flow time of a job is to pay a unit cost/penalty for each unit of time for which the job waits. In fractional flow time, the penalty is only for the remaining portion of the job size (fractional weighted of a job is the remaining portion multiplied the job's weight). Roughly speaking, fractional flow time allows us to view each job as consisting of infinitesimal sized pieces, and hence any work on the job completes some pieces of the job. In general, fractional flow time is easier to handle. Hence it is now a standard approach to obtain a scheduling algorithm first for fractional flow time and convert it to integral flow using additional round of speed augmentation, for example [13, 30].

One crucial property needed in the analysis in [2] for minimizing the weighted flow time + energy was the monotonicity property of the underlying algorithm we discussed. Since HRDF (weighted version of Shortest Remaining Processing Time First) is not monotone, for the analysis of the total weighted flow time plus energy objective, [2] used Highest Density First (HDF) in place of HRDF. Unfortunately, HDF is the algorithm inspired by the fractional flow time view, and hence the analysis was done first on fractional flow time. This type of analysis cannot be used for our problem since the quantity each individual job seeks to minimize is its integral flow, not fractional flow. Hence throughout the analysis we need to argue only with the integral flow time. A consequence of this restriction is that, we can not use multiple rounds of speed augmentation typically seen in the analysis of many scheduling algorithms for flow time.

## 1.4 Other Related Work

Minimizing the average weighted flow time of jobs on unrelated machines has been extensively studied in the offline setting as well. The problem is known to be NP-hard to approximate within a factor of $\Omega(\log P)$ even when all machines are identical, i.e. $p_{ij}$ is the same for all $i$ [24]. Here $P$ is the ratio of the maximum to the minimum job size. The current best result is by Bansal and Kulkarni [8] that achieves an approximation factor of $O(\log n \log P)$.

In the online setting, Avrahami and Azar [3] were first to consider immediate-dispatch non-migratory schedules for minimizing the flow time in the identical machine environment. Chekuri et al. [15] gave the first scalable algorithm for this setting. [15, 13] show that one can obtain a scalable algorithm by assigning jobs to identical machines at random. For the unrelated machine setting, the first scalable algorithm for minimizing weighted flow time on unrelated machine was given by Chadha et al. in a breakthrough result [13]. Extensions to the $l_k$-norms of the flow time on unrelated machines were considered in [29, 2]. Later, Anand et al. [2] improved these

results via a novel dual fitting argument to obtain a slightly better competitive ratio.

Unfortunately, non-migratory algorithms do not lead any bounded competitive ratio for the objective of minimizing flow-time on unrelated machines in the non-clairvoyant setting [25]. On the other hand, if an algorithm is allowed to change the assignment of jobs over time, then scalable algorithms were obtained by Im et al in [27, 28]. In the setting we consider, however, migrating jobs over time is not allowed.

The flow-time objectives have also been considered when machines can change their speed by consuming more energy. The most common objective studied in this line of work is to minimize total weighted flow time plus total energy consumed. This objective offers a natural trade-off between weighted flow time and energy consumed [1]. Namely, one can improve the overall system performance by consuming more energy, and vice versa. For this objective, in the unrelated machines setting, Anand et al. [2] gave an $O(\alpha^2)$-competitive algorithm when each machine consumes energy $s^\alpha$ when it runs in speed $s$. Here the speed of the machine is set to the total (fractional) [2] weight of the jobs remaining at that time following the single machine scheduling result by Bansal et al. [5]. Intuitively, this is a natural way to balance two competing objectives, weighted flow time and energy since at each time the remaining jobs incur a total cost equal to their weight.

Much of the work on selfish scheduling of jobs has been considered in the setting of coordination games first introduced in a seminal paper [16]. In this setting, each machine declares a strongly local scheduling policy. The strategy of each job consists of choosing the machine which minimizes its own completion time. Each job knows the processing lengths of other jobs and hence this induces a game amongst the jobs. The objective in the coordination mechanisms is to design scheduling policies which have small Price of Anarchy (PoA) [33]. PoA measures the degradation in the social cost(objective function) at an equilibrium (typically Nash Equilibrium or Correlated Equilibrium) compared to the offline optimum which can assign each job to any machine. Coordination mechanisms have been extensively studied in the context of scheduling for the makespan objectives [4, 11, 10, 31]. Recently, Cole et al. [17] studied the coordination mechanisms for minimizing the weighted completion objective and showed that many scheduling policies such as Shortest Job First, Round Robin and their weighted extensions to lead to games with constant PoA when all release dates are same (In this case, a job's completion time is equal to its flow time).

However, our proposed model differs from the coordination mechanism model in many significant ways. In the coordination mechanisms, each job has the complete information of the system and knows the size and release dates of other jobs. Furthermore, it is a simultaneous move game. That is, jobs move around until they reach an equilibrium (which may take exponential steps and but is assumed to happen instantaneously). On the other hand, in our model jobs arrive online and each arriving job only knows about the set of jobs already present in the system. The jobs choose a machine looking at only the current state of the system. Hence the resulting configuration of jobs on machines need not be a Nash Equilibrium. We believe that our setting is a more realistic model of real world applications.

[2]The difference between fractional and integral weights will be discussed at the end of Section 1.1.

# 2. ALGORITHM SMOOTH LATEST ARRIVAL PROCESSOR SHARING (SLAPS) AND ITS PROPERTIES

To make our presentation more readable, we first describe the unweighted version of the algorithm (SLAPS) and discuss how it can be extended to weighted jobs in Section 2.2. See [27, 28] for more details. For the total flow time objective, the algorithm SLAPS is exactly the algorithm that will run on all machines. Also for the objective of total flow time plus energy, this algorithm will be used with appropriate dynamic speed scaling. The algorithm SLAPS is non-clairvoyant and local in the sense that if it is run on a machine, its scheduling decision relies only on the jobs assigned to the machine. Hence we will not need to refer to a specific machine to describe SLAPS. Our algorithm SLAPS is parameterized by a non-negative integer $k$, and we may use $\text{SLAPS}_k$ to highlight the parameter it uses. Let $A(t)$ denote the set of alive jobs at time $t$. Order the jobs in $A(t)$ in increasing order of their arrival time. Let $\pi_{j,t}$ denote job $j$'s rank at time $t$ in this order. Here the earliest arriving job in $A(t)$ has rank 1 and the latest arriving job in $A(t)$ has rank $|A(t)|$. We assume without loss of generality that all jobs have distinct arrival times by breaking ties arbitrarily but consistently. Let $s \geq 1$ denote the speed that the algorithm is given. We distribute the total processing power available among the jobs in $A(t)$ by giving each job $j$ the following speed:

$$h_j(t) := s \cdot \frac{\pi_{j,t}^k}{1^k + 2^k + ... + (|A(t)|)^k}, \tag{1}$$

We will refer $h_j(t)$ as $j$'s share or speed at time $t$. Observe that the total share of all jobs in $A(t)$ is exactly $s$ (when $A(t) \neq \emptyset$). This completes the description of the algorithm that each machine runs.

