Online Scalable Scheduling for the ℓ_k-norms of Flow Time Without Conservation of Work

Jeff Edmonds* Sungjin Im† Benjamin Moseley‡

Abstract

We address the scheduling model of arbitrary speed-up curves and the broadcast scheduling model. The former occurs when jobs are scheduled in a multi-core system or on a cloud of machines. Here jobs can be sped up when given more processors or machines. However, the parallelizability of the jobs may vary and the algorithm is required to be oblivious of the parallelizability of a job. The latter model is natural in wireless and LAN networks where requests (or jobs) can be simultaneously satisfied together. Both settings are similar in that two schedules can do different amounts of work to satisfy all the jobs. We focus on optimizing the ℓ_k- norms of flow time. Recently, Gupta et al. [24] gave a $(k+\epsilon)$-speed $O(1)$-competitive algorithm for the ℓ_k norms of flow time in both scheduling settings for fixed k. Inspired by this work, we give the first analysis of a scalable algorithm, i.e. $(1+\epsilon)$-speed $O(1)$-competitive, for all ℓ_k-norms of flow time in both settings for fixed k and $0 < \epsilon \leq 1$. Both problems have a strong lower bound without resource augmentation, so this is the best result that can be shown in the worst case setting up to a constant factor in the competitive ratio.

1 Introduction

Scheduling jobs which arrive online is a fundamental problem. In the most basic setting there are n jobs which arrive over time. Each job i has a fixed processing time p_i. This is the amount of time the scheduler must spend on job i to complete the job. In the *online* setting the arrival time a_i of job i is the first time the scheduler is aware of the job. The system must decide the order in which jobs should be scheduled. The goal of the scheduler is to optimize the quality of service the clients receive. The most well studied quality of service metric in scheduling theory is average (total) flow time[1]. The flow time of a schedule is $\sum_i (C_i - a_i)$ where C_i is the time the scheduler satisfies job i. Notice that the total flow time is equal to the total time all jobs must wait to be satisfied.

It is well known that in the standard single machine scheduling setting the algorithm Shortest-Remaining-Processing-Time (SRPT) is an optimal algorithm for total flow time. Now consider a more complicated scheduling setting where there are m machines (processors) and jobs can be assigned to one or more processors simultaneously. That is, jobs can be scheduled on processors in parallel. We will adopt a standard parallel computing model [15, 19, 17, 30, 10] that first appeared in [16]. This model is known as the arbitrary speed-up curves model. Here, each job i consists of a collection of phases that must be processed in a sequential order. According to the phase the job is in, the job may be considerably sped up when assigned to multiple processors, may not be sped up at all or some variation thereof. This accurately captures the practical scheduling setting where a phase of a job may require a lot of computation which can be considerably sped up when assigned to multiple machines. However, another phase of a job may not require much computation, but rather consists of a series of Input/Output operations which cannot be sped up by being given more computational power. To further capture the practical setting, we require that our algorithm be *non-clairvoyant*. That is, the algorithm is completely unaware of the processing time of the jobs or their parallelizability. Scheduling jobs with varying parallelizability has been a central focus of recent research [15, 19, 10, 24].

It is well known that no non-clairvoyant algorithm can be $O(1)$-competitive for average flow time, even the standard single machine model. Due to this strong lower bound, we will be assuming a resource augmentation model [27]. In this model the algorithm is given s-speed and is compared to an 1-speed optimal solution. We say that an algorithm is s-speed c-competitive if, for all job sequences, the algorithm running processors at speed s has an objective within c times the optimal solution for the objective running processors at speed 1. An algorithm that is $(1+\epsilon)$-speed $O(1)$-competitive for $0 < \epsilon \leq 1$

*York University, Canada. jeff@cs.yorku.ca. Supported in part by NSERC Canada.

†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 grant CNS-0721899.

[1] Flow time is also known as response time or waiting time

is said to be *scalable*. This is because the algorithm is $O(1)$-competitive when given the minimum possible extra resources over the adversary. For problems with strong lower bounds without resource augmentation, this is the best result that can be shown in the worst case setting. The first positive result for average flow time in the speed up curves setting was shown about a decade ago [15]. It was shown that that EQUI (round-robin) is $(2+\epsilon)$-speed $O(1)$-competitive. Recently, [19] gave an elegant potential function analysis to show that an algorithm called LAPS is scalable.

Notice that since we have assumed jobs can have varying degrees of parallelizability, different schedules can do different amounts of work to satisfy the same set of jobs. A similar situation arises in a different scheduling model known as broadcast scheduling. The broadcast model is most natural in a wireless or LAN network. Here there is a server where pages of information are stored. Clients send requests to the server for a page of information. The sever must satisfy all of the requests. More than one client may send a request for the same page. The server can satisfy multiple requests by a single broadcast since the clients are listening in on a multicast channel. Along with practical interest, this scheduling model has received a considerable amount of attention over the last decade in both the offline and online settings in scheduling theory [7, 2, 1, 8, 25, 3, 5]. Work has also been done in stochastic and queuing theory literature [14, 13, 33, 34]. In the broadcast setting, a scheduler can satisfy multiple requests for the same page by a single broadcast. Like the arbitrary speed-up curves setting, different schedules can do different amounts of work. That is, there is not conservation of work between two schedules. This makes scheduling problems considerably more difficult, especially in the online setting. The difficulty is that it is hard to relate the amount of work done by an online algorithm to that by the optimal solution.

Average flow time in the broadcast setting has received a significant amount of attention recently. It was first studied in the offline setting using non-trivial linear programming techniques with resource augmentation [28, 21, 22, 23]. Later, NP-hardness was shown [20, 11]. The best result known offline with resource augmentation is a $(1+\epsilon)$-speed $O(1)$-approximation [3]. Without resource augmentation the best result known is a $O(\log^2 n/\log\log n)$-approximation [4]. It is still open whether this problem admits a $O(1)$-approximation.

It was shown that without resource augmentation any online deterministic algorithm is $\Omega(n)$-competitive [28]. The natural algorithm Longest-Wait-First was shown to be 6-speed $O(1)$-competitive [18]. The analysis of this algorithm has been recently improved to $(3.4+\epsilon)$-speed $O(1)$-competitive [12]. It was a long standing open problem whether there exists a scalable online algorithm for average flow time in broadcast scheduling. This question was recently answered in the affirmative [26, 5]. In [17] a reduction was given from the broadcast setting to the arbitrary speed-up curves model. The reduction translates any s-speed c-competitive algorithm for average flow time for arbitrary speed-up curves into a $2s$-speed c-competitive algorithm for the same objective in broadcast scheduling. The reduction was recently improved to output a $s+\epsilon$-speed $\frac{c}{\epsilon}$-competitive algorithm where $\epsilon > 0$ [5], which implies that a broadcast version of LAPS is a scalable algorithm for average flow time in broadcast scheduling.

