An Online Scalable Algorithm for Minimizing ℓ_k-norms of Weighted Flow Time on Unrelated Machines

Sungjin Im* Benjamin Moseley†

Abstract

We consider the problem of scheduling jobs that arrive online in the *unrelated* machine model to minimize ℓ_k norms of weighted flowtime. In the unrelated setting, the processing time and weight of a job depends on the machine it is assigned to, and it is perhaps the most general machine model considered in scheduling literature. Chadha et al. [10] obtained a recent breakthrough result in obtaining the first non-trivial algorithm for minimizing weighted flowtime (that is, the ℓ_1 norm) in this very general setting via a novel potential function based analysis. They described a simple algorithm and showed that for any $\epsilon > 0$ it is $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive (a scalable algorithm).

In this paper we give the first non-trivial and scalable algorithm for minimizing ℓ_k norms of weighted flowtime in the unrelated machine model; for any $\epsilon > 0$, the algorithm is $O(k/\epsilon^{2+2/k})$-competitive. The algorithm is immediate-dispatch and non-migratory. Our result is of both practical and theoretical interest. Scheduling to minimize ℓ_k norms of flowtime for some small $k > 1$ has been shown to balance total response time and fairness, which is desirable in practice. On the theoretical side, ℓ_k norms for $k > 1$ pose substantial technical hurdles when compared to when $k = 1$ even for the single machine case. Our work develops a novel potential function as well as several tools that can be used to lower bound the optimal solution.

1 Introduction

Scheduling jobs which arrive online is a fundamental problem faced by a variety of systems. In the standard scheduling setting, a sequence of n jobs arrive over time. Each job has a processing time p_i, which is the amount of time the scheduler must devote to the job. In the *online* model job i is released at time r_i and this is the first time the schedule becomes aware of the job. The goal of a scheduler is to optimize the quality of service. Naturally, the definition of the quality of service depends on the needs of the specific system. One of the most popular and natural metrics at the individual job level is flow time[1]. The flow time of job i is the amount of time that elapses between when the job is released r_i and when the job is completed f_i. The most well-known quality of service measure at the overall jobs level is to minimize the total (or equivalently average) flow time of the schedule $\sum_i f_i - r_i$. This measures the total time all jobs must wait to be completed. Minimizing the total flow time is the most common and popular quality of service measures considered in scheduling theory.

In the simplest setting, each of the jobs is to be scheduled on a single machine. It is well known that the algorithm SRPT (Shortest-Remaining-Processing-Time) gives an optimal schedule. A more complicated scheduling setting is where jobs can be distributed on m machines (processors). In this situation, the scheduler must make the decision of which jobs to assign to which machines along with the decision of how to order jobs on each machine. There are two properties which are desirable for the scheduler. Namely, that the scheduler is *non-migratory* and *immediatly-dispatches* jobs. Migrating a job, which was already assigned to a machine, to another machine may be costly or even impossible. Also, due to memory limitation of the main scheduler, it is more desirable for jobs to be dispatched to some machines immediately upon their arrival, rather than to wait in a pool to be dispatched later.

The simplest multiple machines setting is where all machines are *identical*. That is, each job has the same processing time on all machines and any job can be scheduled on any machine. Average flow time has been studied extensively in the identical machine model [24, 2, 1, 25, 6]. Even in this simple case, the best competitive ratio is $O(\min(\log P, \log n/m))$ and there exists a matching lower bound. Here P is the ratio of the biggest job's processing time to the smallest job's processing time. When a strong lower bound exists, a popular model of analysis is the *resource augmentation* model [23, 26]. In this model, an algorithm A with processors of speed s is compared to the optimal solution with processors of speed 1. For any jobs instance, if the objective value achieved by A is within a fac-

*Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. im3@illinois.edu. Partially supported by NSF grants CCF-0728782, CNS-0721899, and Samsung Fellowship.

†Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. bmosele2@illinois.edu. Partially supported by NSF grants CCF-0728782 and CNS-0721899.

[1] Flow time is also known as wait time and response time

tor c of that by the optimal solution, the algorithm A is said to be s-speed c-competitive for the objective. An algorithm that is $(1+\epsilon)$-speed $O(1)$-competitive algorithm is said to be *scalable*, since it is $O(1)$-competitive when given the smallest amount of resources over the adversary. This is the best positive result in the worst case analysis model that can be obtained for problems which have strong lower bounds on the competitive ratio. In the identical machines setting, [11] gave a scalable algorithm.

In practice machines may not be identical. For instance, machines may have different speed processors. One model that captures this situation is the *related* machines model. Here, each machine x has some speed s_x. Job i requires $\frac{p_i}{s_x}$ time to complete if it is assigned to machine x. The related machines model is of practical interest. However, finding good algorithms has been difficult. There are few positive results known [18, 17]. The best known algorithm is $O(\log^2 P)$-competitive [17].

The related machines model is not general enough to capture the variety of today's systems. Consider the situation where some jobs require lots of memory, but each machine does not have the same amount of memory. Or, perhaps a job can only be scheduled on machines which are attached to a specific input/output device. Here, the relation between machines cannot be easily correlated. To capture this more general model, the *unrelated* machines model has been considered. Here each job i has processing time p_{ix} when assigned to machine x. Due to the variety of machines, the job's processing time can be arbitrarily different depending on the machine the job is assigned to. In fact, the processing times may be infinite on some machines, which captures the case where a job cannot be assigned to a specific machine. The unrelated model, is the most general machine model.

Designing algorithms that are competitive for average flow time on unrelated machines has been difficult. In [19] it was shown that no online algorithm can have a bounded competitive ratio for minimizing the average flow time on unrelated machines without resource augmentation. This lower bound was shown in the restricted case where there are only 3 machines and jobs have either unit size on a machine or infinite size. This example shows that simply restricting jobs to certain machines makes the problem much harder than the related or identical models. Another challenge when designing and analyzing algorithms for unrelated machines is that different schedules can do different amounts of work to satisfy the same set of requests. Here scheduling mistakes are significantly more costly if the optimal solution is doing less work than the algorithm. This differs from the standard scheduling setting where any schedule does the same amount of work to satisfy the same set of jobs. See [12, 20] for other examples. Until recently no non-trivial algorithms were known in the online setting for average flow time. In a breakthrough result a $(1+\epsilon)$-speed $O(1)$-competitive algorithm was given for any fixed $0 < \epsilon \leq 1$ [10]. This was also the first $O(1)$-speed $O(1)$-competitive algorithm shown for the more restricted related machines setting.

Unfortunately, algorithms which focus only on minimizing the total flow time may be extremely unfair to individual jobs by allowing some jobs to starve for an arbitrarily long time. Designing a scheduling algorithm that is fair to all jobs overall is important for practical scheduling algorithms [28]. Due to unfairness, competitive algorithms for average flow time are not often implemented [21, 4]. In practice, it is usually more desirable for the system to be predictable for each job than optimal on average [28, 27]. To balance the fairness and the overall performance, one of the most well-studied and practical metrics is to minimize the ℓ_k-norm of flow time $\sqrt[k]{\sum_{i\in[n]}(f_i - r_i)^k}$ for some $k > 1$. In practice, $k \in [2, 3]$ is usually considered. Minimizing the ℓ_k norms of flow time was first introduced by the influential paper of Bansal and Pruhs [4].

In the ℓ_k norm objective, the algorithm is severely penalized when a job waits a substantial amount of time to be satisfied. This forces the algorithm to minimize the variance of flow time and, thereby, ensures the predictability of the system at the individual job level. Further, in the ℓ_k-norm, the flow time is still being considered and the algorithm must also focus on average quality of service. By optimizing the ℓ_k norm of flow time, the algorithm balances average quality of service and fairness. This makes online algorithms that are competitive for the ℓ_k-norm of flow time highly desirable in practice.

For the ℓ_k-norm of flow time it is well known that without resource augmentation that every deterministic algorithm is $n^{\Omega(1)}$-competitive when $1 < k < \infty$, even on a single machine [4]. This contrasts with average flow time, where SRPT is an optimal algorithm on a single machine. It was shown in [5] that SRPT is a scalable algorithm to minimize the ℓ_k norm of flow time on a single machine for all k. Later a scalable algorithm to minimize the ℓ_k norm of flow time on identical machines was given for any k [11]. There are no known *offline* approximation algorithms for minimizing the ℓ_k norms of flow time for any $k \in [1, \infty)$ without resource augmentation on unrelated machines; note that the ℓ_1 norm is equivalent to average flow time. Further, there are no known non-trivial online algorithms for minimizing the ℓ_k norm of flow time even on related machines with any amount of resource augmentation where $1 < k < \infty$.

In this paper we will be considering the *weighted* ℓ_k norms of flow time of a non-migratory schedule. This is a generalization of the ℓ_k norm objective. Here a job i has a weight w_{ix} when assigned to machines x. The goal of the scheduler is to minimize $\sqrt[k]{\sum_{i\in[n]} w_{iM(x)}(f_i - r_i)^k}$ where $M(x)$ is the machine job x is assigned to. For the weighted ℓ_1 norm of flow time, it was recently shown that

no algorithm can be $O(1)$-competitive without resource augmentation on a single machine [3]. It is well known that the algorithm Highest Density First (HDF) is $(1+\epsilon)$-speed $O(1)$-competitive for the ℓ_1 norm of weighted flow time on a single machine [7]. The algorithm HDF is a natural generalization of SRPT where the scheduler always processes the job i such that $\frac{w_i}{p_i}$ is minimized. In [5] is was shown that HDF is also $(1+\epsilon)$-speed $O(1)$-competitive for the ℓ_k norms of weighted flow time for $k \geq 1$ on a single machine. Until recently, there have been no positive non-trivial results on scheduling to minimize the weighted ℓ_k norms of flow time on multiple machines for any $k > 0$, even in the identical machines setting. The algorithm of [10] for minimizing the ℓ_1 norm of flow time on unrelated machines also considers the case where jobs have weights. Their algorithm is $(1+\epsilon)$-speed $O(1)$-competitive algorithm for the weighted ℓ_1 norm of flow time.