It is worth noting that $\text{SLAPS}_0$ is exactly ROUND ROBIN (RR).

## 2.1 Properties of SLAPS

We discuss several interesting properties of the algorithm SLAPS which will be useful throughout the analysis. We first summarize these useful properties in words. The formal description can be found below and in Lemma 3.2.

- Proportional Invariance: The *relative* speed that each job gets does not change when more jobs arrive.
- Bounded Externality: The total delay seen by all the jobs already present in the system (increase in the total flow time) upon arrival of a new job is bounded by the flow time of the newly arriving job, assuming it is the last job arriving into the system. See Lemma 3.2.
- Monotonicity: Consider two schedules $\sigma$ and $\sigma'$ where the set of jobs in $\sigma$ is a subset of jobs in $\sigma'$. Then, at any point in time, if a job $j$ is alive in both schedules, $j$ is processed only slower in $\sigma'$. Further, at any time, $\sigma'$ has no less alive jobs than $\sigma$.

In order to formally describe the above properties, we need to define some notation. Consider a set of jobs $S$ say at time $t$. Now consider another set $S'$ of alive jobs which arrive later than all jobs in $S$. Jobs in $S \cup S'$ are possibly partially processed. Let $h_j^S(t)$ denote job $j$'s share of speed when the algorithm has only the jobs in $S$ in its current queue $A(t)$. Similarly, define $h_j^{S \cup S'}(t')$ for the case where the algorithm has only the jobs in $S \cup S'$ in its current queue $A(t')$ at some time $t'$. Here we are implicitly assuming that the algorithm's instantaneous scheduling decision is oblivious to the remaining job sizes.

*Proportional Invariance.* For any two jobs $j, j' \in S$ and any

time instants $t$ and $t'$ it is the case that $h_j^S(t)/h_{j'}^S(t) = h_j^{S\cup S'}(t')/h_{j'}^{S\cup S'}(t')$.

The property follows from the definition of SLAPS since each newly arriving job slows down the jobs in the set $S$ by exactly the same factor. An interesting consequence of this property is that there exists an one-to-one mapping of time instants between two schedules such that remaining sizes of jobs in the set $S$ are exactly the same.

Now fix time any time $t$, and consider two sets $S$ and $S'$ defined as above. Let $\sigma$ denote the schedule induced when the algorithm works on jobs in $S$ from time $t$. Likewise, $\sigma'$ denote the schedule induced when the algorithm works on jobs in $S$ from time $t$ and jobs in $S'$ arrive later than $t$. Let $h_j^{(\cdot)}$ denote the job $j$'s share of speed in the schedule specified in the superscript. Let $n_j^{(\cdot)}$ denote the number (total weight) of alive jobs in the schedule specified in the superscript. Note that we added $\sigma$ to $A(\tau)$ as superscript to emphasize that $A^\sigma(\tau)$ denotes the set of alive jobs at time $\tau$ in schedule $\sigma$.

*Monotonicity.* For any time $\tau \geq t$, $n_\tau^\sigma \leq n_\tau^{\sigma'}$. Also for any job $j \in A^\sigma(\tau)$, $h_j^\sigma(\tau) \geq h_j^{\sigma'}(\tau)$.

The monotonicity property follows from the definition of SLAPS since the rank of existing jobs does not change due to new jobs arrival but some portion of the processing power is given to new jobs. We stated these properties formally since there is a similar property that does not hold: when some jobs are completed, every job's speed can only increase. Although this seemingly-true property holds when $k = 0$ (RR), it is not the case for general $k$. We prove the bounded externality property in Lemma 3.2.

## 2.2 Weighted Version of SLAPS

In this section we extend the algorithm SLAPS to work for weighted jobs. The main intuition is to think of a job $j$ of weight $w_j$ as $w_j$ copies of unweighted job with the same arrival time. Then the share of job $j$ is defined as the total share of job $j$'s copies; we assume wlog that $w_j$ is an integer. More formally, define $\bar{w}_j := \sum_{j' \in A(t): r_{j'} \leq r_j} w_{j'}$, and $f(x) := 1^k + 2^k + ... + x^k$. Let $j'$ be the job that arrives the latest before job $j$. Then job $j$'s share is defined as $h_j(t) := s \cdot \frac{f(\bar{w}_j) - f(\bar{w}_{j'})}{f(\bar{w}_j)}$. Note that when jobs have unit weights, this definition is exactly the same as the one given for unweighted jobs.

We state a useful inequality that is used frequently in the analysis.

**Proposition 2.1** *Consider any integer $k \geq 0$, and $n \geq 1$. Then it follows*

$$\frac{k+1}{n+k+1} \leq \frac{n^k}{1^k + 2^k + ... + n^k} \leq \frac{k+1}{n}$$

PROOF. To show the first inequality we use $1^k + 2^k + ... + (n-1)^k \leq \int_{x=0}^{n} x^k dx = n^{k+1}/(k+1)$. For the second, use $1^k + 2^k + ... + n^k \geq \int_{x=0}^{n} x^k dx$. □

# 3. TOTAL (AVERAGE) FLOW TIME

In this section we show that SLAPS induces a scalable algorithm (schedule) when each job chooses to go to the machine that minimizes its flow time. In order to present our analysis more transparently, we proceed our analysis assuming all jobs are unweighted, i.e. $w_j = 1$, and the analysis for weighted jobs is presented is similar and we omit the proof in this version. We remind the reader that each job's decision is also myopic in the sense that its decision is made solely based on the current status of machines without considering future jobs. As mentioned before, our model is immediate-dispatch and non-migratory: every job chooses the machines immediately upon its arrival, and no job migrates to other machines once assigned to a machine. We prove the following theorem. For simplicity, $1/\epsilon$ is assumed to be an integer.