Minimizing the total flow time is a natural objective; unfortunately, in practice algorithms which optimize the flow time are not implemented due to a lack of fairness. Indeed, one can easily construct examples where an optimal algorithm for minimizing the total flow time may starve individual jobs of processing power for an arbitrary amount of time. Implementing a fair algorithm is one of the highest priorities in most systems [32]. A popular and practical method to enforce the fairness of a schedule is to optimize the ℓ_k-norm of flow time for some fixed $k > 1$. That is minimizing $\sqrt[k]{\sum_i (C_i - a_i)^k}$. In practice, it is usually the case that $k \in [2, 3]$. Notice that average flow time is equivalent to the ℓ_1 norm of flow time. By optimizing the ℓ_k norm of flow time, the algorithm minimizes the variance of the jobs, thus ensuring fairness while reducing the flow time. Although a schedule that optimizes the ℓ_k norm of flow time for $k > 1$ may have a larger average flowtime over a schedule that optimizes the ℓ_1 norm, the variance of flow time will be reduced. This will make the system more predictable on average for the individual jobs, which is generally more desirable [32, 31].

In an influential paper of Bansal and Pruhs it was shown that any online (clairvoyant) deterministic algorithm is $n^{\Omega(1)}$-competitive for the ℓ_k norms of flow time without resource augmentation in the standard single machine scheduling setting where $k > 1$ [6]. This contrasts with the fact that SRPT is optimal in the ℓ_1 norm in this setting. The same authors showed that SRPT is a scalable algorithm for the ℓ_k norms of flow time on a single machine in the standard scheduling setting [6]. They also showed that the non-clairvoyant algorithm Shortest-Elapsed-Time-First (SETF) is scalable. In the broadcast model Chekuri et al. showed that a variation of LWF is $O(k)$-speed $O(k)$-competitive for the ℓ_k-norms of flow time. Later, Gupta et al. [24] introduced two interesting algorithms that generalize LAPS and the broadcast version of LAPS shown in [5]. These algorithms were shown to be $(k+\epsilon)$-speed $O(k)$ competitive for fixed $\epsilon > 0$ for the ℓ_k-norms of flow time in the broadcast setting and arbitrary speed-up curves setting, respectively.

Results: We focus on the ℓ_k norms of flow time in the broadcast model and the arbitrary speed up curve model. In this paper we consider the algorithms of [24]. We show that these algorithms are $(1+\epsilon)$-speed $O(1)$-competitive algorithm for the ℓ_k norm of flow time for both scheduling settings where k is any fixed integer and ϵ is sufficiently small. Given the strong lower bounds for the two problems, the algorithms introduced are essentially the best positive result that can be shown in the worst case setting up to a constant factor in the competitive ratio for fixed k and ϵ. Specifically, we show the following theorem for the broadcast setting.

THEOREM 1.1. *There is an algorithm that is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^4})$-competitive for the ℓ_k of flow time in the broadcast setting where $k \geq 1$ and $0 < \epsilon < 1$.*

Notice that the competitive ratio is uniform for all k. Our competitive ratio for the ℓ_k norm is within a $O(\frac{1}{\epsilon})$ factor of the best known algorithm for the ℓ_1 norm of flow time in broadcast scheduling for any $k \geq 1$ [5]. In the arbitrary speed up curve model, we prove the following theorem.

THEOREM 1.2. *The algorithm* WLAPS *is $(1+12\epsilon)$-speed $O(\frac{k}{\epsilon^{2k+1}})$-competitive for the ℓ_k-norm objective for each fixed k where $0 < \epsilon \leq \frac{1}{24k}$.*

Here our competitive ratio grows with k. There is a $\Omega(\log n)$ lower bound on any algorithm with $O(1)$-speed in the arbitrary speed up curves setting for the ℓ_∞-norm, which suggests that an algorithm's competitive ratio may need to grow with k [29].

Techniques: To show that an algorithm is $O(1)$ competitive in scheduling theory, it suffices to show that at any time the increase in the algorithm's objective function is within a constant of the increase in the optimal solution's objective function. This is called a local argument. For instance, this was used to show that SETF is a scalable algorithm for the ℓ_k norms of flow time on a single machine in the standard setting. However, this property does not hold in some scheduling problems. It was shown in [28] that no algorithm can be locally competitive with the optimal solution for the ℓ_1 norm in broadcast scheduling. This was due to the fact that an adversary may do a considerably less amount of work as compared to the algorithm to satisfy all the requests.

To avoid a local argument we use a potential function analysis. This has recently become popular in scheduling theory [15, 19, 24, 9]. In this work we introduce an interesting potential function. Our potential function takes insights from [19, 24, 9, 5]. The potential function of [24] is most closely related to our potential function. The potential function is designed to approximate, at any time, our algorithm's future cost minus the adversary's future cost. Thus, our potential function tries to relate the algorithm's current status to the optimal solution's current status.

We will show the competitiveness of our algorithms as follows. Let $\Phi(t)$ denote our potential function. The potential function will be designed so that $\Phi(0) = \Phi(\infty) = 0$. Let $\frac{\mathrm{d}A(t)}{\mathrm{dt}}$ (resp. $\frac{\mathrm{d}O(t)}{\mathrm{dt}}$) be the increase in the algorithm's (respectively OPT's) objective function at time t. Let $\frac{\mathrm{d}\Phi(t)}{\mathrm{dt}}$ be the change in $\Phi(t)$. We will show that $\frac{\mathrm{d}A(t)}{\mathrm{dt}} + \frac{\mathrm{d}\Phi(t)}{\mathrm{dt}} \leq c\frac{\mathrm{d}O(t)}{\mathrm{dt}}$ at all times t where $c > 0$ is a constant. Knowing that $\Phi(0) = \Phi(\infty) = 0$, this implies that $A = \int_0^\infty [\frac{\mathrm{d}A(t)}{\mathrm{dt}}]\mathrm{dt} = \int_0^\infty [\frac{\mathrm{d}A(t)}{\mathrm{dt}} + \frac{\mathrm{d}\Phi(t)}{\mathrm{dt}}]\mathrm{dt} \leq \int_0^\infty c\frac{\mathrm{d}O(t)}{\mathrm{dt}}\mathrm{dt} = c\mathrm{OPT}$. This will complete our analysis.

2 Arbitrary Speed Up Curves

We will be assuming a machine augmentation model [27]. In this model, our algorithm will be given sm processors and is compared to an optimal solution that is given m processors. It is well known that speed augmentation is as powerful as machine augmentation, thus our results translate into the speed augmentation model.

Formally, a problem instance consists of a collection of jobs $\{1, 2, \ldots, n\}$. Each job i has an arrival time a_i and this is the time the scheduler is first aware of the job. The job i consists of a sequence of phases $\langle i^1, i^2, \ldots, i^{\pi_i}\rangle$. A single phase π is represented by tuple $\langle w_i^\pi, \Gamma_i^\pi\rangle$. Here w_i^π is a real number that represents the amount of work in the phase and Γ_i^π is the speedup function for the phase. The function $\Gamma_i^\pi(p)$ maps a nonnegative real number to a nonnegative real number. The value of $\Gamma_i^\pi(p)$ is the rate work is processed for phase π of job i when the job is given p processors running at speed 1. If these processors are running at speed s then phase π gets processed at a rate of $s\Gamma_i^\pi(p)$. We assume that the function Γ_i^π is arbitrary, yet nondecreasing and sublinear (Brent's theorem). That is, a job cannot be processed slower by being given more processors and a job does cannot be processed at a more efficient rate by being given more processors.