Our Results: In this paper we present the *first* non-trivial competitive algorithm for minimizing the weighted ℓ_k-norm of flow time on unrelated parallel machines when $k > 1$. We show that our algorithm is *scalable* for any fixed $k \geq 1$, i.e. $(1+\epsilon)$-speed $O(1)$-competitive for any fixed $0 < \epsilon \leq 1$. That is, our algorithm is constant competitive when given the minimal extra amount of resources over the adversary. Our algorithm is immediate-dispatch and non-migratory. More specifically, we show the following theorem.

THEOREM 1.1. *For any integer $k \geq 1$ and for any $0 < \epsilon \leq 1$, there exists a $(1+\epsilon)$-speed $O(\frac{k}{\epsilon^{2+2/k}})$-competitive algorithm for minimizing weighted ℓ_k-norms of flowtime on unrelated machines. In particular, the algorithm is immediate dispatch and non-migratory.*

We also show the following lower bound on any randomized immediate-dispatch non-migratory algorithm. This lower bound shows that our algorithm is tight up to a constant factor in the competitive ratio for any fixed $0 < \epsilon \leq 1$.

THEOREM 1.2. *For the problem of minimizing ℓ_k norm of flow time on unrelated machines, any randomized immediate-dispatch non-migratory online algorithm, with any speed $s \geq 1$ given, has competitive ratio $\Omega(\frac{k}{s})$.*

It is important to note that our results translate into the problem of minimizing the ℓ_k norm of stretch. The ℓ_k norm of stretch is $\sqrt[k]{\sum_{i\in[n]} \left(\frac{f_i - r_i}{p_{iM(i)}}\right)^k}$ for some fixed schedule. There is a similar lower bound for the ℓ_k norm of stretch as there is for the ℓ_k norm of flow time on a single machine and jobs have no weight [4]. The ℓ_k norm of stretch can be reduced to the weighted ℓ_k norm of flow time by setting $w_{i,x} = (\frac{1}{p_{i,x}})^k$. Stretch is a popular metric and is used to capture the fact that users expect long jobs to take more time than short jobs. That is, a user is likely to expect to wait for a job to complete in proportion to the job's processing time. This objective is commonly considered in database applications [8, 9]. By using this reduction, our result extends to the stretch setting.

Our Techniques: A potential function based argument will be used to analyze our algorithm. Potential function arguments have recently been shown to be very powerful in scheduling analysis. See [15, 16, 20, 10] for examples. When proving the competitiveness of an online algorithm in scheduling, two approaches can be used. One is a local argument. In a local argument, one shows that at any time the increase in the algorithm's objective function is comparable to the increase in the optimal solution's objective function. This was, for instance, used by [4, 5] to show scalable algorithms for the (weighted) ℓ_k norms of flow time on a single machine. However, for some problems, a local argument cannot be used because this does not hold at all times. This is the case in the unrelated machines model. Here, one option is to use a potential function. The main idea of a potential function is that if the increase in the algorithm's objective is too large at some time then there should be 'credit' in the potential function to pay for this increase.

The potential function we introduce in this paper takes insights from [20, 10]. Most closely related to our potential function is that given in [10] which was used to give a scalable algorithm for minimizing average flow time on unrelated machines. In [10], the algorithm used was to place a job on the machine which increases the ℓ_1 norm of flow time the least and then each machine runs HDF on the jobs assigned to it. Although this algorithm is simple and natural, the potential function used to prove its competitiveness is quite non-trivial. The main idea of the potential function used in [10] is to keep track of the volume of remaining work of the algorithm as compared to the adversary. However, this potential cannot be used in the ℓ_k norm setting because the *ages* of jobs contribute to the increase in the algorithm's objective when $k > 1$, not just the volume of unfinished jobs. Indeed, the increase in the ℓ_k-norm objective at any time grows with the time for which each job has been unsatisfied. This is in contrast to the ℓ_1-norm where the increase in the objective at each time only depends on the number of unsatisfied jobs. Any potential function which does not incorporate the amount of time each job in the algorithm's queue has been unsatisfied for will not be useful for upper-bounding the algorithm's ℓ_k-norm flow time. In this paper, we show a novel potential function that incorporates the volume of remaining work and ages of jobs in the algorithm's queue as compared to the adversary' queue. The potential function combines the ages and volume of jobs in an interesting way. The authors tried natural

generalizations of the potential function of [10] for the ℓ_k norm, however these generalizations do not seem to lead to a $O(1)$-speed $O(1)$-competitive algorithm for the ℓ_k norm of flow time when $k > 1$. Like [10], our algorithm runs HDF on each individual machine. However the machine assignment our algorithm uses of jobs comes from the potential function we derive. We tried to analyze the natural algorithm that assigns a job to a machine that increases the ℓ_k norm flow time the least. However, we were unable to find a potential function that can show this algorithm to be $O(1)$-competitive with speed less than $k + \epsilon$.

It is important to note that in the analysis of [10], the optimal solution was restricted to using HDF on each machine. In our analysis, we will be assuming an *arbitrary* optimal solution. We note that designing a potential function is quite non-trivial for minimizing ℓ_k-norm flow time even on a single processor for any $k \geq 1$. One novelty of our result is a potential function-based argument showing that HDF is scalable on a single machine for any fixed $k \geq 1$. Some lemmas we present in the analysis, which compare our algorithm's status with the arbitrary adversary's status, may be of independent interest.

2 Definitions and Preliminaries

Before introducing our algorithm we introduce notation and given an overview of the analysis. Consider any job sequence σ and let $k \geq 1$ be fixed. Let OPT denote a fixed optimal offline solution with 1 speed that does not migrate jobs. That is, a job is processed on only one machine. Let $s = 1 + 30\epsilon$ be the speed our algorithm is given where $0 < \epsilon < \frac{1}{50}$ is a fixed constant. Let $O_x(t)$ be the set of alive jobs assigned to machine x by OPT. Likewise, $A_x(t)$ will denote the set of unsatisfied jobs assigned to machine x by our algorithm. Let $p_i^O(t)$ be the remaining processing time of job i in OPT's schedule at time t and $p_i^A(t)$ be the remaining processing time of job i in our algorithm's schedule. We define $d_{ix} = \frac{w_{ix}}{p_{ix}}$ to be the density of job i on machine x.

For the rest of the paper, if we say ℓ_k norm flow time, we mean the weighted ℓ_k norm flow time, i.e. we may omit the term 'weighted'. To bound the ℓ_k norm flow time, we will drop the outer kth root and focus on bounding $\sum_{i\in[n]} w_{i,M(i)}(f_i - r_i)^k$, the integral kth power flow time. To do this, we will focus on bounding the *fractional* kth power flow time. The total kth power fractional flow time of a schedule is defined to be $\int_{t=0}^{\infty} \sum_{i\in U(t)} k(t - r_i)^{k-1} p_i^A(t) d_{iM(i)} \mathrm{dt}$, where $U(t)$ is the set of unsatisfied jobs in the given schedule at time t and $M(i)$ is the machine job i is assigned to. Equivalently, the fractional kth power flow time of a schedule is $s\int_{t=0}^{\infty} \sum_{i\in J(t)} (t - r_i)^k d_{iM(i)} \mathrm{dt}$ where $J(t)$ is the set of at most m jobs being processed at time t. There are at most m jobs because a machine can be processing at most 1 job at a time. We will focus mostly on the second definition. We will say that $s\sum_{i\in J(t)}(t - r_i)^k d_{iM(i)}$ is $\frac{\mathrm{dA}}{\mathrm{dt}}$, the increase rate of the fractional flow time of are algorithm during $[t, t + \mathrm{dt}]$. Likewise, $\sum_{i\in J^*(t)}(t - r_i)^k d_{iM^*(i)}$ is $\frac{\mathrm{dO}}{\mathrm{dt}}$ the increase rate of the optimal solutions flow time during $[t, t + \mathrm{dt}]$ where $J^*(t)$ are the jobs OPT works on during $[t, t + \mathrm{dt}]$ and $M^*(i)$ is the machine OPT assigns job i to.

2.1 Relationship between fractional and integral flow time We now discuss the relationship between the fraction kth power flow time and integral kth power flow time. First notice that the fractional kth power flow time is at most the integral kth power flow time. Let $0 < \epsilon' \leq 1$. Consider any schedule. Let f_i' denote the first time that a $\frac{1}{1+\epsilon'}$ fraction of job i is completed. The fractional kth power flow time of job i is at least $\frac{\epsilon'}{1+\epsilon'} w_{(iM(i))}(t - a_i)^k$. Now say that we increase the speed of the schedule by $(1+\epsilon')$. It can be seen that with the extra speed that job i will completed by time f_i'. This implies that the integral kth power flow time of the schedule is at most $\sum_{i\in[n]}(f_i' - r_i)^k$. Thus, the kth power integral flow time of the new schedule is at most $(1 + \frac{1}{\epsilon'})$ times the fractional flow time of the original schedule. Hence, if we show that some algorithm is s-speed c-competitive for fractional flow time then this algorithm is $(1 + \epsilon')s$-speed $c(1 + \frac{1}{\epsilon'})$-competitive for integral kth power flow time. For the rest of this paper, we will focus on bounding the kth power fractional flow time.

2.2 Overview of Analysis As mentioned, we will be using a potential function analysis. We overload notation when the context is clear and let A and OPT denote the total kth power fractional flow time of our algorithm and the adversary, respectively. We will define a potential function Φ that changes over time. Like previous work on potential functions, we design Φ such that it does not increase when jobs are completed by OPT and our algorithm [10, 15, 16, 20]. Further, Φ is 0 before jobs arrive and 0 after all jobs are completed by OPT and the algorithm. The total increase in Φ due to the arrival of jobs will be at most 0. We call this the non-continuous change in Φ because it happens only at instantaneous times when jobs arrive or are completed. We then bound the change in Φ during an infinitesimal time where no jobs arrive or are completed. This is the continuous change in Φ. We will show that the total continuous change over all times is at most $-\gamma A + \delta\mathrm{OPT}$ where γ and δ are constants that can depend on k. These are all the events that effect Φ. Knowing that $\Phi = 0$ at time 0 and time ∞ (any time after all jobs are completed by our algorithm and the adversary), we have the total change in Φ is at most 0. Thus, knowing that $-\gamma A + \delta\mathrm{OPT}$ is an upper bound on the total continuous change of Φ and that non-continuous changes do

not increase Φ, we have that $0 \leq -\gamma A + \delta \text{OPT}$. This implies that $A \leq \frac{\delta}{\gamma}\text{OPT}$. This will complete our analysis.