**Theorem 3.1** *Suppose that the online algorithm is given $1 + 3\epsilon$ speed for any $0 < \epsilon \leq 1$. If all machines run SLAPS$_k$ where $k = 1/\epsilon$, the resulting schedule is $O(1/\epsilon^2)$-competitive for the objective of average flow time of all jobs.*

We begin our analysis by proving the bounded externality property of SLAPS. We will show that the increase in total flow time due to job $j$'s arrival and the flow time of job $j$ (assuming no more jobs arrive) are comparable. Let $F_{ij}(r_j)$ denote the flow time of job $j$ when it chooses machine $i$ at time $r_j$ assuming that no more jobs arrive. Let $\Delta_{ij}$ denote the increase in the total flow time. More precisely, $\Delta_{ij}$ is the increase of flow time of all jobs that are in the queue of machine $i$ upon the arrival of job $j$ plus job $j$'s flow time $F_{ij}(r_j)$. We remark once more that $\Delta_{ij}$ and $F_{ij}(r_j)$ are defined assuming that $j$ is the last job on machine $i$.

**Lemma 3.2** $\Delta_{ij} \leq (k+2)F_{ij}(r_j)$.

PROOF. For notational simplicity, consider any fixed machine $i$, and re-index jobs so that every job $j'$ in $A_i(t)$ has rank $j'$. Let $1, 2, ..., n$ be the jobs in $A_i(t)$. Throughout the proof we drop the machine $i$ from the notation if there is no confusion from the context. Let $n+1$ be the newly arriving job. The proportional invariance property of SLAPS states that all jobs are slowed down by exactly the same factor upon the arrival of a new job. This allows us to define a mapping between the schedules before and after the new job $n+1$ arrives.

Let $\sigma$ denote the schedule of jobs $1, 2, ..., n$ on machine $i$ defined from time $r_{n+1}$ assuming that no more jobs arrive, and $\sigma'$ the schedule of jobs $1, 2, ..., n, n+1$. The schedule $\sigma'$ is also defined from time $r_{n+1}$ assuming that no more jobs arrive. Let $C'$ denote the completion time of job $n+1$ in $\sigma'$. We can define a one-to-one map between the time steps in two schedules $\sigma$ and $\sigma'$ with the same remaining processing times of jobs $1, 2, ..., n$. Formally, let $(q_1^\sigma(t), q_2^\sigma(t), \ldots q_n^\sigma(t))$, denote the vector of remaining processing times of jobs in the schedule $\sigma$ and similarly define $(q_1^{\sigma'}(t), q_2^{\sigma'}(t), \ldots q_n^{\sigma'}(t))$. For every time instant $t$ in the schedule $\sigma$, we find the corresponding time instant $t'$ in $\sigma'$ such that, vector of remaining processing times of jobs are equal. Note that, this mapping is only possible due to proportional invariance property of SLAPS. Define the unique function $g(t) : [r_{n+1}, C] \rightarrow [r_{n+1}, C']$ that maps time steps in $\sigma$ to $\sigma'$. Here $C$ denotes the time step in $\sigma$ corresponding to the time $C'$ in $\sigma'$. It is easy to see that $C' \geq C$ due to the monotonicity property of SLAPS. Let $g^{-1}(t)$ denote the time step in $\sigma$ corresponding to $t$ in $\sigma'$.

Consider any infinitesimal time interval $[t, t + \mathrm{d}t)$ in $[r_{n+1}, C']$ where no job is completed. Obviously, flow time of job $n+1$ is $F_{n+1}(r_{n+1}) := C' - r_{n+1}$. Now we look at how much the schedule $\sigma$ slows down during $[t, t + \mathrm{d}t)$ compared to $\sigma'$. That is, we will compare the length of two intervals, $[t, t + \mathrm{d}t)$ and $[g^{-1}(t), g^{-1}(t + \mathrm{d}t))$. The new job $n+1$ slows down other jobs processing, and each alive job in $1, 2, ..., n$ experiences a delay of $\mathrm{d}t - (g^{-1}(t + \mathrm{d}t) - g^{-1}(t))$. Let $n_t$ be the number of alive jobs amongst $1, 2, ..., n$ in $\sigma$ during $[t, t + \mathrm{d}t)$. From the definition (1) we know that each job (except $n+1$) is processed in $\sigma'$ at a slower rate as it is in $\sigma$, and the exact slow down factor is

$\frac{1^k+2^k+\ldots+n_t^k}{1^k+2^k+\ldots+n_t^k+(n_t+1)^k}$. Hence the total increase of flow time during $[t, t+\mathrm{dt})$ in $\sigma'$ from $\sigma$ is at most,

$$\begin{aligned} & \frac{(n_t+1)^k}{1^k+2^k+\ldots+(n_t+1)^k} \cdot n_t \cdot \mathrm{dt} + \mathrm{dt} \qquad (2) \\ \leq \quad & (k+1) \cdot \frac{n_t}{n_t+1} \cdot \mathrm{dt} + \mathrm{dt} \leq (k+2)\mathrm{dt} \end{aligned}$$

The quantity consists of two parts: the first part is due to the increase of flow of "old" jobs and the second $(\mathrm{dt})$ is part of the flow time of job $n+1$. The first inequality is due to Proposition 2.1. We charge the first part to the second, which completes the proof. □

## 3.1 Dual Fitting

To prove Theorem 3.1, we use the dual fitting framework in [2]. We first write a linear programming relaxation of the problem $\mathsf{LP}_{\mathsf{primal}}$ described below which was first given by [2]. It has a variable $x_{ijt}$ for each machine $i \in [m]$, each job $j \in [n]$ and each unift time slot $t \geq r_j$. If the machine $i$ processes the job $j$ during the whole time slot $t$, then this variable is set to 1. The first constraint says that every job has to be completely processed. The second constraint says that a machine cannot process more than one unit of jobs during any time slot. Note that the LP allows a job to be processed simultaneously across different machines.

$$\text{Min} \quad \sum_j \sum_i \sum_{t \geq r_j} w_j \cdot \left( \frac{t-r_j}{p_{ij}} + \frac{1}{2} \right) \cdot x_{ijt} \qquad (\mathsf{LP}_{\mathsf{primal}})$$

$$\begin{aligned} \sum_i \sum_{t \geq r_j} \frac{x_{ijt}}{p_{ij}} \quad & \geq \quad 1 \qquad && \forall j \\ \sum_{j\,:\,t \geq r_j} x_{ijt} \quad & \leq \quad 1 \qquad && \forall i, t \\ x_{ijt} \quad & \geq \quad 0 \qquad && \forall i, j, t \,:\, t \geq r_j \end{aligned}$$

We briefly discuss how the objective function lower bounds the total weighted flow time of a feasible integral schedule where each job is processed on a single machine. For more details see [2]. Say job $j$ is assigned to machine $i$. In the objective function, one can easily see the term $\sum_{t \geq r_j} w_j \cdot (\frac{t-r_j}{p_{ij}}) \cdot x_{ijt}$ is at most job $j$'s weighted flow time due to the first constraint. In fact, the quantity is the fractional weighted flow time of the job $j$, which is well known to be at most its integral weighted flow time, minus $w_j p_{ij}/2$. The remaining term $\sum_{t \geq r_j} w_j \cdot (1/2) \cdot x_{ijt}$ equals $w_j p_{ij}/2$. Hence, $\mathsf{LP}_{\mathsf{primal}}$ is a valid relaxation. We note that the '1/2' in the objective is crucial since otherwise the LP can cheat by processing a job simultaneously on a lot of machines, thereby paying very little for the job.