A feasible schedule specifies at each time how many processors are assigned to each job. At any time, the total number of processors assigned to jobs can be at most sm, the number of processors. A job can be assigned to a non-integral number of processors. Given a schedule $\mathcal{S}_s$ with sm processors, let $\mathcal{S}(i,t)$ processors be assigned to job i at time t. Say that at time t_π phase π is started for job i and this phase is completed at time $t_{\pi+1}$ then $\int_{t_\pi}^{t_{\pi+1}} \Gamma_i^\pi(\mathcal{S}_s(i,t))\mathrm{dt} = w_i^\pi$. The time $t_{\pi+1}$ is also the time that phase $\pi+1$ is started. The completion time C_i of job i occurs when the final phase of the job is completed.

For the ℓ_k-norm objective, the total cost is $\left(\sum_{i\in[n]}(C_i - a_i)^k\right)^{1/k}$. It is useful to note that minimizing this objective function is equivalent to minimizing

$\sum_{i\in[n]}(C_i - a_i)^k$ after taking the ℓ_k norm objective function to the power k. We will call $(C_i - a_i)^k$ the kth power flow time of job i. For the rest of this paper, we will focus on minimizing this objective.

We assume that our algorithm is non-clairvoyant, in that the algorithm only knows when jobs arrive, how many processors it assigned to the job over time and when a job completes. Notice that the algorithm is unaware of the phases of a job, the speedup function of the phases or what phase a job is in. Although, this may seem to be too harsh on the algorithm, as stated we will be able to show that a non-clairvoyant algorithm can be $O(1)$-competitive with just a small amount of resource resource augmentation even under this very restrictive setting. Further, this accurately models the practical systems where it is unlikely that complete information is known on the parallelizability of jobs.

Following the lead of [15, 19, 24] we will focus only on restricted instances. Here each phase π of a job i is either fully parallelizable, that is $\Gamma_i^\pi(p) = p$ or the phase is sequential $\Gamma_i^\pi(p) = 1$. Notice that by definition of a sequential phase, a job can be processed even if no processors are assigned to the job. In [24] it was shown that any worst case instance is of this form. Intuitively, this restriction does not give any advantage to the algorithm since we have assumed the algorithm is non-clairvoyant.

2.1 Algorithm It is known that a single s speed processor is as powerful as a s unit speed processors. This is because a s speed processor can simulate s unit speed processors. Knowing this, we focus on restricted instances, scale the number of processors by m and assume that the optimal solution is given a single unit speed processor and the algorithm is given s unit speed processors. Let β and ϵ be constants such that $\beta = \epsilon^k$ and $\epsilon \leq 1/(24k)$. The speed our algorithm is given is then $s = 1 + 12\epsilon$.

The algorithm we consider is WLAPS, Weighted Latest Arrival Processor Sharing, which was introduced in [24]. The algorithm WLAPS takes a parameter β. Before defining WLAPS, some notation is introduced. For each job $i \in [n]$, let $w_i(t) = k(t - a_i)^{k-1}$. The value of $w_i(t)$ is the rate at which the k^{th} power flowtime of job i increases at time t. If job i is unsatisfied by WLAPS, $w_i(t)$ is the increase in WLAPS objective function due to job i. Let $N_a(t)$ denote the jobs which have been released yet are unsatisfied by WLAPS at time t. Let $w(t) = \sum_{i\in N_a(t)} w_i(t)$. Let $N'_a(t)$ denote the set of jobs in $N_a(t)$ that have the latest arrival times such that $\sum_{i\in N'_a} w_i(t) = \beta w(t)$.

The algorithm WLAPS at time t shares its processing power amongst the jobs in $N'_a(t)$ in proportion to their weights. For job $i \in N'_a(t)$, WLAPS processes job i at a rate of $x_i(t) := s\frac{w_i(t)}{\beta w(t)}$. Notice that the total processing power used by WLAPS at any time is at most s.

Say that at some time t there does not exist a set of latest arriving jobs whose total weight is exactly $\beta w(t)$. Then the definition of WLAPS needs to be modified. In this case, let $N'_a(t)$ be the minimal set of latest arriving jobs whose weight exceed $\beta w(t)$. The algorithm WLAPS only works on jobs in $N'_a(t)$. Let job j be the job which arrived the earliest in $N'_a(t)$. The amount of processing power WLAPS gives every job in $N'_a \setminus \{j\}$ is defined as before. The job j receives processing power $x_j(t) := s\cdot\frac{\beta w(t) - (\sum_{i\in N'_a(t)\setminus\{j\}} w_i(t))}{\beta w(t)}$. The processing power given to job j is proportional to the weight of j that overlaps the β fraction of the total weight. Throughout the analysis we will assume that there exists a set of latest arriving jobs whose total weight is exactly $\beta w(t)$. This assumption is not needed for the analysis. However, the analysis is more readable with the assumption.

2.2 Potential Function We assume WLOG that all jobs arrive at distinct times. Let $N_o(t)$ be the set of released, yet unsatisfied jobs in the optimal solutions schedule. Let $x_i(t)$ denote the amount of parallel work for job i which OPT has done but WLAPS has not at time t; if WLAPS have processed more parallel work for job i than OPT, then $x_i(t)$ is zero. An analogous quantity $y_i(t)$ is defined for sequential work of job i. Let σ_i be the total sequential work for job i. To analyze WLAPS we will use a potential function analysis. For this analysis we will define a potential function $\Phi(t)$. The potential function $\Phi(t)$ will not increase when a job arrives nor when a job is completed by OPT or WLAPS. Further, $\Phi(0) = \Phi(\infty) = 0$. During all times $[t, t + \mathrm{d}t]$ when no job arrives or is completed, it will be shown that $\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{WLAPS}(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Phi(t) \leq c\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{OPT}(t)$, where c is some constant. Here $\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{WLAPS}(t) = \sum_{i\in N_a(t)} w_i(t)$ and $\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{OPT}(t) = \sum_{i\in N_o(t)} w_i(t)$. If $\Phi(t)$ meets each of these conditions then WLAPS is c competitive. Our potential function $\Phi(t)$ is defined as follows:

$$\Phi(t) := \sum_{i\in N_a(t)} \Big(t - a_i + \frac{1}{\epsilon} \sum_{\substack{a_j \geq a_i \\ j \in N_a(t)}} x_j(t)\Big)^k + (\frac{2}{\epsilon})^{2k+1} \sum_{i\in N_a(t)} w_i(t)y_i(t)$$

2.3 Intuition Behind the Potential Function Let $\Phi_1(t)$ be the first term of $\Phi(t)$ and $\Phi_2(t)$ be the second term of $\Phi(t)$. The boundary conditions of our potential function are satisfied trivially. When job i arrives at time t, the potential function has no change since $t - a_i = 0$ and $x_i = 0$ on arrival. The optimal solution completing a job has no effect on the potential function. When al-

gorithm completes a job i the potential function can only decrease, since all terms are positive.

Before analyzing the change in Φ, we discuss high level intuition of the analysis. As stated, in [6] it was shown that SRPT and SJF are scalable algorithms when all jobs have only one phase which is fully parallelizable. To prove this, the authors used a local argument. They showed that at each time t, the sum of the age^{k-1} of the jobs that are still alive under the algorithm's schedule is at most a constant c times the corresponding value for those that are alive under the optimal schedule. When jobs can have a varying degree of parallelizability, this local property no longer holds. This is why we resort to a potential function based argument. When the algorithm's current costs are less than c times the optimal's, the algorithm saves some into a bank account so that when the algorithm's current costs are higher than this, he can pay for them by withdrawing these reserves. The potential function measures how much is currently in the bank. The proof must show that at each point in time, these costs balance, namely that $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq c\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$.