3 Algorithm and Potential Function

Our algorithm is defined as follows. After jobs are assigned to a machine, each machine runs jobs in a highest density first (HDF) ordering. That is, if i and j are on the same machine x and $d_{xi} > d_{xj}$ then i is processed before j. Without loss of generality, we assume that all jobs have a unique density. Let S be a set of unsatisfied jobs assigned to a single machine. Further, say job $j \in S$ has density less than all other jobs in S. Then in a HDF ordering, job j will have to wait at least $\frac{1}{s}\sum_{i \in S} p_i^A(t)$ time units before it is scheduled when the algorithm has speed s. This fact will be used in our potential function and the arrival condition for our algorithm. To simplify notation, we define $V^{(set)}_{(condition)}$ to be the total processing time (volume) of the jobs in the (set) satisfying the $(condition)$; for example $V^{A_x(t)}_{>d_{ix}} := \sum_{\substack{j \in A_x(t) \\ d_{jx} > d_{ix}}} p_j^A(t)$.

Our algorithm assigns a job to a machine as soon as the job arrives. A job is assigned to the machine which will increase our algorithm's objective function the least, given the current state of the algorithm. However, our algorithm will put less emphasis on the current age of jobs than the remaining amount of time jobs will have to wait to be satisfied. When a job a arrives at time t, it is immediately assigned to a machine x which minimizes the following expression,

$$\begin{aligned}
&\Delta_x(t,a) \\
= \;& d_{ax}\int_{\tau=0}^{p_{ax}} \left(V^{A_x(t)}_{>d_{ix}} + \tau\right)^k \mathrm{d}\tau \\
&+ \sum_{\substack{i \in A_x(t) \\ d_{ix} < d_{ax}}} d_{ix}\Bigg[\int_{\tau=0}^{p_i^A(t)} \left(\epsilon(t-r_i) + V^{A_x(t)}_{>d_{ix}} + p_{ax} + \tau\right)^k \\
&\qquad - \left(\epsilon(t-r_i) + V^{A_x(t)}_{>d_{ix}} + \tau\right)^k\Bigg]\mathrm{d}\tau
\end{aligned}$$

The first term of the arrival condition is used to capture the cost that job a will incur if is assigned to machine x. The second term captures how much job a will increase the fractional kth power flow time of jobs which now will have to wait for job a to finish. These are the jobs which have density less than job a on machine x.

We are now ready to define our potential function. The potential function is designed so that at any time t there is enough credit in the potential function to pay for the kth power flow time of the remaining jobs in the algorithm's queue. To do this, for each job i we put more emphasis on the remaining time job i has to wait before being finished, where credit is gained by our algorithm's doing more work over the adversary via speed augmentation, over the current age of a job. The potential function is also carefully constructed not to increase when jobs arrive. We begin by defining our potential function Φ_x for machine x. Our potential function Φ will then be $\sum_{x \in [m]} \Phi_x$. At time t, we define Φ_x to be,

$$\begin{aligned}
&\Phi_x(t) \\
(3.1) \quad &= \sum_{i \in A_x(t)} d_{ix}\int_{\tau=0}^{p_i^A(t)} \left(\epsilon(t-r_i) + V^{A_x(t)}_{>d_{ix}} + \tau\right)^k \mathrm{d}\tau \\
(3.2) \quad &- \sum_{i \in O_x(t)} d_{ix}\int_{\tau=0}^{p_i^O(t)} \left(\epsilon(t-r_i) + V^{A_x(t)}_{>d_{ix}} + \tau\right)^k \mathrm{d}\tau \\
(3.3) \quad &- \sum_{i \in A_x(t)} d_{ix}\int_{\tau=0}^{p_i^A(t)} \Bigg[\left(\epsilon(t-r_i) + V^{A_x(t)}_{>d_{ix}} + V^{O_x(t)}_{>d_{ix}} + \tau\right)^k \\
&\qquad - \left(\epsilon(t-r_i) + V^{A_x(t)}_{>d_{ix}} + \tau\right)^k\Bigg]\mathrm{d}\tau
\end{aligned}$$

Notice that when a job is completed there is no change in the potential function. Thus, we need only to analyze the change in time, processing by OPT and the algorithm, and the effect of jobs arrival. For the sake of analysis we let $\Phi_{x,1}(t)$, $\Phi_{x,2}(t)$ and $\Phi_{x,3}(t)$ denote the part of $\Phi_x(t)$ in (3.1), (3.2) and (3.4) respectively. The second term $\Phi_{x,2}$ is included to relate the algorithm's queue to the optimal solution's queue. The third term $\Phi_{x,3}$, with the second term, is designed to eliminate the changes in $\Phi_{x,1}$ due to jobs arriving and being placed on machines by the algorithm.

4 Non-continuous Changes

In this section, we study the non-continuous changes, which occur only when new jobs are released. Consider any job a arriving at time t. We now bound the increase in $\Phi(t)$ due to a's arrival. Say that A assigns a to machine x and OPT assigns a to machine $y \neq x$; the case $y = x$ will be addressed later. The changes occur only in Φ_x and Φ_y. It is easy to see that $\Delta\Phi_{x,1}(t) = \Delta_x(t,a)$, and $\Delta\Phi_{x,2}(t), \Delta\Phi_{x,3}(t) \leq 0$.

We now study the change $\Phi_y(t)$ due to the adversary's assigning a to machine y. It is easy to see that $\Delta\Phi_{y,1}(t) = 0$. We can *upperbound* the change in $\Phi_{y,2}(t)$ and $\Phi_{y,3}(t)$ due to the adversary's placement of job a into machine y as follows.

$$\Delta\Phi_{y,2}(t) \quad = \quad -d_{ay}\int_{\tau=0}^{p_{ay}} \left(V^{A_y(t)}_{>d_{ay}} + \tau\right)^k \mathrm{d}\tau$$

The change in $\Phi_{y,3}(t)$ is as follows.

$$\begin{aligned}
&\Delta\Phi_{y,3}(t) \\
= \;& -\sum_{\substack{i \in A_y(t) \\ d_{iy} < d_{ay}}} d_{iy}\int_{\tau=0}^{p_i^A(t)} \Bigg[\left(\epsilon(t-r_i) + V^{A_y(t)}_{>d_{iy}} + V^{O_y(t)}_{>d_{iy}} + p_{ay} + \tau\right)^k \\
&\qquad - \left(\epsilon(t-r_i) + V^{A_y(t)}_{>d_{iy}} + V^{O_y(t)}_{>d_{iy}} + \tau\right)^k\Bigg]\mathrm{d}\tau
\end{aligned}$$

$$\leq -\sum_{\substack{i \in A_y(t) \\ d_{iy} < d_{ay}}} d_{iy} \int_{\tau=0}^{p_i^{A}(t)} \Big[\Big(\epsilon(t-r_i) + V_{>d_{iy}}^{A_y(t)} + p_{ay} + \tau \Big)^k - \Big(\epsilon(t-r_i) + V_{>d_{iy}}^{A_y(t)} + \tau \Big)^k \Big] \mathrm{d}\tau$$

Thus we have that $\Delta\Phi_{y,2}(t) + \Delta\Phi_{y,3}(t) \leq -\Delta_y(t,a)$. By definition of the machine our algorithm places job a on, the total change due to job a's placement is no greater than 0, that is $\Delta\Phi(t) = \Delta\Phi_x(t) + \Delta\Phi_y(t) \leq \Delta_x(t,a) - \Delta_y(t,a) \leq 0$. If A and OPT both assign a to the same machine x, one can easily show that $\Delta\Phi_{x,1}(t) = \Delta_x(t,a)$ and $\Delta\Phi_{x,2}(t) + \Delta\Phi_{x,3}(t) \leq -\Delta_x(t,a)$, thereby obtaining the same result that $\Delta\Phi(t) \leq 0$.

5 Continuous Changes

In this section, we study the continuous change of Φ_x during an infinitesimal interval $[t, t+\mathrm{d}t)$. We will be concentrating on a single machine x. Let OPT_x and A_x denote the total kth power fractional flow time of the jobs assigned to machine x for OPT and the algorithm, respectively. Let T_x denote the first time when all jobs, assigned to machine x, are completed by the algorithm and also by the optimal solution. Let $t_0 = 0, t_1, ..., t_u$ be the times when non-continuous changes occur. For notational purposes let $t_{u+1} = T_x$. It is easy to see that the potential function is differentiable at all times except when non-continuous changes occur. In our analysis, differentiation is used only when it is well defined, which is sufficient for our analysis. Thus by $\int_{t=0}^{\infty} f(t)\mathrm{d}t$, we will mean $\int_{\bigcup_{v=0}^{u}(t_v, t_{v+1})} f(t)\mathrm{d}t$.

Recall that continuous change come from time elapsing and job processing. Job completion and arrival are non-continuous changes, which have been shown to not increase Φ_x. Recall that we assume that our algorithm processes the job of the highest density. Let $a(x,t)$ denote the job of highest density on machine x at time t. Let $b(x,t)$ denote the jobs the optimal solution processes on machine x at time t. For brevity, we will proceed with our analysis assuming that $a(x,t)$ and $b(x,t)$ exist; if $a(x,t)$ or $b(x,t)$ does not exist, the analysis only becomes simpler and the upper bound we will obtain still holds.