Now we write the dual of the above linear program.

$$\text{Max} \quad \sum_j y_j - \sum_i \sum_t z_{it} \qquad (\mathsf{LP}_{\mathsf{dual}})$$

$$\begin{aligned} \frac{y_j}{p_{ij}} - z_{it} \quad & \leq \quad w_j(\frac{t-r_j}{p_{ij}} + \frac{1}{2}) \qquad && \forall i, j, t \,:\, t \geq r_{ij} \\ y_j \quad & \geq \quad 0 \qquad && \forall j \\ z_{it} \quad & \geq \quad 0 \qquad && \forall i, t \end{aligned}$$

**Setting dual variables:** We proceed our analysis by defining the dual variables as follows. As mentioned before, we will assume that all jobs have weight 1, and extend to arbitrary weights later. Say job $j$ chose to go to machine $i$. We set $y_j$ to be $1/8$ times the flow time of job $j$ at the time of its arrival assuming that no more jobs arrive, i.e. $y_j = F_{ij}(r_j)/8$. We set $z_{it}$ to be $1/8(k+3)$ times the total number of alive jobs at the time $t$ on machine $i$, i.e. $z_{it} = \frac{n_{it}}{8(k+3)}$ where $n_{it}$ denotes the total number of jobs that are alive at time $t$ on machine $i$.

**Bounding the dual objective:** By Lemma 3.2 we know that $\sum_j y_j$ is $1/8(k+2)$ times the total flow time. Also due to the well known fact that the sum of number of alive jobs over all times equals the total flow time, we conclude that $\sum_i \sum_t z_{it} = F^A/8(k+3)$, where $F^A$ is the total flow time of all jobs in our algorithm. Therefore, the dual objective is $\Omega(1/k^2) = O(\epsilon^2)$ times the total flow time, which establishes the competitive ratio claimed in Theorem 3.1, if the dual variables satisfy all the constraints.

It now remains to show that this definition of dual variables satisfies the dual constraints. Note that we need to satisfy the dual constraint for any pair of job $j$ and machine $i$. We consider two cases.

*Case 1: Job $j$ chooses machine $i$.* We first show the constraint is satisfied assuming that job $j$ is the job arriving the last, then later will lift this assumption using the monotonicity property of SLAPS.

For any time $t \in [r_j, r_j + F_{ij}(r_j)]$, $F_{ij}(r_j) - (t - r_j)$ is the remaining flow time of job $j$. We observe that the speed with which the job $j$ is processed can only increase as time passes. This follows from the proposition stated below and noting that $j$ is the highest ranked job.

**Proposition 3.3** *$n^k/(1^k + 2^k + \ldots + n^k)$ is decreasing in $n$.*

PROOF. We view $n^k/(1^k+2^k+\ldots+n^k)$ as $\frac{\sum_{l \in [n]} l^k - (l-1)^k}{\sum_{l \in [n]} l^k}$. We use the following: given positive $a_i, b_i$ such that $\frac{a_1}{b_1} \geq \frac{a_2}{b_2} \geq \ldots \geq \frac{a_l}{b_l}$, it follows that $\frac{\sum_{l' \in [l-1]} a_i}{\sum_{l' \in [l-1]} b_i} \geq \frac{\sum_{l' \in [l]} a_i}{\sum_{l' \in [l]} b_i}$. Hence it suffices to show $\frac{l^k-(l-1)^k}{l^k} = 1 - (1-1/l)^k$ is decreasing in $l$, which is indeed the case. □

Hence we deduce that $F_{ij}(r_j) - (t - r_j) \leq p_{ij}/h_{ij}(t)$

where $h_{ij}(t)$ is the speed job $j$ gets at time $t$. Also we know by Proposition 2.1 that for all $t \geq r_j$, $h_{ij}(t) \geq s \cdot \frac{n_{it}^k}{\sum_{l=1}^{n_{it}} l^k} \geq s \cdot \frac{k+1}{n_{it}+k+1}$.

Hence we get,

$$\begin{aligned} & \frac{8y_j - (t-r_j)}{p_{ij}} = \frac{F_{ij}(r_j) - (t-r_j)}{p_{ij}} \\ \leq \quad & \frac{1}{h_{ij}(t)} \leq \frac{1}{s} \cdot (1 + \frac{n_{it}}{k+1}) \\ = \quad & \frac{1}{s}(1 + \frac{k+3}{k+1} \cdot 8z_{it}) \leq 1 + 8z_{it} \end{aligned}$$

Note that we used the facts that $s = 1 + 3\epsilon$ and $k = 1/\epsilon$. Arranging terms shows the dual constraints are satisfied.

We now explain why the assumption that no more jobs arrive after $j$ is w.l.o.g. Note that $y_j$ relies only on the jobs arriving no later than job $j$, hence has no dependency on jobs arriving after job $j$. Finally, the monotonicity property of SLAPS states that for any *fixed* time $t \geq r_j$, when more jobs arrive into machine $i$, the number of alive jobs at time $t$ on machine $i$, $n_{it}$ can only increase. Hence, if more jobs arrive, it only gives more slack to the dual constraint.

*Case 2: Job $j$ chooses a machine other than $i$.* Say that job $j$ chose to go to machine $i' \neq i$. As in the previous case, we can without

loss of generality assume that no more jobs arrive. Throughout the analysis we will be concerned with two schedules $\sigma$ and $\sigma'$. The first schedule $\sigma$ is the real schedule which describes how jobs are processed on machine $i$ since time $r_j$. The second schedule is a fictitious schedule on machine $i$ obtained if job $j$ were assigned to machine $i$, rather $i'$. The proof from the case one implies that dual constraints are satisfied if we were to consider the schedule $\sigma'$. However, we need to work with schedule $\sigma$. One cause of concern is that the number of alive jobs in $\sigma$ may be much less than the number of alive jobs in $\sigma'$. However, by carefully mapping the two schedules we show that this is not the case.