Consider a time t. The increase rate in our objective function at time t is $\sum_{i\in N_a(t)} w_i(t)$. Likewise the increase rate in the optimal solution's objective function is $\sum_{i\in N_o(t)} w_i(t)$. If the increase in OPT's objective is comparable to the increase in WLAPS objective, then we can charge the increase of WLAPS' objective directly to the optimal solution along with any increase in $\Phi(t)$. This is where the definition of the algorithm is crucially used, since WLAPS is defined by the ages of jobs, it will help relate the ages of WLAPS' unsatisfied jobs to OPT's unsatisfied jobs. However, if the two objectives are not comparable then the decrease in $\Phi(t)$ must be used to pay for the increase in WLAPS objective. Here there are two cases either most of the ages of jobs that are being processed by WLAPS are in a sequential phase or they are in a parallel phase.

First say that most of the ages of the jobs WLAPS are working on are in a sequential phase. In this case, each of the jobs in a sequential phase gets processed whether or not WLAPS devotes processing power to the jobs. Since all of these jobs are being processed at a fast rate, WLAPS will be completing enough work to show that WLAPS is drifting its queue towards the optimal solution's queue. This case is captured by the second term in the potential function, Φ_2. Intuitively, $w_i(t)y_i(t)$ is an approximation of the remaining cost job i will pay in the algorithm's objective function for sequential phases.

The second case is when our algorithm is processing mostly parallel work. Here, since the algorithm is processing jobs with more speed than the adversary is, via resource augmentation, we will again drift towards the adversary's queue. This case is captured by the first term in the potential function. This is, $\Phi_1(t)$ will decrease enough to pay for any increase in the algorithm's objective. Intuitively, we derive Φ_1 by observing that $(t - a_i + \sum_{a_j\geq a_i, j\in N_a(t)} x_j(t))^k$ is an approximation of remaining cost job i will pay in the algorithm's objective function for parallel phases.

2.4 Analysis For simple notation, let $W_i(t) := k\Big(t - a_i + \frac{1}{\epsilon}\sum_{a_j\geq a_i, j\in N_a(t)} x_j(t)\Big)^{k-1}$. We will study each of the changes in $\Phi(t)$ separately depending on where the change comes from. In the final analysis, we will aggregate all the changes.

OPT's processing: First we consider the change in $\Phi(t)$ when the optimal solution processes jobs which are in a parallel phase. Let job q be the job with the latest arrival time in $N_a(t)$. The largest increase in $\Phi_1(t)$ occurs when the optimal solution processes job q. Since the optimal solution has 1-speed, $x_q(t)$ increases at a rate of at most 1. Thus we have $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t) \leq \frac{1}{\epsilon}\sum_{i\in N_a(t)} k\Big(t - a_i + \frac{1}{\epsilon}\sum_{a_j\geq a_i, j\in N_a(t)} x_j(t)\Big)^{k-1} = \frac{1}{\epsilon}\sum_{i\in N_a(t)} W_i(t)$.

We now address the change in $\Phi_2(t)$. The optimal solution processes each job i in $N_o(t)$ at a rate of 1 if i is in sequential phase, thus increasing $y_i(t)$ at a rate of at most 1. Recall that a job in a sequential phase is processed at a rate of 1 whether or not it receives any processing power. In the worst case, every job in $N_o(t)$ is in a sequential phase. Thus $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_2(t) \leq (\frac{2}{\epsilon})^{2k+1}\sum_{i\in N_o(t)} w_i(t)\mathrm{dt} \leq (\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$. Hence the total change rate of $\Phi(t)$ due to OPT's processing is $\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq \frac{1}{\epsilon}\sum_{i\in N_a(t)} W_i(t) + (\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$.

WLAPS's processing: We partition $N_a(t)$ into $\mathcal{S}(t)$ and $\mathcal{P}(t)$ such that $\mathcal{S}(t)$ contains all jobs in $N_a(t)$ that are in a sequential phase and $\mathcal{P}(t)$ contains all jobs in $N_a(t)$ that are in a parallel phase under the schedule of WLAPS at time t. Let $\mathcal{P}'(t) := N'_a(t) \cap \mathcal{P}(t)$ and $\mathcal{S}'(t) = N'_a(t) \cap \mathcal{S}(t)$. Consider any job $j \in \mathcal{P}'(t) \setminus N_o(t)$. Since OPT has completed job j, the variable $x_j(t)$ is just the remaining amount of parallel work for job i for WLAPS to process. Therefore, $x_j(t)$ decreases at a rate of $-s\frac{w_i(t)}{\beta w(t)}$ by the definition of how WLAPS distributes its s processors at each time. Note that this change occurs in $\sum_{a_j\geq a_i, j\in N_a(t)} x_j(t)$ for any job $i \in N_a(t) \setminus N'_a(t)$. This is because all jobs in $N'_a(t)$ (the jobs WLAPS chooses to process) have arrived no later than any job in $N_a(t) \setminus N'_a(t)$. Hence,

$$\text{(2.1)}\qquad \frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t) \;\le\; -\frac{1}{\epsilon}\left(\sum_{j\in\mathcal{P}'(t)\setminus N_o(t)} s\frac{w_j(t)}{\beta w(t)}\right)\cdot\left(\sum_{i\in N_a(t)\setminus N'_a(t)} W_i(t)\right)$$

For the final analysis we need to obtain an upper bound on (2.1). The following propositions and lemmas will be useful tools. The two following easy propositions were shown in [24].

PROPOSITION 2.1. *For any job i, $\sum_{a_j\ge a_i, j\in N_a(t)} x_j(t) \le t - a_i$.*

This proposition easily follows since $\sum_{a_j\ge a_i, j\in N_a(t)} x_j(t)$ is the amount of parallel work OPT is ahead of WLAPS for jobs released after time a_i. Since OPT has 1 processor, this is at most $t - a_i$. The following proposition is trivial since $y_i(t)$ can grow at a rate of at most 1 at each time.

PROPOSITION 2.2. *For any job i, $y_i(t) \le t - a_i$.*

The following proposition is easily obtained by applying proposition 2.1.