Before addressing the change in Φ_x, we show some interesting properties of fractional ℓ_k norm flow time that will be used throughout the analysis. The following two lemmas can be applied to any valid schedule. These will be used later in particular to bound the change in Φ by OPT. In fact, these lemmas can be used as a new lower bound on the optimal solution's schedules. Both lemmas can be applied to any problem sequence where all job are assigned to a single machine. These lemmas may be of independent interest. In our setting, we can just restrict our attention to the jobs which are assigned to some specific machine. For a schedule B on some problem instance $\mathcal{I}$, we let f_i^B denote the finish time of job i at time t under B's schedule. The quantity $p_i^B(t)$ denotes the remaining processing time of job i at time under B's schedule. Let $B(t)$ denote the alive jobs in the queue under the schedule of B. We let $B(\mathcal{I})$ denote the total kth power fractional weighted flow time of $B(\mathcal{I})$'s schedule.

LEMMA 5.1. *Consider any given instance $\mathcal{I}$ where all jobs are assigned to a single machine. Then for any valid schedule B, with speed s' given, on the instance $\mathcal{I}$,*

$$\int_{t=0}^{\infty} \sum_{i \in B(t)} d_i \int_{\tau=0}^{\frac{p_i^B(t)}{s'}} k\Big(\frac{V_{>d_i}^{B(t)}}{s'} + \tau \Big)^{k-1} \mathrm{d}\tau \mathrm{d}t \leq \frac{1}{s'} B(\mathcal{I}).$$

To have a feel of the above lemma, consider when $k = 1$, $s' = 1$ and an infinitesimal time interval $[d, d+\mathrm{d}t)$. Then the change in (LHS) during the interval is $\mathrm{d}t \sum_{i \in B(t)} d_i p_i^B(t)$, which is exactly the increase of weighted fractional flow time during the interval. When $k = 1$ this relation is known to be folklore, and usefully used in scheduling theory. It is, however, not so obvious when $k > 1$. The proof is placed in Appendix B.

LEMMA 5.2. *Consider any given instance $\mathcal{I}$ where all jobs are assigned to a single machine. Suppose B is a valid schedule with s' speed given on the instance $\mathcal{I}$. Let v be any constant and $V_{>v}^{B(t)} = \sum_{i \in B(t), d_i > v} p_i^{B(t)}$. Then,* $(V_{>v}^{B(t)})^k \leq \sum_{\substack{i \in B(t) \\ d_i > v}} d_i \int_{\tau=0}^{p_i^B(t)} k(V_{>v}^{B(t)} + \tau)^{k-1} \mathrm{d}\tau$. *In particular, for any function $g(t) : [0, \infty) \to \mathbb{R}$, it holds that* $\int_{t=0}^{\infty} g(t) \Big(V_{>g(t)}^{B(t)} / s' \Big)^k \mathrm{d}t \leq \frac{1}{s'} B(\mathcal{I})$.

The above lemma, with Lemma 5.1, will enable the analysis without making any assumption on the adversary's scheduling. Especially, Lemma 5.2 is interesting for the following reason. In the lemma, (LHS) is an expression involving two quantifies which are not related at all. One is a quantity regarding to volume of alive jobs under some schedule B, and the other is any arbitrary function $g(t)$. Due to this lemma, we will be able to bound the changes involving our algorithm's queue status and the adversary's schedule, without explicitly correlating the two. The following corollaries are immediate from the two lemmas.

COROLLARY 5.1.

$$\int_{t=0}^{\infty} d_{a(x,t)} (V_{>d_{a(x,t)}}^{O(t)})^k \mathrm{d}t \leq \mathrm{OPT}_x.$$

COROLLARY 5.2.

$$\int_{t=0}^{\infty} \sum_{i \in O_x(t)} d_{ix} \int_{\tau=0}^{p_i^O(t)} k(V_{>d_{ix}}^{O(t)} + \tau)^{k-1} \mathrm{d}\tau \mathrm{d}t \leq \mathrm{OPT}_x.$$

The following proposition will be used throughout the analysis.

PROPOSITION 5.1. *For any constant $0 \leq \epsilon < 1$ and any integer $k \geq 1$, $(1+\frac{\epsilon}{k})^k \leq 1+2\epsilon$.*

The following two lemmas easily follow using the definition of kth power of fractional flow time.

LEMMA 5.3.

$$\begin{aligned}&\int_{t=0}^{\infty} d_{b(x,t)}\Big(\epsilon(t-r_{b(x,t)})+p^O_{b(x,t)}(t)\Big)^k \mathrm{d}t\\ \leq\ & 2(1+\epsilon)^k \mathrm{OPT}_x \leq 2^{k+1}\mathrm{OPT}_x\end{aligned}$$

LEMMA 5.4.

$$\begin{aligned}&\int_{t=0}^{\infty}\sum_{i\in O_x(t)} d_{ix}\int_{\tau=0}^{p_i^O(t)} k(\epsilon(t-r_i)+\tau)^{k-1}\mathrm{d}\tau\mathrm{d}t\\ \leq\ & 2(1+\epsilon)^{k-1}\mathrm{OPT}_x \leq 2^k \mathrm{OPT}_x.\end{aligned}$$

We are now ready to bound the continuous changes in Φ_x.Since we are focusing on each specific machine x, as far as there is no specific reason to highlight the machine, we drop x from the notation. Note that the changes in $\Phi(t)$ during $[t, t+\mathrm{d}t]$ occur because of job processing and the change in time. The job $a(t)$ is processed by the algorithm by an amount of $s\mathrm{d}t$, since the algorithm has s speed. The job $b(t)$ is processed by OPT by an amount of $\mathrm{d}t$, since OPT has 1 speed. Further, time t will increase by $\mathrm{d}t$. These are the only changes affecting Φ when no jobs arrive or are completed. Recall that our goal is to upper bound the total change in Φ over all time by a multiplicative factor of A and OPT. In the continuous analysis our goal will be to bound the total change in Φ_x by OPT_x and A_x for each specific machine x.

5.1 Analyzing $\frac{\mathrm{d}}{\mathrm{d}t}\Phi_{x,1}(t)$

$$\begin{aligned}&\frac{\mathrm{d}}{\mathrm{d}t}\Phi_{x,1}(t)\\ =\ & \sum_{i\in A(t)\setminus\{a(t)\}} d_i\int_{\tau=0}^{p_i^A(t)}(\epsilon-s)k\Big(\epsilon(t-r_i)+V^{A(t)}_{>d_i}+\tau\Big)^{k-1}\mathrm{d}\tau\\ & +d_{a(t)}(\epsilon-s)\Big(\epsilon(t-r_{a(t)})+p^A_{a(t)}(t)\Big)^k\\ & -\epsilon d_{a(t)}\Big(\epsilon(t-r_{a(t)})\Big)^k\\ =\ & -(s-\epsilon)\sum_{i\in A(t)} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V^{A(t)}_{>d_i}+\tau\Big)^{k-1}\mathrm{d}\tau\\ & -sd_{a(t)}\Big(\epsilon(t-r_{a(t)})\Big)^k \qquad (5.4)\end{aligned}$$

5.2 Analyzing $\frac{\mathrm{d}}{\mathrm{d}t}\Phi_{x,2}(t)$

$$\begin{aligned}&\frac{\mathrm{d}}{\mathrm{d}t}\Phi_{x,2}(t)\\ =\ & -\frac{\mathrm{d}}{\mathrm{d}t}\sum_{i\in O(t)} d_i\int_{\tau=0}^{p_i^O(t)}\Big(\epsilon(t-r_i)+V^{A(t)}_{>d_i}+\tau\Big)^k\mathrm{d}\tau =\\ & +(s-\epsilon)\sum_{\substack{i\in O(t)\\ d_i<d_{a(t)}}} d_i\int_{\tau=0}^{p_i^O(t)} k\Big(\epsilon(t-r_i)+V^{A(t)}_{>d_i}+\tau\Big)^{k-1}\mathrm{d}\tau \qquad (5.5)\\ & -\epsilon\sum_{\substack{i\in O(t)\\ d_i\geq d_{a(t)}}} d_i\int_{\tau=0}^{p_i^O(t)} k\Big(\epsilon(t-r_i)+\tau\Big)^{k-1}\mathrm{d}\tau \qquad (5.6)\\ & +d_{b(t)}\Big(\epsilon(t-r_{b(t)})+V^{A(t)}_{>d_{b(t)}}+p^O_{b(t)}(t)\Big)^k \qquad (5.7)\end{aligned}$$

Lines (5.5) and (5.6) are due to the change in time and the algorithm's processing. Line (5.7) is from OPT's processing. We are concerned with upper bounding the change in Φ, so can ignore (5.6). We will first bound $\int_{t=0}^{\infty}(5.5)$ and then we will concentrate on bounding $\int_{t=0}^{\infty}(5.7)$.

5.2.1 Bounding the total change of (5.5) over time

By considering whether $V^{A(t)}_{>d_i} \leq \frac{k}{\epsilon}(\epsilon(t-r_i)+\tau)$ or not, we have

$$\begin{aligned}&\int_{t=0}^{\infty}(5.5)\mathrm{d}t \leq\\ &(s-\epsilon)(1+\frac{k}{\epsilon})^{k-1}\int_{t=0}^{\infty} d_i\sum_{\substack{i\in O(t)\\ d_i<d_{a(t)}}}\int_{\tau=0}^{p_i^O(t)} k(\epsilon(t-r_i)+\tau)^{k-1}\mathrm{d}\tau\mathrm{d}t\\ &+(s-\epsilon)(1+\frac{\epsilon}{k})^{k-1}k\int_{t=0}^{\infty}\sum_{\substack{i\in O_x(t)\\ d_i<d_{a(t)}}} d_i p_i^O(t)(V^{A(t)}_{>d_i})^{k-1}\mathrm{d}t \qquad (5.8)\end{aligned}$$

(5.9)

Via simple algebra and Lemma 5.4, we have $(5.8) \leq (\frac{4k}{\epsilon})^k\mathrm{OPT}_x$. The second term (5.9) can be bounded by the following lemma.