As we did in the proof of Lemma 3.2 we can define one-to-one mapping between $\sigma$ and $\sigma'$ so that the two schedules have exactly the same remaining processing time for all jobs except $j$ which are assigned to machine $i$. We will use $g(t)$ to denote the unique time step in $\sigma'$ corresponding to $t$ in $\sigma$. We let $C'$ denote job $j$'s completion time in $\sigma'$. Let $C = g^{-1}(C')$. Consider any time $t \in [r_j, C]$. Consider an infinitesimal time interval $[t, t+\mathrm{dt})$ in $[r_j, C]$. We note that the length of the interval in $\sigma'$ corresponding to $[t, t+\mathrm{dt})$ in $\sigma$ is at most

$$\mathrm{dt} + \frac{(n_{it}+1)^k}{1^k + 2^k + ... + n_{it}^k}\mathrm{dt} \leq \mathrm{dt} + \frac{k+1}{n_t}(1+\frac{1}{n_{it}})^k \mathrm{dt} \quad (3)$$

where $n_{it}$ is the number of alive jobs at time $t$ on machine $i$ in schedule $\sigma$. This is because every alive job in $\sigma$ is processed slower than in $\sigma'$ by a factor of $(\sum_{l=1}^{n_{it}} l^k)/(\sum_{l=1}^{n_{it}+1} l^k)$. The inequality is due to Proposition 2.1.

Note that job $j$ is alive at time $g(t)$ in schedule $\sigma'$, and that is the only difference from the schedule $\sigma$ at time $t$. We now consider two cases. If $n_t \geq k+1$, (3) is at most $4\mathrm{dt}$. Now suppose that $n_{it} < k+1$. Since job $j$ has the highest rank, the fraction the job $j$ gets from the total speed $s$ in $\sigma'$ is at least $(n_{it}+1)^k/(1^k + 2^k + ... + (n_{it}+1)^k) \geq \frac{k+1}{k+1+n_{it}+1} \geq 1/2$ by Proposition 2.1. Hence the total length of such time steps is at most $2p_{ij}$. Hence we have shown that for any $t \in [r_j, C)$,

$$g(t) - t \leq 4(t - r_j) + 2p_{ij} \quad (4)$$

We are now ready to complete our analysis.

$$\begin{aligned}
& \frac{8y_j - (t-r_j)}{p_{ij}} = \frac{F_{i'j}(r_j) - (t-r_j)}{p_{ij}} \\
\leq\ & \frac{F_{ij}(r_j) - (t-r_j)}{p_{ij}} \quad \text{[Since job } j \text{ chose } i' \text{ over } i\text{]} \\
=\ & \frac{F_{ij}(r_j) - (g(t)-r_j) + (g(t)-t)}{p_{ij}} \\
\leq\ & \frac{1}{s}(1 + \frac{n_{it}+1}{k+1}) + \frac{(g(t)-t)}{p_{ij}} \quad (5) \\
\leq\ & 2 + 8z_{it} + \frac{4(t-r_j)}{p_{ij}} + 2 \quad (6)
\end{aligned}$$

The inequality in (5) follows from the same argument in the previous case since $\sigma'$ is constructed by assigning job $j$ to $i$. Here note that the '1' is added to $n_{it}$ to count job $j$. The inequality (6) follows from (4). By rearranging terms, we complete the analysis.

# 4. EXTENSION TO TOTAL FLOW + ENERGY

In this section we extend the online greedy scheduling problem to the speed scaling setting. In this setting, each machine's speed can be adjusted dynamically, and consumes power $s^\alpha$ when it runs with speed $s$ where $\alpha > 1$ is a constant which is uniform to all machines. The goal is to design local scheduling policy and a dynamic speed scaling algorithm to minimize the total flow time plus energy consumption. We aim to prove the following theorem.

**Theorem 4.1** *There exists an online algorithm such that when each machine runs the algorithm, and each job goes to the machine that minimizes its flow time without considering future jobs, the resulting schedule is $O(\alpha^2)$-competitive for the problem of minimizing the total flow time plus energy in the unrelated machines setting.*

## 4.1 Algorithm

As before, each machine will run SLAPS$_k$ with $k = \lceil \alpha \rceil$. In the analysis, for simplicity, we will assume that $\alpha$ is an integer greater than 1; this assumption can be easily removed, and the same analysis goes through. The speed of machine $i$ at time $t$ is set to $n_{it}^{1/\alpha}$. Here $n_{it}$ denotes the total number of alive jobs at time $t$ that are assigned to machine $i$. For the sake of analysis, we will assume that each machine gets more speed by factor $s = \max(\frac{k+3}{k}, \frac{(k+1)(k+2)}{k+3})$ if $\alpha < 4$ and $s = \frac{k+3}{k}$ otherwise, than it should get for consuming $n_{it}$. That is, each machine $i$ consumes power equal to $n_{it}$ and gets speed $s \cdot n_{it}^{1/\alpha}$. Note that the total power is equal to $s^\alpha$ times the total flow time. However, this is only $O(1)$ times the cost of flow time. Hence we will focus on bounding the total flow time, and factor in the energy cost at the end of analysis. Our algorithm for setting the speed at each time instant is same as the one used in [5, 23].

## 4.2 Analysis of Total Flow + Energy

In this section we give the proof of theorem 4.1. Taking similar steps as in the analysis of the average flow time objective, we will show that the increase in total flow time due to job $j$'s arrival and the flow time of job $j$ at the time of arrival are comparable. We use the same notation $F_{ij}(r_j)$ and $\Delta_{ij}$. We prove a similar bound as one in the Lemma 4.2 taking into consideration the effect of dynamic speed scaling.

**Lemma 4.2** $\Delta_{ij} \leq (k+2)F_{ij}(r_j)$.

PROOF. The proof is similar to that of Lemma 4.2, hence we will omit few details and highlight the differences. Recall that, $\sigma$ denotes the schedule of jobs $1, 2, ..., n$ on machine $i$ defined from time $r_{n+1}$ assuming that no more jobs arrive, and $\sigma'$ the schedule of jobs $1, 2, ..., n, n+1$. Note that, jobs are renamed to match their rank and $n+1$ is the latest arriving job. Let $g(t) : [r_{n+1}, C] \rightarrow [r_{n+1}, C']$ be the function which maps time steps in $\sigma$ to $\sigma'$ where remaining processing lengths of jobs are same. Unlike in the case of proof of Lemma 4.2, here two schedules run at different speeds. However, proportional invariance property still holds and it is easy to see that every job is slowed by exactly the same amount. Hence, $g(t)$ is well defined.