PROPOSITION 2.3. $\sum_{i\in N_a(t)} W_i(t)$

$$\le \sum_{i\in N_a(t)} (1+\frac{1}{\epsilon})^{k-1} w_i(t) = (1+\frac{1}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t).$$

By Proposition 2.1, we can bound a part of (2.1),

$$\begin{aligned}
&\sum_{i\in N_a(t)\setminus N'_a(t)} W_i(t) = \sum_{i\in N_a(t)} W_i(t) - \sum_{i\in N'_a(t)} W_i(t)\\
&\ge \sum_{i\in N_a(t)} W_i(t) - (1+\frac{1}{\epsilon})^{k-1}\sum_{i\in N'_a(t)} w_i(t)\\
&= \sum_{i\in N_a(t)} W_i(t) - \beta(1+\frac{1}{\epsilon})^{k-1}\sum_{i\in N_a(t)} w_i(t)\\
&\ge \Big[1-\beta(1+\frac{1}{\epsilon})^{k-1}\Big]\sum_{i\in N_a(t)} W_i(t)
\end{aligned}$$

(2.2)

The last equality is due to the definition of $N'_a(t)$, and the last inequality is due to $W_i(t) \ge w_i(t)$ for all $i \in N_a(t)$. And by the definition of $\mathcal{P}'(t)$, we have

$$\sum_{j\in\mathcal{P}'(t)\setminus N_o(t)} \frac{w_j(t)}{\beta w(t)}$$

$$\begin{aligned}
&\ge \frac{1}{\beta w(t)}\Big[\sum_{j\in N'_a(t)} w_j(t) - \sum_{j\in\mathcal{S}(t)} w_j(t)\\
&\qquad - \sum_{j\in N_o(t)} w_j(t)\Big]\\
&= 1 - \frac{1}{\beta w(t)}\Big[\sum_{j\in\mathcal{S}(t)} w_j(t) + \sum_{j\in N_o(t)} w_j(t)\Big]
\end{aligned}$$

(2.3)

From (2.1), (2.2) and (2.3), we have

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t) \le -s\frac{1}{\epsilon}\Big(1-\beta(1+\frac{1}{\epsilon})^{k-1}\Big)\\
&\Big(1-\frac{1}{\beta w(t)}\Big[\sum_{j\in\mathcal{S}(t)} w_j(t) + \sum_{j\in N_o(t)} w_j(t)\Big]\Big)\sum_{i\in N_a(t)} W_i(t)
\end{aligned}$$

(2.4)

For any job $i \in \mathcal{S}(t)\setminus N_o(t)$, $y_i(t)$ decreases at a rate of 1 by definition of sequential work. Thus, $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_2(t) \le -(\frac{2}{\epsilon})^{2k+1}\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t)$.

Time Elapse: We now address the change in $\Phi(t)$ due to the change in time. The change in $\Phi_1(t)$ is $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t) = \sum_{i\in N_a(t)} W_i(t)$. The change in $\Phi_2(t)$ is, $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_2(t) = (\frac{2}{\epsilon})^{2k+1}\sum_{i\in N_a(t)} k(k-1)(t-a_i)^{k-2}y_i(t)$.

Our goal is now to bound $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_2(t)$ by $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)$. To this end, we partition jobs in $N_a(t)$ into 'old' jobs $\mathcal{O}(t)$ and 'young' jobs $\mathcal{Y}(t)$. Recall that σ_i is the total sequential work for job i. A job $i \in N_a(t)$ is in $\mathcal{Y}(t)$ if $(t-a_i) \le k(\frac{2}{\epsilon})^{2k+1}\sigma_i$; otherwise, the job is in $\mathcal{O}(t)$. The increase in $\Phi_2(t)$ due to jobs in $\mathcal{Y}(t)$ will be charged directly to the optimal solution's cost in the following lemma. This idea is similar to that given in [24] and the proof can be found in the appendix.

LEMMA 2.1. $\int_0^\infty \sum_{i\in\mathcal{Y}(t)} k(k-1)(t-a_i)^{k-2}y_i(t)\mathrm{dt} \le k^{k+1}(\frac{2}{\epsilon})^{k(2k+1)}\mathrm{OPT}$.

The change in $\Phi_2(t)$ due to old jobs is at most $(\frac{2}{\epsilon})^{2k+1}\sum_{i\in\mathcal{O}(t)} k(k-1)(t-a_i)^{k-2}\sigma_i(t) \le \sum_{i\in\mathcal{O}(t)} k(t-a_i)^{k-1} \le \sum_{i\in N_a(t)} k(t-a_i)^{k-1}$, by definition of old jobs. Thus after excluding the young jobs, the total increase in $\Phi(t)$ due to the change in time is, $\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \le 2\sum_{i\in N_a(t)} W_i(t)$.

2.5 Completing the Analysis: For the final analysis, we add the upper bound on the change for each of the cases we studied in the previous section. Let $\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)$ denote the change (rate) that is obtained from $\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t)$ by removing the increase due to time elapse for the young jobs. We will show that $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t) \le$

$2(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$. Then we will have

$$\begin{aligned}
\mathrm{WLAPS} &= \int_0^\infty \Big(\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)\Big)\mathrm{dt} \\
&= \int_0^\infty \Big(\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t)\Big)\mathrm{dt} \\
&\qquad\qquad \text{[Since } \Phi(0) = \Phi(\infty) = 0\text{]} \\
&\leq \int_0^\infty \Big(\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)\Big)\mathrm{dt} \\
&\qquad\qquad + k^{k+1}(\frac{2}{\epsilon})^{k(2k+1)}\mathrm{OPT} \\
&\leq \int_0^\infty \Big(2(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)\Big)\mathrm{dt} \\
&\qquad\qquad + k^{k+1}(\frac{2}{\epsilon})^{k(2k+1)}\mathrm{OPT} \\
&\leq 3k^{k+1}(\frac{2}{\epsilon})^{k(2k+1)}\mathrm{OPT}
\end{aligned}$$

The first inequality comes from Lemma 2.1, which gives an upper bound on the total increase due to time elapse over all times for the young jobs. Recall that we have been considering the objective of minimizing the sum of the kth power flowtime. Since we are actually interested in ℓ_k-norms we take the outer kth root, which proves Theorem 1.2.

It now remains to show $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t) \leq 2(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$. By adding the upper bounds we obtained in the previous section, we have

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t) \leq \\
(2.5)\ &\Big(3+\frac{1}{\epsilon}\Big)\sum_{i\in N_a(t)} W_i(t) \\
(2.6)\ &-s\frac{1}{\epsilon}\Big(1-\beta(1+\frac{1}{\epsilon})^{k-1}\Big)\Big(1-\frac{1}{\beta w(t)}\Big[\sum_{j\in\mathcal{S}(t)} w_j(t) \\
&\quad + \sum_{j\in N_o(t)} w_j(t)\Big]\Big)\sum_{i\in N_a(t)} W_i(t) \\
(2.7)\ &-(\frac{2}{\epsilon})^{2k+1}\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t) \\
(2.8)\ &+(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)
\end{aligned}$$

We remind the reader that (2.6) and (2.7) come from the change due to WLAPS's processing jobs in a parallel phase and jobs in a serial phase, respectively. Recall that $\beta = \epsilon^k$ and $s \geq 1+12\epsilon$, where $0 < \epsilon \leq \frac{1}{24k}$. We consider three cases.

Case (a): $\sum_{i\in N_o(t)} w_i(t) \geq \epsilon\beta\sum_{i\in N_a(t)} w_i(t)$. This is the easiest case where OPT has jobs whose total weight is comparable to that of the jobs in WLAPS's queue. In this case, by Proposition 2.3 and simple algebra, $(2.5)+(2.8) \leq \frac{4}{\epsilon}(\frac{2}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+(2.8) \leq 2(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$.

Case (b): $\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t) \geq \epsilon\beta\sum_{i\in N_a(t)} w_i(t)$. In this case, the decrease due to WLAPS's processing jobs in a sequential phase will offset other positive terms. Again, by Proposition 2.3 and an easy calculation, $(2.5)+(2.7) \leq \frac{4}{\epsilon}(\frac{2}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) - \epsilon\beta(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) \leq 0$. And clearly, $(2.8) \leq 2(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$.