LEMMA 5.5.

$$\begin{aligned}&\int_{t=0}^{\infty}\sum_{i\in O_x(t)} d_i p_i^O(t)(V^{A(t)}_{>d_i})^{k-1}\mathrm{d}t\\ \leq\ & \frac{1}{k}(\frac{k}{\epsilon})^{k-1}\mathrm{OPT}_x\\ & +3\epsilon\int_{t=0}^{\infty}\sum_{i\in A(t)} d_i\int_{\tau=0}^{p_i^A(t)}(V^{A(t)}_{>d_i}+\tau)^{k-1}\mathrm{d}\tau\mathrm{d}t\end{aligned}$$

We give a sketch of the proof here deferring the formal proof to Appendix B. We partition $O_x(t)$ into $I(t)$ and $I'(t)$ depending on whether a job i satisfies $V^{A(t)}_{>d_i} \leq \frac{k}{\epsilon}V^{O(t)}_{>d_i}$ or not. For the easier set $I(t)$ the total cost of (5.5) restricted to

$I(t)$ over time can be bounded by a multiplicative of OPT_x using Corollary 5.2. For the other set $I'(t)$, we use the fact that there exists a large volume of jobs of density bigger than d_i for each job $i \in I'(t)$ in A's queue than in OPT's queue. Then we will be able to show that the total cost of (5.5) regarding to $I'(t)$ is the second term in the inequality in the lemma.

By summing each of these expressions we have,

$$\begin{aligned}
&\int_{t=0}^{\infty}(5.5)\mathrm{d}t \\
\leq\ & 3(\frac{4k}{\epsilon})^k\text{OPT}_x \\
&+9\epsilon(s-\epsilon)\int_{t=0}^{\infty}\int_{\tau=0}^{p_i^A(t)} k(V_{>d_i}^{A(t)}+\tau)^{k-1}\mathrm{d}\tau\mathrm{d}t
\end{aligned} \tag{5.10}$$

5.2.2 Bounding the total change of (5.7) over time By considering whether $V_{>d_{b(t)}}^{A(t)} \leq \frac{k}{\epsilon}\Big(\epsilon(t-r_{b(t)})+p_{b(t)}^O(t)\Big)$ or not, we have

$$\begin{aligned}
&\int_{t=0}^{\infty}(5.7)\mathrm{d}t \\
\leq\ & (1+\frac{\epsilon}{k})^k\int_{t=0}^{\infty} d_b(t)(V_{>d_{b(t)}}^{A(t)})^k \\
&+(1+\frac{k}{\epsilon})^k d_b(t)\Big(\epsilon(t-r_{b(t)})+p_{b(t)}^O(t)\Big)^k\mathrm{d}t \\
\leq\ & (1+2\epsilon)\int_{t=0}^{\infty}\sum_{\substack{i\in A(t)\\ d_i>d_{b(t)}}} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(V_{>d_i}^{A(t)}+\tau\Big)^{k-1}d\tau\mathrm{d}t \\
&+2(\frac{4k}{\epsilon})^k\text{OPT}_x
\end{aligned} \tag{5.11}$$

The last inequality follows by applying Lemma 5.2 and 5.3.

5.3 Analyzing $\frac{\mathrm{d}}{\mathrm{d}t}\Phi_{x,3}(t)$ We first study the change coming from our algorithm's processing and time elapse, which is

$$(s-\epsilon)\sum_{i\in A_x(t)} d_i\int_{\tau=0}^{p_i^A(t)} k\Big[\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+V_{>d_i}^{O(t)}+\tau\Big)^{k-1} \tag{5.12}$$

$$-\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\Big]\mathrm{d}\tau$$

$$+sd_{a(t)}\Big[\Big(\epsilon(t-r_{a(t)})+V_{>d_{a(t)}}^{O(t)}\Big)^k-\Big(\epsilon(t-r_{a(t)})\Big)^k\Big] \tag{5.13}$$

We need the following lemma whose proof is very similar to that of Lemma 5.5. The lemma is slightly different from Lemma 5.5; roughly speaking the (LHS) is from the algorithm's perspective rather than from the optimal solution's. The proof can be found in Appendix B.

LEMMA 5.6.

$$\begin{aligned}
&\int_{t=0}^{\infty}\sum_{i\in A(t)} d_i p_i^A(t)(V_{>d_i}^{O(t)})^{k-1}\mathrm{d}t \\
\leq\ & (\frac{\epsilon^2}{k})^{k-1}\int_{t=0}^{\infty} d_i\sum_{i\in A(t)}\int_{\tau=0}^{p_i^A(t)}(V_{>d_i}^{A(t)}+\tau)^{k-1}\mathrm{d}\tau\mathrm{d}t \\
&+\frac{3k}{\epsilon^2}\text{OPT}_x.
\end{aligned}$$

We first bound $\int_{t=0}^{\infty}(5.12)\mathrm{d}t$. We can assume that $k\geq 2$ since $(5.12)=0$ when $k=1$. By considering whether or not $V_{>d_i}^{O(t)} \leq \frac{\epsilon}{k}\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)$, and via simple algebra, we have

$$\begin{aligned}
&(5.12) \\
\leq\ & 2\epsilon(s-\epsilon)\sum_{i\in A_x(t)} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau \\
&+(1+\frac{k}{\epsilon})^{k-1}(s-\epsilon)k\sum_{i\in A_x(t)} d_i p_i^A(t)(V_{>d_i}^{O(t)})^{k-1}
\end{aligned}$$

By Lemma 5.6, the fact $s\leq 2$, and a simple algebra it follows that

$$\begin{aligned}
&\int_{t=0}^{\infty}(5.12)\mathrm{d}t \\
\leq\ & 5\epsilon(s-\epsilon)\int_{t=0}^{\infty}\sum_{i\in A_x(t)} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau\mathrm{d}t \\
&+(\frac{2k}{\epsilon})^{k+1}\text{OPT}_x
\end{aligned} \tag{5.14}$$

We now bound (5.13). By considering whether $V_{>d_{a(t)}}^{O(t)} \leq \frac{\epsilon}{k}(\epsilon(t-r_a))$ or not, we have

$$(5.13) \leq 2\epsilon s d_{a(t)}\Big(\epsilon(t-r_{a(t)})\Big)^k+s(1+\frac{k}{\epsilon})^k d_{a(t)}(V_{>d_{a(t)}}^{O(t)})^k.$$

Thus, by Corollary 5.1,

$$\int_{t=0}^{\infty}(5.13)\mathrm{d}t \leq 2\epsilon s d_{a(t)}\int_{t=0}^{\infty}\Big(\epsilon(t-r_{a(t)})\Big)^k\mathrm{d}t+2(\frac{2k}{\epsilon})^k\text{OPT}_x \tag{5.15}$$

We now study the change coming from OPT's processing in Φ_3, which is as follows.

$$\sum_{\substack{i\in A(t)\\ d_i<d_{b(t)}}} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+V_{>d_i}^{O(t)}+\tau\Big)^{k-1}\mathrm{d}\tau \tag{5.16}$$

$$\leq \quad (1+\frac{\epsilon}{k})^{k-1}\sum_{\substack{i\in A(t)\\ d_i<d_{b(t)}}} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau$$
$$+(1+\frac{k}{\epsilon})^{k-1}\sum_{\substack{i\in A(t)\\ d_i<d_{b(t)}}} k d_i p_i^A(t)(V_{>d_i}^{O(t)})^{k-1}$$

The inequality easily follows by considering whether $V_{>d_i}^{O(t)} \leq \frac{\epsilon}{k}\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)$ or not. And by Lemma 5.6 and simple algebra, we obtain

$$\int_{t=0}^{\infty}(5.16)\mathrm{d}t$$
$$\leq (1+2\epsilon)\int_{t=0}^{\infty}\sum_{\substack{i\in A(t)\\ d_i<d_{b(t)}}} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau\mathrm{d}t$$

(5.17)

$$+3\epsilon\int_{t=0}^{\infty}\sum_{i\in A(t)} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau\mathrm{d}t+(\frac{2k}{\epsilon})^{k+1}\mathrm{OPT}_x$$

(5.18)

Here (5.18) was obtained assuming $k \geq 2$. When $k = 1$, $\int_{t=0}^{\infty}(5.16)\mathrm{d}t \leq (5.17)$. Thus the above upper bound holds for all $k \geq 1$.

6 Final Analysis

We complete our analysis by aggregating all changes, both non-continuous and continuous. By gathering all continuous changes for each machine $x \in [m]$ studied in the previous section, we obtain

$$\begin{aligned}
&\int_{t=0}^{\infty}\frac{\mathrm{d}}{\mathrm{d}t}\Phi_x(t)\mathrm{d}t\\
=\quad &\int_{t=0}^{\infty}[\frac{\mathrm{d}}{\mathrm{d}t}\Phi_{x,1}(t)+\frac{\mathrm{d}}{\mathrm{d}t}\Phi_{x,2}(t)+\frac{\mathrm{d}}{\mathrm{d}t}\Phi_{x,3}(t)]\mathrm{d}t\\
\leq\quad &\int_{t=0}^{\infty}(5.4)\mathrm{d}t+(5.10)+(5.11)+(5.14)\\
&\qquad +(5.15)+(5.17)+(5.18)\\
\leq\quad &((1+2\epsilon)-(1-17\epsilon)(s-\epsilon))\cdot\\
&\sum_{i\in A_x(t)} d_{ix}\int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_{ix}}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau\\
&-s(1-2\epsilon)\epsilon^k\int_{t=0}^{\infty} d_{a(x,t),x}(t-r_{a(x,t)})^k\mathrm{d}t\\
&+8(\frac{4k}{\epsilon})^{K+1}\mathrm{OPT}_x\\
\leq\quad &-s(1-2\epsilon)\epsilon^k A_x+8(\frac{4k}{\epsilon})^{k+1}\mathrm{OPT}_x
\end{aligned}$$

The second inequality can be obtained by combining (5.11) with (5.17). The last inequality holds when $1+30\epsilon \leq s \leq 2$, and $0 < \epsilon \leq \frac{1}{50}$. Since Φ is 0 before no jobs are released and also after all jobs are completed by A and OPT, the total change of Φ is 0. Recall that the sum of non-continuous changes is at most 0. Thus the total continuous change of Φ over time is non-negative. Hence from the above inequality for each machine x, we have $0 \leq \sum_{x\in[m]}\int_{t=0}^{\infty}\frac{\mathrm{d}}{\mathrm{d}t}\Phi(t)_x\mathrm{d}t \leq -s(1-2\epsilon)\epsilon^k A+8(\frac{4k}{\epsilon})^{k+1}\mathrm{OPT}$.