Consider any infinitesimal time interval $[t, t+\mathrm{dt})$ in $[r_{n+1}, C']$ where no job is completed. Clearly, flowtime of job $n+1$ is $F_{n+1}(r_{n+1}) := C - r_{n+1}$. Now we look at how much the schedule $\sigma'$ slows down during $[t, t+\mathrm{dt})$ compared to $\sigma$ during $[g^{-1}(t), g^{-1}(t+\mathrm{dt}))$. The new job $n+1$ slows down other jobs, and recall that delay seen by each job is $dt - (g^{-1}(t+\mathrm{dt}) - g^{-1}(t))$. Here we also need to consider the fact that machine $i$ runs at speed $n_{it}^{1/\alpha}$ in $\sigma$ and $(n_{it}+1)^{1/\alpha}$ in $\sigma'$. It is easy to see that the slow down factor is

$$\frac{1^k + 2^k + ... + n_{it}^k}{1^k + 2^k + ... + n_{it}^k + (n_{it}+1)^k} \cdot \frac{(n_{it}+1)^{1/\alpha}}{n_{it}^{1/\alpha}}$$

Here the second term $\frac{(n_{it}+1)^{1/\alpha}}{n_{it}^{1/\alpha}}$ is greater than 1, hence we can ignore it in lower bounding the slow down factor. Therefore, rest of the analysis remains exactly the same as in the proof of Lemma 4.2. This completes the proof.

□

## 4.3 Dual Fitting Analysis for Total Flow + Energy

We write a convex programming relaxation for the average flow plus energy objective. At a high level, this formulation is essentially the same as the one presented in [2] but is easier to work with and interpret. Due to this, we modestly simplify the analysis and make it exactly similar to analysis of the flow time objective. As noted earlier, since each job chooses the machine which minimizes its flow time upon its arrival, we no longer can afford to work with the fractional flow time of jobs unlike the previous works.

Consider the convex program $\mathsf{CP}_{\mathsf{primal}}$ described below. It has a variable $x_{ijt}$ for each machine $i \in M$, each job $j \in J$ and each unit time-slot $t \geq r_j$. If the machine $i$ processes the job $j$ during the whole time slot $t$, then this variable is set to 1. It has a variable $s_{it}$ for each machine $i$ and each time slot $t$. The variable $s_{it}$ indicates the speed used by machine $i$ at time $t$.

$$\text{Min} \quad \sum_j \sum_i \sum_{t \geq r_j} w_j \cdot x_{ijt} \cdot \left( \frac{t - r_j}{p_{ij}} + \frac{1}{2} \right) + \sum_i \sum_t s_{it}^{\alpha} \qquad (\mathsf{CP}_{\mathsf{primal}})$$

$$\begin{aligned} \sum_i \sum_{t \geq r_j} \frac{x_{ijt}}{p_{ij}} &\geq 1 && \forall j \\ \sum_{j\,:\,t \geq r_j} x_{ijt} &\leq s_{it} && \forall i, t \\ x_{ijt} &\geq 0 && \forall i, j, t \,:\, t \geq r_j \end{aligned}$$

The first constraint says that every job has to be completely processed. The second constraint says that a machine cannot process more than $s_{it}$ units of the jobs during any time slot. Note that the convex program allows a job to be processed simultaneously across different machines. The first term in the objective is a lower bound on the *total cost* of any feasible schedule was shown in [2]. We would like to point out that the first term in the objective does not lower bound the total flow time of jobs. The second term is obviously the total power consumed. Hence this convex program is at most $O(1)$ times the cost of optimal solution.

We derive the dual of the above convex program by following the template outlined in [18]. In the dual objective, the second part is the conjugate function of $\sum_i \sum_t s_{it}^{\alpha}$.

$$\text{Max} \quad \sum_j y_j - (1 - \frac{1}{\alpha}) \cdot \alpha^{\frac{1}{1-\alpha}} \sum_i \sum_t u_{it}^{\frac{\alpha}{\alpha-1}} \qquad (\mathsf{CP}_{\mathsf{dual}})$$

$$\begin{aligned} \frac{y_j}{p_{ij}} - z_{it} &\leq w_j(\frac{t - r_j}{p_{ij}} + \frac{1}{2}) && \forall i, j, t \,:\, t \geq r_j \\ z_{it} &\leq u_{it} \\ y_j &\geq 0 && \forall j \\ z_{it} &\geq 0 && \forall i, t \end{aligned}$$

**Setting the dual variables:** Our analysis will be based on dual fitting. Again, we first study unweighted jobs and extend it to weighted jobs. Say job $j$ chose to go to machine $i$. We set $y_j$ to be $1/8$ times the flow time of job $j$ at the time of its arrival assuming that no more jobs arrive, i.e. $y_j = F_{ij}(r_j)/40$. We set $z_{it}$ and $u_{it}$ to be $1/8(k+3)$ fraction of the number of jobs alive at the time $t$ on machine $i$, i.e. $z_{it} = u_{it} = \frac{n_{it}^{(\alpha-1)/\alpha}}{8(k+3)}$ where $n_{it}$ denotes the total number of jobs that are alive at time $t$ on machine $i$.

**Bounding the dual objective:** We first show that from our definition of dual variables the dual objective is $\Omega(1/\alpha^2)$ times the total flow time of our algorithm, denoted by $F^A$.

$$\begin{aligned} & \text{OPT} \\ \geq\ & \sum_j y_j - (1 - \frac{1}{\alpha}) \cdot \alpha^{\frac{1}{1-\alpha}} \cdot \sum_i \sum_t u_{it}^{\frac{\alpha}{\alpha-1}} \\ \geq\ & \sum_j \frac{1}{8} F_{ij}(r_j) - (1 - \frac{1}{\alpha}) \cdot \alpha^{\frac{1}{1-\alpha}} \cdot \frac{1}{8(k+3)^{\alpha/(\alpha-1)}} \sum_i \sum_t n_{it} \\ \geq\ & \frac{1}{8(k+2)} F^A - \frac{1}{8(k+3)} F^A \quad \text{[By Lemma 4.2]} \qquad (7) \\ =\ & \Omega(1/k^2) F^A = \Omega(1/\alpha^2) F^A \qquad (8) \end{aligned}$$

This establishes that the dual objective is at least $\Omega(1/\alpha^2)$ times the total flow time. Moreover, the energy cost of our schedule is at most $s^{\alpha}$ times the total flow time, which we noted is at most $O(1)$ times the cost of flow time. Therefore, if we prove that dual constraints are satisfied, it follows from the weak duality theorem for convex programs and $\epsilon = 1/\lceil \alpha \rceil$ that the competitive ratio of our algorithm is at most $O(\alpha^2)$.