Case (c): Neither case (a) nor case (b). Then we have $\sum_{i\in N_o(t)} w_i(t) + \sum_{i\in\mathcal{S}(t)} w_i(t) \leq 3\epsilon\beta\sum_{i\in N_a(t)} w_i(t)$. This is the case where most (in terms of weights) of the jobs WLAPS are processing are in a parallel phase. By simple algebra, $(2.5)+(2.6) \leq \frac{1+3\epsilon}{\epsilon}\sum_{i\in N_a(t)} W_i(t) - \frac{1}{\epsilon}s(1-\epsilon(1+\frac{\epsilon}{k})^{k-1})(1-3\epsilon)\sum_{i\in N_a(t)} W_i(t) \leq 0$. In all cases the desired inequality holds, and this completes the analysis.

3 Broadcast Scheduling

The broadcast setting is defined as follows. There are n pages of information stored at a server. Page p has size σ_p. Over time requests arrive for specific pages. Each request is for one page and there can be multiple requests for the same page. The arrival time of a request r_i is a_i. The page r_i requests will be defined when needed. Each page p is divided into unit pieces $(1,p),(2,p),\ldots(\sigma_p,p)$. One unit piece can be broadcasted by the server in a unit time slot. A request r_i for page p is satisfied if it receives each of the integer pieces of page p in *sequential* order; here some pieces of other pages can be transmitted between the pieces of page p. Notice that multiple requests can be satisfied simultaneously. The time a request is satisfied or completed by our algorithm is denoted as C_i. The flow time of a request r_i is $(C_i - a_i)$. The goal of the scheduler is to minimize the ℓ_k norm of flow time $\sqrt[k]{\sum_i (C_i-a_i)^k}$.

3.1 Fractional ℓ_k norm Flow Time To bound the ℓ_k norm of flow time of a schedule, we will focus on bounding the total kth power flow time of a schedule, $\sum_i (C_i - a_i)^k$. Here we have dropped the outer kth root. To do this, we focus on bounding the kth power flow time of a *fractional* schedule. In a fractional schedule the server is allowed to broadcast an infinitesimal amount of data for more than one page in a single time slot. Let $y_p(t)$ denote the rate at which page p is broadcasted at time t. In the fractional model, the finish time of a request is different than in the integral model. The finish time of request r_i for page p is now defined to be the first time t that $\int_{a_i}^t y_p(t)\mathrm{dt} = \sigma_p$. Notice that in this setting a request need not receive the unit pieces of page p sequentially. Bansal et al. [5] showed a reduction from the integral broadcast setting to the fractional broadcast setting. This reduction implies that a s-speed c-competitive algo-

rithm for the fraction broadcast setting can be converted into an algorithm that is $s(1+\epsilon')$-speed $O(\frac{c}{\epsilon'})$ for the integral broadcast setting where $\epsilon > 0$. See [24] for details. Due to this reduction, we will focus on the fractional setting.

3.2 Algorithm Let $\beta = \epsilon^{2k-1} > 0$ and $0 < \epsilon < \frac{1}{10}$ be constants. We assume that our algorithm is given $s = 1 + 10\epsilon$ speed. Let $N_a(t)$ and $N_o(t)$ denote the set of unsatisfied requests at time t under the algorithm's schedule and OPT schedule, respectively. We will, for simple notation, sometimes use $i \in$ to denote $r_i \in$. For a request r_i, let $w_i(t) = k(t-a_i)^{k-1}$ be the rate at which the kth power flow time of request r_i increases at time t. This will also be called the weight of r_i at time t. Let $w(t) = \sum_{i \in N_a(t)} w_i(t)$. Let $N'_a(t) \subseteq N_a(t)$ denote the set of the earliest arriving requests in $N_a(t)$ whose total weight adds up to $\beta w(t)$. As in the arbitrary speed up curves setting, we use a simplifying assumption that there is a set of earliest arriving requests whose weights sum up to be exactly $\beta w(t)$.

We denote the algorithm as BWLAPS, that is, the broadcast version of LAPS. The algorithm BWLAPS devotes its processing power to the requests in $N'_a(t)$. A request $r_i \in N'_a(t)$ is processed at a rate of $x_i(t) = s\frac{w_i(t)}{\beta w(t)}$. Notice that the total speed used is at most s. By processing a request r_i for page p, we mean that the page p is broadcasted. More formally, let $S_p(t)$ be the set of requests for page p in $N_a(t)$. Then page p is broadcasted at a rate of $y_p(t) = \sum_{i \in S_p(t)} x_i(t)$.

3.3 Potential Function Let $y^*_p(t)\mathrm{d}t$ be the rate at which that OPT broadcasts page p at time t. Let $\mathtt{Opt}(t_1, t_2, p) = \int_{t_1}^{t_2} y^*_p(t)\mathrm{d}t$. For a request r_i let $\mathtt{On}(t_1, t_2, r_i) = \int_{t_1}^{t_2} x_i(t)\mathrm{d}t$. Say that request r_i is for page p, then we define $z_i(t)$ to be $\frac{\mathtt{On}(t,\infty,r_i)\cdot\mathtt{Opt}(a_i,t,p)}{\sigma_p}$. Our potential function is now defined as,

$$\Phi(t) := \sum_{i \in N_a(t)} (t - a_i + \frac{1}{\epsilon} \sum_{\substack{r_j \in N_a(t) \\ a_j \geq a_i}} z_j(t))^k.$$

Notice that our potential function uses the variable z_i, rater than x_i as in the speed up curves setting. This is used because there could be multiple requests for the same page. It is designed to mimic the properties of x_i in the speed-up curves setting: A key property of x_i was that the increase in $\sum_i x_i(t)$ due to OPT's processing $\mathrm{d}t$ amount of work is at most $\mathrm{d}t$; this is not true anymore in the broadcast setting. However it can be shown that the total increase in $\sum_i z_i(t)$ is also at most $\mathrm{d}t$; see the proof of Lemma 3.1. Further, if OPT has satisfied r_i (for page p) and BWLAPS broadcasts p at a rate of s for $\mathrm{d}t$ time then the decrease in z_i is at least $s\mathrm{d}t$.

The boundary conditions of our potential function are satisfied trivially. When job i arrives at time t, the potential function has no change since $t - a_i = 0$ and $z_i(t) = 0$ on arrival. The optimal solution completing a job has no effect on the potential function. When algorithm completes a job i the potential function can only decrease, since all terms are positive. For the remaining analysis, to simplify notation, we let $W_i(t) = k(t - a_i + \frac{1}{\epsilon}\sum_{r_j \in N_a(t), a_j \geq a_i} z_j(t))^{k-1}$.

3.4 Continuous Change in $\Phi(t)$ It remains to consider the change in Φ during a time interval $[t, t + \mathrm{d}t]$ when no jobs arrive or are completed. Let $\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{BWLAPS}(t) = \sum_{r_i \in N_a(t)} k(t-a_i)^{k-1}\mathrm{d}t$ and let $\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{OPT}(t) = \sum_{r_i \in N_o(t)} k(t-a_i)^{k-1}\mathrm{d}t$. The values of $\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{BWLAPS}(t)$ and $\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{OPT}(t)$ are the increase rate of the kth power flow time of BWLAPS' schedule and OPT's, respectively. Our goal is to show that $\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{BWLAPS}(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Phi(t) \leq \frac{1}{\beta}(\frac{2}{\epsilon})^{k+1}\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{OPT}(t)$ at all times t.