Thus we obtain $(A)^{1/k} \leq O(\frac{k}{\epsilon^{2+1/k}})(\mathrm{OPT})^{1/k}$. By scaling ϵ appropriately in the algorithm we have the following theorem,

THEOREM 6.1. *For any integer $k \geq 1$ and for any $0 < \epsilon \leq 1$, there exists a $(1+\epsilon)$-speed $O(\frac{k}{\epsilon^{2+1/k}})$-competitive algorithm for minimizing weighted ℓ_k-norms of fractional flowtime on unrelated machines. In particular, the algorithm is immediate dispatch and non-migratory.*

Using the relation between integral kth power flow time and fractional kth power flow time discussed in Section 2.1, we can show Theorem 1.1.

7 Lowerbound on the competitive ratio of the algorithm

In this section we prove Theorem 1.2.

Proof. Suppose that we have $m = 2^k$ machines. We create the following adversarial instance $\mathcal{I}$. It has k groups of jobs: G_α, $\alpha \in [k]$. Jobs in each group (set) G_α can be assigned to only a subset of machines $\mathcal{M}_\alpha$ where $|G_\alpha| = |\mathcal{M}_\alpha| = 2^{k+1-\alpha}$. All jobs have uniform size. For simplicity, we assume that all jobs are released at time 0, but the algorithm is given jobs to schedule one by one; this can be simulated by letting jobs have sufficiently large size and arrive at distinct integer times during $[0, 2m]$. We will assume that the jobs in G_i arrives before G_j if $i < j$.

Let $\mathtt{LOAD}_\alpha$ denote the average load of the machines $\mathcal{M}_\alpha$ for jobs in group $G_1, G_2, ..., G_\alpha$, i.e. the number of jobs from $G_1, G_2, ..., G_\alpha$ assigned to the machines $\mathcal{M}_\alpha$ divided by $|\mathcal{M}_\alpha|$. We will decide $\mathcal{M}_\alpha$ in an adversarial manner so that $\mathtt{LOAD}_\alpha \geq \alpha$. Also we maintain $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_\alpha$ form an inclusion-wise chain, that is $\mathcal{M}_\alpha \subseteq ... \subseteq \mathcal{M}_2 \subseteq \mathcal{M}_1$. All jobs in the first group G_1 can be assigned to any machine, i.e. $\mathcal{M}_1 = [m]$. Then each $\mathcal{M}_\alpha$ for $\alpha \geq 2$ is inductively defined, after the algorithm's decision on the jobs $G_{\alpha-1}$, to be the half machines from $\mathcal{M}_{\alpha-1}$ having the largest number of jobs assigned. Using the fact that the average load on $\mathcal{M}_\alpha$ is at least $\mathtt{LOAD}_{\alpha-1}$ and adding G_α will increase the average load by at least one, it can be easily shown that $\mathtt{LOAD}_\alpha \geq \alpha$ for all $\alpha \in [k]$.

Note that the algorithm has kth power of flow time at least $(k/s)^k$ due to the jobs in group G_k. On the other hand, there exists a schedule where every machine is assigned at most two jobs: all and only jobs in G_α are assigned to $\mathcal{M}_\alpha \setminus \mathcal{M}_{\alpha+1}$; $\mathcal{M}_{k+1} = \emptyset$. This can be easily seen by noting that $\mathcal{M}_\alpha \setminus \mathcal{M}_{\alpha+1}, \alpha \in [k]$ are disjoint sets of machines and $|G_\alpha|/|\mathcal{M}_\alpha \setminus \mathcal{M}_{\alpha+1}| = 2$. Thus this schedule has kth power of flow time at most $m(1^k + 2^k) = 2^k(2^k + 1)$. The desired lower bound on the competitive ratio immediately follows.

8 Discussions and Conclusions

In this paper we introduced a scalable algorithm for the ℓ_k norm of flow time in the unrelated machines model for any fixed $k > 0$. It is important to note that our algorithm knows the speed (ϵ). That is the algorithm uses ϵ to determine the machine assignment of jobs. Knowing the speed the algorithm is given has recently been shown to be useful in scheduling analysis [22, 16, 13, 14]. One possible candidate algorithm that does not depend on ϵ is the algorithm that assigns a job to the machines which increases the (fractional) kth norm of flow time the least. We currently do not know if this algorithm is scalable or not. The authors were able only to show that it is $O(1)$-competitive with speed $k + \epsilon$. It would be of interest to give a scalable algorithm that does not depend on ϵ or show that no such algorithm exists. Our lower bound is restricted to immediate-dispatch algorithms. Thus there may exist a scalable non-immediate-dispatch algorithm whose competitive ratio does not grow with k. It would be interesting if one can obtain a scalable algorithm compared to the migratory optimal schedule.

Acknowledgements: We thank Chandra Chekuri for many helpful discussions and his support.

References

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

[2] Baruch Awerbuch, Yossi Azar, Stefano Leonardi, and Oded Regev. Minimizing the flow time without migration. *SIAM J. Comput.*, 31(5):1370–1382, 2002.

[3] Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 1238–1244, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[4] Nikhil Bansal and Kirk Pruhs. Server scheduling in the lp norm: a rising tide lifts all boat. In *STOC*, pages 242–250, 2003.

[5] Nikhil Bansal and Kirk Pruhs. Server scheduling in the weighted lp norm. In Martin Farach-Colton, editor, *LATIN*, volume 2976 of *Lecture Notes in Computer Science*, pages 434–443, 2004.

[6] Luca Becchetti and Stefano Leonardi. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *J. ACM*, 51(4):517–539, 2004.

[7] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *J. Discrete Algorithms*, 4(3):339–352, 2006.

[8] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 270–279, 1998.

[9] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Improved algorithms for stretch scheduling. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 762–771, 2002.

[10] 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 *STOC*, pages 679–684, 2009.

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

[12] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Longest wait first for broadcast scheduling. In *WAOA '09: Proceedings of 7th Workshop on Approximation and Online Algorithms*, 2009.

[13] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Minimizing maximum response time and delay factor in broadcasting scheduling. In *ESA '09: Proceedings of the seventeenth annual European symposium on algorithms*, pages 444–455, 2009.

[14] Chandra Chekuri and Benjamin Moseley. Online scheduling to minimize the maximum delay factor. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 1116–1125, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[15] Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000.

[16] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 685–692, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[17] Naveen Garg and Amit Kumar. Better algorithms for minimizing average flow-time on related machines. In *ICALP (1)*, pages 181–190, 2006.

[18] Naveen Garg and Amit Kumar. Minimizing average flow time on related machines. In Jon M. Kleinberg, editor, *STOC*, pages 730–738. ACM, 2006.

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

[20] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling jobs with varying parallelizability to reduce variance. In *SPAA '10: 22nd ACM Symposium on Parallelism in Algorithms and Architectures*, 2010.

[21] Mor Harchol-Balter, Mark E. Crovella, and Sungsim Park. The case for srpt scheduling in web servers. Technical report, 1998.

[22] Sungjin Im and Benjamin Moseley. An online scalable algorithm for average flow time in broadcast scheduling. In *SODA '10: Proceedings of the Twentieth Annual ACM -SIAM Symposium on Discrete Algorithms*, 2010.

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

[24] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *J. Comput. Syst. Sci.*, 73(6):875–

891, 2007.

[25] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[26] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Perform. Eval. Rev.*, 34(4):52–58, 2007.

[27] Abraham Silberschatz and Peter Galvin. *Operating System Concepts, 4th edition*. Addison-Wesley, 1994.

[28] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2007.

A Slicing transformation

In this section we explain a "slicing" technique that will be used for our analysis. For this technique, we will focus on an instance where there is a single machine and since we are focusing on a single machine, we drop the machine x notation. Let s' denote the speed the schedule considered is given. In the slicing technique each job i is replaced with a set of jobs $\mathcal{J}_i$ of uniform processing time $\Delta' = s'\Delta$ and uniform weight $w'_i = \frac{w_i\Delta'}{p_i}$, where Δ is a sufficiently small constant. Note that each new job having size $s'\Delta$ in $\mathcal{J}_i$ requires Δ amount of time to be finished. There are a total of $\frac{p_i}{\Delta'}$ jobs in $\mathcal{J}_i$. Notice that each job's density is the same as that of job i and total volume of jobs in $\mathcal{J}_i$ is the same as the size of job i, i.e. $p_i = V^{\mathcal{J}_i}$. These jobs all arrive at the same time job i arrives. This method was used in [5] to reduce the problem of minimizing ℓ_k-norm of weighted flow time to its unweighted version. To our best knowledge, it has not been formally stated anywhere that the slicing transformation does not affect the weighted *fractional* ℓ_k norm of flow time.

To formally define the transformation, we need more notation. For the (old) given instance of jobs $\mathcal{I}$, let $\mathcal{B}(\mathcal{I})$ denote the schedule under the scheduling policy B. We call jobs in $\mathcal{I}'$ new to distinguish them from jobs in $\mathcal{I}$. The new schedule $\mathcal{B}'(\mathcal{I}')$ for the new instance $\mathcal{I}'$ is naturally defined from the old schedule $\mathcal{B}(\mathcal{I})$. That is, at any time t, job i is processed under $\mathcal{B}(\mathcal{I})$ if and only if a job in $\mathcal{J}_i$ is processed under $\mathcal{B}'(\mathcal{I}')$; this mapping is well-defined since the slicing preserves the volume in replacing each job i with the jobs $\mathcal{J}_i$. For each i, jobs in $\mathcal{J}_i$ are ordered in an arbitrary but fixed way. We let $\mathcal{J}_i(t)$ denote the jobs in $\mathcal{J}_i$ which are alive at time t. Note that $p_i(t)$ in $\mathcal{B}$'s schedule is the same as $V^{\mathcal{J}_i(t)}$ in $\mathcal{B}'$'s schedule. We say that an objective function (or expression) `fFUNC` is resilient to slicing if it gives the same value for two instances $\mathcal{I}$ and $\mathcal{I}'$. When $\mathcal{I}$ and $\mathcal{I}'$ are well-understood in the context, they may be dropped. The following lemma easily follows from the definition of kth power of weighted fractional flow time and the slicing transformation.