It now remains to show that this definition of dual variables satisfies the dual constraints. Note that we need to satisfy the dual constraints for any pair of job $j$ and machine $i$. We consider two cases.

Case 1: *Job $j$ chooses machine $i$.* We first observe that given the following lemma, it suffices to show that the constraint is satisfied assuming that job $j$ is the last arriving job. This is because, our definition of $y_j$ does not change upon arrival of new jobs and as a consequence of the following lemma, for any *fixed* time, $z_{it}$ can only increase when more jobs arrive.

**Lemma 4.3** *Consider two schedules $\sigma$ and $\sigma'$ such that:*

- *In both schedules, the same set of jobs arrive before the job $j$ arrives.*
- *In $\sigma$, the job $j$ is the last arriving job while in $\sigma'$ some jobs arrive after job $j$ arrives.*

*Then at all times $t \geq r_j$, every job with arrival time at most $r_j$ has been processed more in $\sigma$ than in $\sigma'$.*

PROOF. By the proportional invariance property, we observe that all jobs arriving before $r_j$ are processed at the same relative rate in both schedules. Reindex the jobs such that any job $l$ has the rank $l$. Let total number of alive jobs be $n$. If a new job is added, the speed of job $l$ will change from, $l^k \cdot n^{1/\alpha}/(1^k + 2^k + ... + n^k)$ to $l^k \cdot (n+1)^{1/\alpha}/(1^k + 2^k + ... + (n+1)^k)$.

We will show that,

$$\frac{n^{1/\alpha}}{1^k + 2^k + ... + n^k} \geq \frac{(n+1)^{1/\alpha}}{1^k + 2^k + ... + (n+1)^k}$$

This will prove that any 'old' job's (jobs with release time at most $r_j$) speed can only decrease when new jobs arrive. Hence the lemma will follows. Toward this end, we use the following: given positive $a_i, b_i$ such that $\frac{a_1}{b_1} \geq \frac{a_2}{b_2} \geq ... \geq \frac{a_l}{b_l}$, it follows that

$\frac{\sum_{l' \in [l-1]} a_i}{\sum_{l' \in [l-1]} b_i} \geq \frac{\sum_{l' \in [l]} a_i}{\sum_{l' \in [l]} b_i}$, and observe that it suffices to show that

$$\frac{(n+1)^{1/\alpha} - n^{1/\alpha}}{(n+1)^k}$$

is decreasing. This can be easily verified by differentiating the function. □

Hence we focus on the case when $j$ is the last arriving job and show that all the dual constraints are satisfied.

**Proposition 4.4** *The function $n^{1/\alpha} \cdot \frac{k+1}{n+k+1}$ is non-decreasing in the interval $[1, k + 1/\alpha - 1)$ and it is non-increasing everywhere else for all positive $n$.*

PROOF. We omit the details as this can be easily verified by looking at the derivate of the function. □

Consider the speed at which job $j$ is processed at time $t$. Using Prooposition 2.1 we have,

$$h_{ij}(t) \geq s \cdot \frac{n_{it}^k}{\sum_{l=1}^{n_{it}} l^k} \cdot n_{it}^{1/\alpha} \geq s \cdot \frac{k+1}{n_{it}+k+1} \cdot n_{it}^{1/\alpha}$$

Let $h = s \cdot \min(\frac{k+1}{n_{it}+k+1} \cdot n_{it}^{1/\alpha}, \frac{k+1}{k+2})$. From Proposition 4.4 we conclude that, $h \leq h_{ij}(t)$ for all times $t \geq r_j$. For any time $t \in [r_j, r_j + F_{ij}(r_j)]$, $F_{ij}(r_j) - (t - r_j)$ measures the remaining flow time of job $j$. We observe that,

$$F_{ij}(r_j) - (t - r_j) \leq \frac{p_{ij}(t)}{h}$$

This follows from a) The right hand side of the inequality measures the projected flow time of the job $j$if it continues to get the speed $h$. b) However, the speed job $j$ gets is lower bounded by $h$ by Proposition 4.4. Now we consider two cases to show that dual constraints are satisfied.

*Case a):* $\alpha > 4$. In this case, note that we can assume without loss of generality that $h = s \cdot \frac{k+1}{n_{it}+k+1} \cdot n_{it}^{1/\alpha}$. This is because due to Proposition 4.4, the function increases in only the range $[1, 2)$ and then starts decreasing. However, if there is only one job it gets the entire speed, it is easy to see that dual constraints are trivially satisfied. Consider,

$$\begin{aligned} & \frac{8y_j - (t - r_j)}{p_{ij}} = \frac{F_{ij}(r_j) - (t - r_j)}{p_{ij}} \leq \frac{1}{h_{ij}(t)} \leq \frac{1}{h} \\ \leq \; & \frac{1}{s} \cdot (1 + \frac{n_{it}}{k+1}) \cdot n_{it}^{-1/\alpha} = \frac{1}{s}(1 + \frac{k+3}{k+1} \cdot 8z_{it}) \leq 1 + 8z_{it} \end{aligned}$$

Note that we used the fact that $s \geq \frac{k+3}{k}$. Arranging terms shows the dual constraint is satisfied.

*Case b):* $\alpha \leq 4$. Suppose $h = s \cdot \frac{k+1}{k+2}$ since we already showed the other case. Then we have,

$$\begin{aligned} & \frac{8y_j - (t - r_j)}{p_{ij}} = \frac{F_{ij}(r_j) - (t - r_j)}{p_{ij}} \leq \frac{1}{h_{ij}(t)} \leq \frac{1}{h} \\ \leq \; & \frac{1}{s} \cdot \frac{k+2}{k+1} \cdot n^{1-1/\alpha} \leq \frac{1}{s} \cdot \frac{k+2}{k+1} \cdot 8(k+3) \cdot z_{it} \leq 1 + 8z_{it} \end{aligned}$$

Again we used the fact that, $s \geq \frac{(k+3)(k+2)}{k+1}$ and $k = \alpha \leq 4$. This concludes the first case (1).