Before we proceed, we introduce some simple lemma and proposition. The following easy lemma was shown in [5]; for completeness, we put its proof in Appendix A.

LEMMA 3.1. *For any job $i \in N_a(t)$ it is the case that $\sum_{r_j \in N_a(t), a_j \geq a_i} z_j(t) \leq (t - a_i)$.*

The following proposition easily follows from the above lemma.

PROPOSITION 3.1. $\sum_{i \in N_a(t)} W_i(t) \leq \sum_{i \in N_a(t)} (1 + \frac{1}{\epsilon})^{k-1} w_i(t) \leq (\frac{2}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{BWLAPS}(t)$.

We address each of the possible changes in $\Phi(t)$. First it is easy to see that the change in $\Phi(t)$ due to time is $\sum_{i \in N_a(t)} W_i(t)$. We now address the change in $\Phi(t)$ due to OPT's processing. Let page p be the page OPT broadcasts at time t and let $S_p(t)$ be the requests in $N_a(t)$ for page p. First we observe that $\sum_{r_j \in S_p(t), a_j \geq a_i} \mathtt{On}(t, \infty, r_j) \leq \sigma_p$ due to the fact that the algorithm needs to broadcast page p for at most a σ_p amount of time to satisfy each request for page p in $N_a(t)$. Also, we have that $\Delta\mathtt{Opt}(a_j, t, p) \leq \mathrm{d}t$ because the optimal solution has only 1 speed. Using these two facts, for any request $r_i \in N_a(t)$ we can bound the change in $\sum_{r_j \in N_a(t), a_j \geq a_i} z_j(t)$ to be at most $\sum_{r_j \in S_p(t), a_j \geq a_i} \frac{\mathtt{On}(t,\infty,r_j)}{\sigma_p} \cdot \Delta\mathtt{Opt}(a_j, t, p) \leq \mathrm{d}t$. This allows us to upper bound the change in $\Phi(t)$ due to OPT's processing to be $\frac{1}{\epsilon}\sum_{i \in N_a(t)} W_i(t)$.

Now it can easily be seen that the change in $\Phi(t)$ due to our algorithm's processing is non-positive. For the remainder of the analysis, we will consider two cases.

Case (a): $\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{BWLAPS}(t) \leq \frac{1}{\beta\epsilon}\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{OPT}(t)$. In this case we can charge $\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{BWLAPS}(t)$ and $\frac{\mathrm{d}}{\mathrm{d}t}\Phi(t)$ directly

to the optimal solution. Indeed, by Proposition 3.1 and simple algebra, $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq \sum_{i\in N_a(t)} w_i(t) + (1 + \frac{1}{\epsilon})\sum_{i\in N_a(t)} W_i(t) \leq 2(\frac{2}{\epsilon})^k \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) \leq \frac{1}{\beta}(\frac{2}{\epsilon})^{k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$

Case (b): $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) > \frac{1}{\beta\epsilon}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$. For this case the decrease in $\Phi(t)$ due to our algorithm's processing will play a crucial role to offset other increases. By a similar reasoning as done in the analysis for the arbitrary speed up curve setting, the total change (rate) due to our algorithm's processing is at most,

$$(3.9) \qquad \frac{1}{\epsilon}\Big(\sum_{i\in N'_a(t)\setminus N_o(t)} \frac{\mathrm{d}}{\mathrm{dt}} z_i(t)\Big) \sum_{i\in N_a(t)\setminus N'_a(t)} W_i(t)$$

Note that for any $r_i \in N'_a(t) \setminus N_o(t)$, it is the case that $\frac{\mathrm{d}}{\mathrm{dt}} z_i(t) \leq \frac{\mathrm{d}}{\mathrm{dt}}\mathtt{On}(t, \infty, r_i) = -\frac{w_i(t)}{\beta w(t)}$ by definition of BWLAPS. Using the assumption that $\frac{1}{\beta\epsilon}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) < \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t)$, we observe that

$$\sum_{i\in N'_a(t)\setminus N_o(t)} \frac{\mathrm{d}}{\mathrm{dt}} z_i(t) = -\sum_{i\in N'_a(t)\setminus N_o(t)} \frac{s w_i(t)}{\beta w(t)}$$
$$\leq -\sum_{i\in N'_a(t)} \frac{s w_i(t)}{\beta w(t)} + \sum_{i\in N_o(t)} \frac{s w_i(t)}{\beta w(t)} \leq -s(1-\epsilon).$$

By simple algebra and Proposition 3.1 we have that

$$\sum_{i\in N_a(t)\setminus N'_a(t)} W_i(t) = \sum_{i\in N_a(t)} W_i(t) - \sum_{i\in N'_a(t)} W_i(t)$$
$$\geq \sum_{i\in N_a(t)} W_i(t) - (1+\frac{1}{\epsilon})^{k-1} \sum_{i\in N'_a(t)} w_i(t)$$
$$\geq (1 - \beta(1+\frac{1}{\epsilon})^{k-1}) \sum_{i\in N_a(t)} W_i(t)$$
$$\text{[Since } \sum_{i\in N'_a(t)} w_i(t) = \beta \sum_{i\in N_a(t)} w_i(t)$$
$$\leq \beta \sum_{i\in N_a(t)} W_i(t)]$$

Thus we obtain $(3.9) \leq -\frac{s(1-\epsilon)}{\epsilon}(1 - \beta(1 + \frac{1}{\epsilon})^{k-1})\sum_{i\in N_a(t)} W_i(t)$.

We are now ready to complete this case. Recall that $0 < \epsilon < 1/10$, $s = 1 + 10\epsilon$, and $\beta = \epsilon^{2k-1}$. Then by combining the change due to the changes in time, OPT's processing and the algorithm's processing we have $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq (2 + \frac{1}{\epsilon} - \frac{s(1-\epsilon)}{\epsilon}(1 - \beta(1 + \frac{1}{\epsilon})^{k-1})\sum_{i\in N_a(t)} W_i(t) \leq 0$.

Thus in both cases (a) and (b), the desired inequality $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq \frac{1}{\beta}(\frac{2}{\epsilon})^{k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$ and we are now ready to complete our analysis.

$$\mathrm{BWLAPS} = \int_0^\infty \Big(\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t)\Big)\mathrm{dt}$$
$$\leq \int_0^\infty \frac{1}{\beta}(\frac{2}{\epsilon})^{k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)\mathrm{dt} = \frac{2^{k+1}}{\epsilon^{3k}}\mathrm{OPT}$$

By taking the outer kth root in the objective function and scaling ϵ and β we have the following theorem.

THEOREM 3.1. *The algorithm* BWLAPS *is* $(1+\epsilon)$*-speed* $O(\frac{1}{\epsilon^3})$*-competitive for the* ℓ_k *of flow time in the fractional broadcast setting where* $k \geq 1$ *and* $0 < \epsilon < 1$.

Using the reduction from the fractional broadcast setting to the integral setting, we have Theorem 1.1.