LEMMA A.1. *The weighted kth power of fractional flow time is resilient to slicing for any schedule $\mathcal{B}$.*

Proof of [Lemma A.1] Consider a unit time slot $[t, t+\Delta)$. Let $j \in \mathcal{J}_i$ be the job in $\mathcal{I}'$ completed during the time slot. Its contribution to weighted kth power of fractional flow time is $\int_{\tau=0}^{\Delta} d_j(t - r_j + \tau)^k s' \mathrm{d}\tau$. In the instance $\mathcal{I}$, Δ' amount of work for job i is done, which gives exactly the same contribution. □

We now discuss the relationship between *integral* kth power flow time of the new instance and the *fractional* kth power flow time of the old instance. We assume that the time is partitioned into unit time slots of size Δ and during each time slot exactly one job is completed. WLOG we can further assume that jobs arrive only at the beginning of a time slot. These are valid assumptions assuming Δ is sufficiently small. We let $\mathcal{T}$ denote the set of unit times. We will be interested in considering the *integral* kth power flow time of the new instance. Let $N^{(set)}_{(condition)}$ denote the number of alive jobs in the *(set)* that satisfy the *(condition)*. In the slotted time model, we will consider `dFUNC`, the discrete version of `fFUNC` for the new instance $\mathcal{I}'$. This will be explicitly defined when it is used. When there is a need to stress that $\mathcal{I}'$ is obtained by slicing jobs into $(s'\Delta)$-sized jobs, we will use $\mathcal{I}'(\Delta)$.

Then if $\lim_{\Delta\to 0} \texttt{dFUNC}(\mathcal{I}'(\Delta)) = \texttt{fFUNC}(\mathcal{I})$, we will be able to work with the discrete version of the function `dFUNC` for $\mathcal{I}'$ to obtain the desired result regarding to the given function `fFUNC` for $\mathcal{I}$. We will move between the discrete and continuous time models depending on whichever gives an easier analysis. Notice that this property holds for between fractional kth power flow time of the original instance and integral kth power flow time of the new instance when $\Delta \to 0$.

B Omitted Proofs

Proof of [Lemma 5.1] By Lemma A.1, we know that (RHS) is resilient to slicing. We first show that so is (LHS). Recall that we use $\mathcal{I}'$ to denote the new instance obtained by slicing jobs. Let `fFUNC` denote the function which takes an instance and gives the value of (LHS) on the schedule by B on the instance. To save notation, for job $i \in B(t)$ and for any job $j \in \mathcal{J}_i(t)$, we will abuse the notation $_{>d_j}$ to include not only the jobs of density at least d_j but also the jobs in $\mathcal{J}_i(t)$ of the same density which are finished before job j in the schedule B'. Consider an infinitesimal interval $[t, t+\mathrm{d}t)$. Then the change in $\texttt{fFUNC}(B(\mathcal{I}))$ for a job i is $d_i \int_{\tau=0}^{p_i^B(t)/s'} k((V^{B(t)}_{>d_i}/s') + \tau)^{k-1}\mathrm{d}\tau\mathrm{d}t$. It can be easily shown to be equal to $\sum_{j\in\mathcal{J}_i(t)} d_i \int_{\tau=0}^{\Delta} k((V^{B(t)}_{>d_i}/s') + (V^{\mathcal{J}_i(t)}_{>d_j}/s') + \tau)^{k-1}\mathrm{d}\tau\mathrm{d}t = \sum_{j\in\mathcal{J}_i(t)} d_j \int_{\tau=0}^{\Delta} k((V^{B'(t)}_{>d_j}/s')+\tau)^{k-1}\mathrm{d}\tau\mathrm{d}t$, which is exactly the increase in $\texttt{fFUNC}(B(\mathcal{I}'))$ for jobs $\mathcal{J}_i$, thus proving (LHS) being resilient to slicing.

To proceed our argument in the slotted time model, we need to define the discrete version of both sides. Define

dFUNC, a discrete version of fFUNC as follows.

$$\begin{aligned}\texttt{dFUNC}(B') &= \Delta\sum_{t\in\mathcal{T}}\sum_{j\in B'(t)} d_j\Delta k(\Delta N_{>d_j}^{B'(t)})^{k-1}\\ &= \Delta^{k+1}\sum_{t\in\mathcal{T}}\sum_{j\in B'(t)} d_j k(N_{>d_j}^{B'(t)})^{k-1}\end{aligned}$$

It is easy to see that $\texttt{dFUNC}(B')$ goes arbitrarily close to $\texttt{fFUNC}(B)$ when $\Delta \to 0$. Define the discrete version $\texttt{dFLOW}^k$ for $\texttt{fFLOW}^k$ as follows.

$$\texttt{dFLOW}^k(B') = s'\Delta^{k+1}\sum_{t\in\mathcal{T}}\sum_{j\in B(t)} d_j k(\frac{f_j-t}{\Delta})^{k-1}$$

The discrete function $\texttt{dFLOW}^k$ scatters each job j's kth power of flow time over time. Job j contributes $s'\Delta^2 k(f_j - t)^{k-1}$ to $\texttt{dFLOW}^k$ at each unit time t; thus one can think of j being released at time f_j and having increasing contribution in the reverse time order, and being finished at time r_j. More concretely, it can be noted that each job j's contribution to $\texttt{dFLOW}^k$ is approximately its kth power of weighted flow time from the following: $\Delta' d_j(f_j - r_j)^k = \Delta'\int_{t=r_j}^{f_j} k(f_j - t)^{k-1}\mathrm{d}t \simeq s'\Delta^2\sum_{t\in\mathcal{T}}\sum_{j\in B(t)} k(f_j-t)^{k-1}$; here " $\simeq$ " holds only when $f_j - r_j$ is sufficiently big compared to Δ, but by noting that the number of such exceptional jobs are negligible, $\texttt{dFLOW}^k$ can be shown to converge to $\texttt{fFLOW}^k$ when $\Delta \to 0$.

To complete our analysis, it is sufficient to show the following on each unit time slot $[t, t+\Delta]$.

$$\sum_{j\in B'(t)} d_j(N_{>d_j}^{B'(t)})^{k-1} \le \sum_{j\in B'(t)} d_j(\frac{f_j-t}{\Delta})^{k-1}.$$

For simple notation, let $B'(t) = \{1, 2, 3, ..., u\}$, and $d_1 \ge d_2 \ge d_3 \ge ... \ge d_u$. Then $\sum_{j\in B'(t)} d_j(N_{>d_j}^{B'(t)})^{k-1} = \sum_{i\in[u]} d_u(u-1)^{k-1}$. Since only one job can be completed at each unit time, $\frac{f_j-t}{\Delta}$ is a distinct integer for all $j \in B'(t)$. It is easy to see that $\sum_{j\in B'(t)} d_j(\frac{f_j-t}{\Delta})^{k-1}$ has the minimum value when $\frac{f_j-t}{\Delta} = j$, completing the proof. □

Proof of [Lemma 5.2] For this lemma we need the following proposition,

PROPOSITION B.1. *For any $x_1, x_2, ..., x_n \ge 0$,*

$$(\sum_{i=1}^n x_i)^k = \sum_{i=1}^n k\int_{\tau=0}^{x_i}(\sum_{j>i}^n x_j + \tau)^{k-1}\mathrm{d}\tau.$$

Proof.

$$\begin{aligned}(RHS) &= \sum_{i=1}^n((\sum_{j\ge i}^n x_j)^k - (\sum_{j>i}^n x_j)^k)\\ &= \sum_{i=1}^n(\sum_{j\ge i}^n x_j)^k - \sum_{i=2}^n(\sum_{j\ge i}^n x_j)^k\\ &= (LHS)\end{aligned}$$

By Proposition B.1, we have

$$\begin{aligned}&v\Big(\sum_{d_i>v}(p_i^B(t)/s')\Big)^k\\ =\ & v\sum_{\substack{i\in B(t)\\ d_i>v}}\int_{\tau=0}^{p_i^B(t)/s'}\Big(k\sum_{\substack{j\in B(t)\\ d_j>\max(d_i,v)}}(p_j(t)/s')+\tau\Big)^{k-1}\mathrm{d}\tau\\ \le\ & \sum_{i\in B(t)} d_i\int_{\tau=0}^{p_i^B(t)/s'} k\Big((V_{>d_i}^{B(t)}/s')+\tau\Big)^{k-1}\mathrm{d}\tau\end{aligned}$$

By multiplying both sides by s'^k, we obtain the first inequality. The above inequality, with Lemma 5.1, immediately implies the second desired inequality. □

Proof of [Lemma 5.3] By considering whether $(t - r_i) \ge \tau$ or not, we have

$$\begin{aligned}&(LHS)\\ \le\ & 2^k\Big[\int_{t=0}^{\infty} d_{b(x,t)}(t-r_{b(x,t)})^k\mathrm{d}t + \int_{t=0}^{\infty} d_{b(x,t)}(p_{b(x,t)}^O(t))^k\mathrm{d}t\Big]\end{aligned}$$

The first term $\int_{t=0}^{\infty} d_{b(x,t)}(t-r_i)^k\mathrm{d}t$ is easily upper bounded by OPT_x from the definition of kth power of fractional flow time. The second term $\int_{t=0}^{\infty} d_{b(x,t)}(p_{b(x,t)}^O(t))^k\mathrm{d}t$ can be bounded also by OPT_x by observing that each job i can contribute at most $\int_{\tau=0}^{p_{ix}} d_{ix}(p_{ix}-\tau)^k\mathrm{d}\tau$, the minimum value of kth power of fractional flow time for job i be completed by any schedule. □