Case 2: *Job $j$ chooses a machine other than $i$.* Say that job $j$ chose to go to machine $i' \neq i$. As in the previous case, we can without loss of generality assume that no more jobs arrive. Throughout the analysis we will be concerned with two schedules $\sigma$ and $\sigma'$.The first schedule $\sigma$ is the real schedule which describes how jobs are processed on machine $i$ since time $r_j$. The second schedule is a fictitious schedule on machine $i$ obtained if job $j$ were assigned to machine $i$, rather $i'$.

We define an one-to-one mapping from time steps in $\sigma$ to time steps $\sigma'$ so that in both schedules all jobs except $j$ have exactly the same remaining processing times at two corresponding times. We will use $g(t)$ to denote the unique time step in $\sigma$ corresponding to $t$ in $\sigma'$. We can obtain the same relation as we did in the second case in showing dual constraints are satisfied in Section 3. The only difference is to take into account dynamic speed scaling, but it turns out that the quantity $g(t) - t$ measuring the delay due to job $j$ can only decrease with dynamic speed scaling. Intuitively, in $\sigma'$ the machine will run faster in the dynamic speed scaling setting compared to the static speed scaling setting.

$$g(t) - t \leq (t - r_j) + 2p_{ij} \tag{9}$$

We are now ready to complete our analysis.

$$\begin{aligned} & \frac{8y_j - (t - r_j)}{p_{ij}} \\ = \; & \frac{F_{i'j}(r_j) - (t - r_j)}{p_{ij}} \\ \leq \; & \frac{F_{ij}(r_j) - (t - r_j)}{p_{ij}} \quad \text{[Since job } j \text{ chose } i' \text{ over } i\text{]} \\ = \; & \frac{F_{ij}(r_j) - (g(t) - r_j) + (g(t) - t)}{p_{ij}} \\ \leq \; & 1 + 8z_{it} + 1 + \frac{(g(t) - t)}{p_{ij}} \qquad (10) \\ \leq \; & 8z_{it} + \frac{t - r_j}{p_{ij}} + 4 \qquad (11) \end{aligned}$$

The inequality (10) follows from the same argument in the previous case since $\sigma'$ is constructed by assigning job $j$ to $i$. Here note that the '1' is added to $n_{it}$ to count job $j$. The inequality (11) follows from (9). By rearranging terms, we complete the analysis.

## Acknowledgements

Part of this works was done while the authors were at Duke, where S. Im was supported in part by NSF grant CCF-1008065, and J. Kulkarni by NSF grants CCF-1408784, IIS- 1447554, and CCF-1348696. Later, S. Im continued this work at UC Merced supported in part by NSF grant CCF-1409130.

## 5. REFERENCES

[1] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4), 2007.

[2] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, pages 1228–1241, 2012.

[3] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 11–18, 2003.

[4] Yossi Azar, Kamal Jain, and Vahab Mirrokni. (almost) optimal coordination mechanisms for unrelated machine scheduling. In *Proceedings of the nineteenth annual*

*ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 323–332, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[5] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 693–701, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[6] Nikhil Bansal and Mor Harchol-Balter. Analysis of srpt scheduling: investigating unfairness. In *SIGMETRICS/Performance*, pages 279–290, 2001.

[7] Nikhil Bansal, Ravishankar Krishnaswamy, and Viswanath Nagarajan. Better scalable algorithms for broadcast scheduling. In *ICALP (1)*, pages 324–335, 2010.

[8] Nikhil Bansal and Janardhan Kulkarni. Minimizing flow-time on unrelated machines. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 851–860, 2015.

[9] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *SODA*, pages 270–279, 1998.

[10] Ioannis Caragiannis. Better bounds for online load balancing on unrelated machines. In *SODA*, pages 972–981, 2008.

[11] Ioannis Caragiannis. Efficient coordination mechanisms for unrelated machine scheduling. In *SODA*, pages 815–824, 2009.

[12] Valeria Cardellini, Michele Colajanni, and Philip S. Yu. Request redirection algorithms for distributed web systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(4):355–368, April 2003.

[13] Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation. In *Symposium on Theory of Computing*, pages 679–684, 2009.

[14] Ho-Leung Chan, Jeff Edmonds, and Kirk Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. *Theory Comput. Syst.*, 49(4):817–833, 2011.

[15] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *STOC*, pages 363–372, 2004.

[16] G. Christodoulou, E. Koutsoupias, and A. Nanavati. Coordination mechanisms. *Theor. Comput. Sci.*, 410(36):3327–3336, August 2009.

[17] Richard Cole, José R. Correa, Vasilis Gkatzelis, Vahab Mirrokni, and Neil Olver. Inner product spaces for minsum coordination mechanisms. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 539–548, New York, NY, USA, 2011. ACM.

[18] Nikhil R. Devanur. Fisher markets and convex programs.

[19] Nikhil R. Devanur and Zhiyi Huang. Primal dual gives almost optimal energy efficient online algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1123–1140, 2014.

[20] Jeff Edmonds, Sungjin Im, and Benjamin Moseley. Online scalable scheduling for the $\ell_k$-norms of flow time without conservation of work. In *ACM-SIAM Symposium on Discrete Algorithms*, 2011.

[21] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 685–692, 2009.

[22] Kyle Fox, Sungjin Im, and Benjamin Moseley. Energy efficient scheduling of parallelizable jobs. In *SODA*, pages 948–957, 2013.

[23] N. Garg and A. Kumar. Better algorithms for minimizing average flow-time on related machines. In *ICALP (1)*, 2006.

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

[25] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1242–1253. SIAM, 2012.

[26] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Scalably scheduling power-heterogeneous processors. In *ICALP (1)*, pages 312–323, 2010.

[27] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: non-clairvoyant scheduling under polyhedral constraints. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 313–322, 2014.

[28] Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 531–540, 2014.

[29] Sungjin Im and Benjamin Moseley. An online scalable algorithm for minimizing $\ell_k$-norms of weighted flow time on unrelated machines. In *ACM-SIAM Symposium on Discrete Algorithms*, 2011.

[30] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42(2):83–97, 2011.

[31] Nicole Immorlica, Li (Erran) Li, Vahab S. Mirrokni, and Andreas S. Schulz. Coordination mechanisms for selfish scheduling. *Theor. Comput. Sci.*, 410(17):1589–1598, April 2009.

[32] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[33] Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. In *STACS*, pages 404–413, 1999.

[34] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)*, pages 351–364, Prague, Czech Republic, 2013.

[35] Nguyen Kim Thang. Lagrangian duality in online scheduling with resource augmentation and speed scaling. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 755–766, 2013.