4 Conclusions

In this paper we have shown a scalable algorithm for the ℓ_k norms of flow time in broadcast scheduling for all $k \geq 1$. We also have given a scalable algorithm for the same objective in the speed-up curves setting for any fixed $k \geq 1$. Our competitive ratio grows with k for the speed up curves setting. We have preliminary evidence of a $\Omega(k \ln k)$ lower bound on algorithms that are given speed at most $1 + \epsilon$ where $\epsilon \leq 1$. Though there is an exponential gap between these upper and lower bound, it does indicate that the competitive ratio should grow with k. It would be interesting to tighten the gap between the upper and lower bounds. Our lower-bound will appear in a full version of this paper.

It is important to note that our algorithms depend on the speed ϵ. That is β depends on ϵ in WLAPS and BWLAPS. It would be interesting to show, in either setting, that a scalable algorithm must depend on ϵ to be $O(1)$-competitive for fixed k or to give a scalable algorithm that does not explicitly depend on ϵ.

References

[1] S. Acharya, M. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 2(6):50–60, Dec 1995.

[2] Demet Aksoy and Michael J. Franklin. "rxw: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Trans. Netw.*, 7(6):846–860, 1999.

[3] Nikhil Bansal, Moses Charikar, Sanjeev Khanna, and Joseph (Seffi) Naor. Approximating the average response time in broadcast scheduling. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 215–221, 2005.

[4] Nikhil Bansal, Don Coppersmith, and Maxim Sviridenko. Improved approximation algorithms for broadcast scheduling. *SIAM J. Comput.*, 38(3):1157–1174, 2008.

[5] Nikhil Bansal, Ravishankar Krishnaswamy, and Viswanath Nagarajan. Better scalable algorithms for broadcast scheduling. Technical report, 2009.

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

[7] Amotz Bar-Noy, Randeep Bhatia, Joseph (Seffi) Naor, and Baruch Schieber. Minimizing service and operation costs of periodic scheduling. *Math. Oper. Res.*, 27(3):518–544, 2002.

[8] Yair Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 558–559, 2000.

[9] 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.

[10] Ho-Leung Chan, Jeff Edmonds, and Kirk Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *SPAA*, pages 1–10, 2009.

[11] Jessica Chang, Thomas Erlebach, Renars Gailis, and Samir Khuller. Broadcast scheduling: algorithms and complexity. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 473–482. Society for Industrial and Applied Mathematics, 2008.

[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] R. K. Deb. Optimal control of bulk queues with compound poisson arrivals and batch service. *Opsearch.*, 21:227–245, 1984.

[14] R. K. Deb and R. F. Serfozo. Optimal control of batch service queues. *Adv. Appl. Prob.*, 5:340–361, 1973.

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

[16] Jeff Edmonds, Donald D. Chinn, Tim Brecht, and Xiaotie Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. *J. Scheduling*, 6(3):231–250, 2003.

[17] Jeff Edmonds and Kirk Pruhs. Multicast pull scheduling: When fairness is fine. *Algorithmica*, 36(3):315–330, 2003.

[18] Jeff Edmonds and Kirk Pruhs. A maiden analysis of longest wait first. *ACM Trans. Algorithms*, 1(1):14–32, 2005.

[19] 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.

[20] Thomas Erlebach and Alexander Hall. Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. *J. Scheduling*, 7(3):223–241, 2004.

[21] Rajiv Gandhi, Samir Khuller, Yoo-Ah Kim, and Yung-Chun (Justin) Wan. Algorithms for minimizing response time in broadcast scheduling. *Algorithmica*, 38(4):597–608, 2004.

[22] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding in bipartite graphs. In *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 323–332, 2002.

[23] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, 2006.

[24] 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.

[25] Alexander Hall and Hanjo Täubig. Comparing push- and pull-based broadcasting. or: Would "microsoft watches" profit from a transmitter?. In *Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA 03)*, pages 148–164, 2003.

[26] 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.

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

[28] Bala Kalyanasundaram, Kirk Pruhs, and Mahendran Velauthapillai. Scheduling broadcasts in wireless networks. *Journal of Scheduling*, 4(6):339–354, 2000.

[29] Kirk Pruhs, Julien Robert, and Nicolas Schabanel. Minimizing maximum flowtime of jobs with arbitrary parallelizability. Manuscript, 2010.

[30] Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 491–500, 2008.

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

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

[33] J. Weiss. Optimal control of batch service queues with nonlinear waiting costs. *Modeling and Simulation*, 10:305–309, 1979.

[34] J. Weiss and S. Pliska. Optimal policies for batch service queueing systems. *Opsearch*, 19(1):12–22, 1982.

A Omitted Proofs

Proof of [Lemma 2.1]

$$
\begin{aligned}
(LHS) \quad &\leq \quad \sum_{i\in[n]} \int_{a_i}^{a_i+k(\frac{2}{\epsilon})^{2k+1}\sigma_i} k(k-1)(t-a_i)^{k-2} y_i(t)\mathrm{dt} \\
&\qquad\qquad \text{[From definition of } \mathcal{Y}(t)\text{]} \\
&\leq \quad \sum_{i\in[n]} \int_{a_i}^{a_i+k(\frac{2}{\epsilon})^{2k+1}\sigma_i} k(k-1)(t-a_i)^{k-1}\mathrm{dt} \\
&\qquad\qquad \text{[By Proposition 2.2]}
\end{aligned}
$$

$$= \quad (k-1)\sum_{i\in[n]}(k(\frac{2}{\epsilon})^{2k+1})^k\sigma_i^k \leq k^{k+1}(\frac{2}{\epsilon})^{k(2k+1)}\text{OPT}$$

[Since OPT $\geq \sum_{i\in[n]}\sigma_i^k$]

□

Proof of [Lemma 3.1] Let $S_p(t)$ denote the set of requests in $N_a(t)$ that are for page p. First notice that $\sum_{r_j\in S_p(t)} \texttt{On}(t,\infty,r_j) \leq \sigma_p$ for all times t and pages p. Indeed, each request $r_j \in S_p(t)$ is satisfied once page p has been broadcasted by an amount of σ_p. By definition of $\texttt{On}$ the claim follows. For any fixed request $r_i \in N_a(t)$ for page p we have that,

$$\begin{aligned}
& \sum_{r_j\in N_a(t), a_j\geq a_i} z_j(t) \\
= \quad & \sum_p \sum_{r_j\in S_p(t), a_j\geq a_i} \frac{\texttt{On}(t,\infty,r_j)\cdot\texttt{Opt}(a_j,t,p)}{\sigma_p} \\
\leq \quad & \sum_p \sum_{r_j\in S_p(t), a_j\geq a_i} \frac{\texttt{On}(t,\infty,r_j)\cdot\texttt{Opt}(a_i,t,p)}{\sigma_p} \\
= \quad & \sum_p \texttt{Opt}(a_i,t,p)\Big[\frac{1}{\sigma_p}\sum_{r_j\in S_p(t), a_j\geq a_i}\texttt{On}(t,\infty,r_j)\Big] \\
\leq \quad & \sum_p \texttt{Opt}(a_i,t,p) \ \text{ [Since } \textstyle\sum_{r_j\in S_p(t)}\texttt{On}(t,\infty,r_j)\leq\sigma_p\text{]} \\
\leq \quad & (t-a_i) \quad \text{[Since OPT has 1-speed]}
\end{aligned}$$

□