Proof of [Lemma 5.4] By considering whether $(t - r_i) \ge \tau$ or not, we have

$$\begin{aligned}&(LHS)\\ \le\ & 2^{k-1}\Big[\int_{t=0}^{\infty}\sum_{i\in O_x(t)} d_i p_i^O(t)k(t-r_i)^{k-1}\mathrm{d}t\\ &\quad + \int_{t=0}^{\infty}\sum_{i\in O_x(t)} d_{ix}\int_{\tau=0}^{p_i^O(t)} k(\tau)^{k-1}\mathrm{d}\tau\mathrm{d}t\Big]\end{aligned}$$

By the definition of kth power of fractional flow time and Corollary 5.2, the lemma easily follows. □

Proof of [Lemma 5.5] Consider any fixed time t. We can assume that all jobs in $A(t)$ and $O(t)$ have infinitesimal size, since both sides are resilient to slicing. We partition $O_x(t)$ into $I(t)$ and $I'(t)$ depending on whether a job i satisfies $V_{>d_i}^{A(t)} \le \frac{k}{\epsilon}V_{>d_i}^{O(t)}$ or not. For the set $I(t)$, by Corollary 5.2, we have

$$\int_{t=0}^{\infty} \sum_{i\in I(t)} d_i p_i^O(t)(V_{>d_i}^{A(t)})^{k-1}\mathrm{d}t$$
$$\leq \ (\frac{k}{\epsilon})^{k-1}\int_{t=0}^{\infty}\sum_{i\in I(t)} d_i \int_0^{p_i^O(t)} (V_{>d_i}^{O(t)}+\tau)^{k-1}\mathrm{d}\tau\mathrm{d}t$$
$$\leq \ \frac{1}{k}(\frac{k}{\epsilon})^{k-1}\mathrm{OPT}_x.$$

We now focus on $I'(t)$. We first show that there exists a family of disjoint sets $G_i(t) \subseteq A(t), \forall i \in I'(t)$ satisfying all the following conditions:

1. $\forall i \in I'(t), V^{G_i(t)} = \frac{1}{\epsilon}p_i^O(t)$
2. $\forall i \in I'(t), \forall j \in G_i(t), V_{>d_j}^{A(t)} \geq (1-\frac{1}{k})V_{>d_i}^{O(t)}$
3. $\forall i \in I'(t), \forall j \in G_i(t), d_j \geq d_i$.

The family can be constructed as follows. For simple notation, let $I'(t) = [u]$ and jobs are indexed in decreasing order of density, that is $d_1 \geq d_2 \geq ... \geq d_u$. We inductively define $G_1(t)$ to $G_u(t)$. To each group $G_i(t)$, we assign $\frac{1}{\epsilon}p_i^O(t)$ volume of jobs from $\{j \in A(t) \mid (1-\frac{1}{k})V_{>d_i}^{A(t)} \leq V_{>d_j}^{A(t)} \leq V_{>d_i}^{A(t)}\} \setminus (\bigcup_{1\leq i'<i} G_{i'}(t))$. This can be done because

$$V^{(\bigcup_{1\leq i'<i} G_{i'}(t))} + \frac{1}{\epsilon}p_i^O(t)$$
$$= \ \frac{1}{\epsilon}\sum_{i'\in[i]} p_i^O(t) \leq \frac{1}{\epsilon}V_{\geq d_i}^{O(t)} \leq \frac{1}{k}V_{>d_i}^{A(t)}.$$

The last inequality comes from the definition of $I'(t)$; here the infinitesimal size of $p_i^O(t)$ is ignored.

We are now ready to complete the proof. For each $i \in I'(t)$, the term $d_i p_i^O(t)(V_{>d_i}^{A(t)})^{k-1}$ in (LHS) is charged to the term in (RHS)

$$\sum_{j\in G_i(t)} d_j \int_{\tau=0}^{p_j^A(t)} (V_{>d_j}^{A(t)}+\tau)^{k-1}\mathrm{d}\tau$$
$$\geq \ d_i \frac{1}{\epsilon}p_i^O(t)((1-\frac{1}{k})V_{>d_i}^{A(t)})^{k-1} \geq \frac{d_i}{3\epsilon}p_i^O(t)(V_{>d_i}^{A(t)})^{k-1}$$

The first inequality holds because of the three conditions each group G_i satisfies. Hence we have,

$$\int_{t=0}^{\infty}\sum_{i\in I'(t)} d_i p_i^O(t)(V_{>d_i}^{A(t)})^{k-1}\mathrm{d}t$$
$$\leq \ 3\epsilon\int_{t=0}^{\infty}\sum_{i\in I'(t)}\sum_{j\in G_i(t)} d_j \int_{\tau=0}^{p_j^A(t)} (V_{>d_j}^{A(t)}+\tau)^{k-1}\mathrm{d}\tau\mathrm{d}t$$
$$\leq \ 3\epsilon\int_{t=0}^{\infty}\sum_{i\in A(t)} d_i \int_{\tau=0}^{p_i^A(t)} (V_{>d_i}^{A(t)}+\tau)^{k-1}\mathrm{d}\tau\mathrm{d}t$$

This completes the proof. □

Proof of [Lemma 5.6] We partition jobs in $A_x(t)$ into $I(t)$ and $I'(t)$; each job i in $A(t)$ satisfying $V_{>d_i}^{O(t)} \leq \frac{\epsilon^2}{k}V_{>d_i}^{A(t)}$ is in $I(t)$, otherwise it is in $I'(t)$. For the set $I(t)$, it is trivial to see that

$$\sum_{i\in I(t)} d_i p_i^A(t)(V_{>d_i}^{O(t)})^{k-1}$$
$$\leq \ (\frac{\epsilon^2}{k})^{k-1}\sum_{i\in A(t)} d_i \int_{\tau=0}^{p_i^A(t)} (V_{>d_i}^{A(t)}+\tau)^{k-1}\mathrm{d}\tau$$

For the other set $I'(t)$, we will show that

$$\sum_{i\in I'(t)} d_i p_i^A(t)(V_{>d_i}^{O(t)})^{k-1}$$
$$\leq \ \frac{3k^2}{\epsilon^2}\sum_{i\in O(t)} d_i \int_{\tau=0}^{p_i^O(t)} (V_{>d_i}^{O(t)}+\tau)^{k-1}\mathrm{d}\tau$$

The remaining proof is very similar to that of Lemma 5.5. As in the proof of Lemma 5.5, we can assume that jobs in $A(t)$ and $O(t)$ have infinitesimal size. Also similarly, we can find a family of disjoint sets $G_i(t) \subseteq O(t), i \in I'(t)$ such that

1. $\forall i \in I'(t), V^{G_i(t)} = \frac{\epsilon^2}{k^2}p_i^A(t)$.
2. $\forall i \in I'(t), \forall j \in G_i(t), V_{>d_j}^{O(t)} \geq (1-\frac{1}{k})V_{>d_i}^{O(t)}$.
3. $\forall i \in I'(t), \forall j \in G_i(t), d_j \geq d_i$.

This can be found as follows. For simple notation, let $I'(t) := [u]$ and jobs are indexed in decreasing order of density, that is $d_1 \geq d_2 \geq ... \geq d_u$. We inductively define $G_1(t)$ to $G_u(t)$. To each group $G_i(t)$, we assign $\frac{\epsilon^2}{k^2}p_i^A(t)$ volume of jobs from $\{j \in O(t) | (1-\frac{1}{k})V_{>d_i}^{O(t)} \leq V_{>d_j}^{O(t)} \leq V_{>d_i}^{O(t)}\} \setminus \bigcup_{i'\in[i-1]} G_{i'}(t)$. This can be done because

$$V^{(\bigcup_{i'\in[i-1]} G_i(t))} + \frac{\epsilon^2}{k^2}p_i^A(t)$$
$$= \ \frac{\epsilon^2}{k^2}\sum_{i'\in[i]} p_i^A(t) \leq \frac{\epsilon^2}{k^2}V_{>d_i}^{A(t)} \leq \frac{1}{k}V_{>d_i}^{O(t)}.$$

The last inequality is due to the definition of $I'(t)$ ignoring the infinitesimal size of $p_i^A(t)$.

We are now ready to complete our proof. For each $i \in I'(t)$, the term $p_i^A(t)(V_{>d_i}^{O(t)})^{k-1}$ in (LHS) is charged to

$$\sum_{j\in G_i(t)} d_j \int_{\tau=0}^{p_j^O(t)} (V_{>d_j}^{O(t)}+\tau)^{k-1}\mathrm{d}\tau$$
$$\geq \ d_i V^{G_i(t)}((1-\frac{1}{k})V_{>d_i}^{O(t)})^{k-1} \geq \frac{\epsilon^2}{3k^2}d_i p_i^A(t)(V_{>d_i}^{O(t)})^{k-1}$$

Hence we have

$$
\begin{aligned}
& \int_{t=0}^{\infty} \sum_{i \in I'(t)} d_i p_i^A(t) (V_{>d_i}^{O(t)})^{k-1} \mathrm{dt} \\
\leq \; & \frac{3k^2}{\epsilon^2} \int_{t=0}^{\infty} \sum_{i \in I'(t)} \sum_{j \in G_i(t)} d_j \int_{\tau=0}^{p_i^O(t)} (V_{>d_i}^{O(t)} + \tau)^{k-1} \mathrm{d}\tau \mathrm{dt} \\
\leq \; & \frac{3k^2}{\epsilon^2} \int_{t=0}^{\infty} \sum_{i \in O_x(t)} d_i \int_{\tau=0}^{p_i^O(t)} (V_{>d_i}^{O(t)} + \tau)^{k-1} \mathrm{d}\tau \mathrm{dt} \\
\leq \; & \frac{3k}{\epsilon^2} \mathrm{OPT}_x
\end{aligned}
$$

The last inequality is due to Corollary 5.2. This completes the proof. □