Scheduling Jobs with Varying Parallelizability to Reduce Variance

Anupam Gupta[*]
Computer Science Dept.
Carnegie Mellon University
Pittsburgh PA 15213
anupamg@cs.cmu.edu

Sungjin Im[†]
Dept. of Computer Science.
University of Illinois
Urbana, IL 61801.
im3@illinois.edu

Ravishankar Krishnaswamy[*]
Computer Science Dept.
Carnegie Mellon University
Pittsburgh PA 15213
ravishan@cs.cmu.edu

Benjamin Moseley[‡]
Dept. of Computer Science.
University of Illinois
Urbana, IL 61801.
bmosele2@illinois.edu

Kirk Pruhs[§]
Computer Science Dept.
University of Pittsburgh
Pittsburgh PA 15260
kirk@cs.pitt.edu

ABSTRACT

We give a $(2+\epsilon)$-speed $O(1)$-competitive algorithm for scheduling jobs with arbitrary speed-up curves for the ℓ_2 norm of flow. We give a similar result for the broadcast setting with varying page sizes.

Categories and Subject Descriptors

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

General Terms

Algorithms, Theory

Keywords

Online Algorithms, Scheduling Algorithms

1. INTRODUCTION

We consider scheduling dynamically arriving jobs that have varying degrees of parallelizability (that is, some jobs may be sped up considerably when simultaneously run on multiple processors, while other jobs may speed up by very little) on a multiprocessor system. The most obvious settings where this problem arises is scheduling multi-threaded processes on a chip with multiple cores/processors, and scheduling multi-process applications in a server farm. We adopt the following general model of parallelizability, which was apparently first introduced in [6] and later used in [5, 7, 8, 16, 15, 4]: we have m identical fixed speed processors. Each job i arrives at time a_i, and consists of a sequence of phases. Each phase needs to finish some amount of work, and has a speedup function that specifies the rate at which work is processed in that particular phase (as a function of the number of processors assigned to the job). The speedup functions have to be nondecreasing (a job doesn't run slower if it is given more processors), and sublinear (a job satisfies Brent's Theorem: increasing the number of processors doesn't increase the efficiency of computation), but the functions are unconstrained otherwise.

The scheduler needs an *assignment policy* to determine how many processors are allocated to each job at each point in time. In order to be implementable in a real system, we require that this policy be *online*, since the scheduler will not in general know about jobs arriving in the future. This policy also ideally should be *nonclairvoyant*, since a scheduler usually does not know the size/work of a job when the job is released, nor the degree to which that job is parallelizable. So a nonclairvoyant algorithm only knows when jobs have been released and which have finished in the past, and how many processors have been allocated to each job at each point of time in the past.

Given a schedule, the most natural quality of service measure for a job is its response time (also called the flow or waiting time) $F_i := C_i - a_i$, which is the length of time between when the job i is released at time a_i and when it completes at time C_i. To get a quality of service measure for the entire schedule, one must combine the quality of service measures of the individual jobs. The most commonly used way to do this is to take the average, or the sum (the ℓ_1 norm), of the flow times of the individual jobs. But to quote from Silberschatz and Galvin's classic text *Operating Systems Concepts* [17], "A system with reasonable and predictable response time may be considered more desirable than a system that is faster on the average, but is highly

[*]Supported in part by NSF awards CCF-0448095 and CCF-0729022, and an Alfred P. Sloan Fellowship.
[†]Partially supported by NSF grants CCF-0728782, CNS-0721899, and Samsung Fellowship.
[‡]Partially supported by NSF grant CNS-0721899.
[§]Supported in part by NSF grants CNS-0325353, IIS-0534531, and CCF-0830558, and an IBM Faculty Award.

variable." and " ... for interactive systems, it is more important to minimize the variance in the response time than it is to minimize the average response time." Hence, in many settings, the ℓ_2 norm (the square root of the sum of the squares) of job flow times may be a better quality of service measure of schedules than the ℓ_1 norm of job flow times.

A Simple Instance: *As a well known concrete example of difference between the ℓ_1 and ℓ_2 norms, consider a single-machine instance where two jobs are released at time 0, and one job is released at each integer time $1, 2, \ldots, n$. All jobs are identical, and the system takes one unit of time to finish each job. When the objective is to minimize the ℓ_1 norm of the flow time, one can see that every non-idling schedule is optimal. In particular, the schedule that has flow time 1 for all jobs except for one of the jobs released at time 0 (which will have flow time n) is also optimal. This however is not optimal for the ℓ_2 norm. Scheduling jobs in order of their release time results in the optimal schedule where all jobs have flow time at most 2. Thus a schedule that is good under the ℓ_2 norm reduces the variance of the job flow times relative to an optimal schedule for the ℓ_1 norm.*

Our Results: To recap, we address the problem of designing an online nonclairvoyant assignment policy that is competitive for the objective of the ℓ_2 norm of the flow time for jobs with arbitrary speed-up curves. Our main result is the following:

Theorem 1.1 *We give a nonclairvoyant assignment policy that is $(2+\epsilon)$-speed $O(1)$-competitive for minimizing the ℓ_2 norm of the flow time for jobs with arbitrary speedup curves.*

Note that all our results use *resource augmentation*: An algorithm A is s-speed c-competitive if, for every instance, the algorithm A with processors running at speed s guarantees a schedule whose objective is within a factor of c of the value of the optimal objective with unit speed processors. Intuitively, an s-speed $O(1)$-competitive algorithm guarantees that the schedule can handle a load up to a $\frac{1}{s}$ fraction of the load that an optimal offline scheduler can handle. Thus an $(1+\epsilon)$-speed $O(1)$-competitive algorithm is said to be *scalable* since it can handle almost the same load as optimal. For further elaboration, see [14, 13]. It is well known that no online algorithm can be $O(1)$-competitive for ℓ_k norms of flow time without resource augmentation [2].

1.1 Context, Intuition, and Our Techniques

To get a feel for the problem, let us consider two special cases: Consider first the case that all jobs are *fully parallelizable* (i.e., increasing the number of processors assigned to a job by a factor f reduces the time required by a factor of f), which is essentially equivalent to having a single processor. In the initial paper popularizing resource augmentation analysis [10], the algorithm Shortest Elapsed Time First (SETF) which shares the processors evenly among all jobs that have been processed the least (which necessarily are the later arriving jobs) was shown to be scalable for the ℓ_1 norm of flow. This was generalized by [2] to show that SETF is scalable for all ℓ_k norms of flow for $1 \leq k < \infty$. So intuitively, in the case of parallel work, the "right" algorithm is independent of the norm, equivalently the "right" algorithm for the ℓ_1 norm extends to all ℓ_k norms!

Now consider the more general case of jobs with arbitrary speed-up curves, but only for the ℓ_1 norm. [5] showed that the assignment policy EQUI that shares the processors evenly among all active jobs is $(2+\epsilon)$-speed $O(1)$-competitive for average flow time. Subsequently, [8] introduced the algorithm Late Arrival Processor Sharing (LAPS), which shares the processors evenly among the latest arriving constant fraction of the jobs, and showed that LAPS is scalable for average flow time. The intuition behind LAPS was to mimic SETF, by giving more processors to later arriving jobs, but to spread the processing power more evenly in case that the latest arriving jobs are sequential (their processing rate does not increase even if they are assigned more processors). In this special case, the "right" strategy for arbitrary speed-curves is basically the same as the "right" strategy for parallel work.

Given these two special cases, everyone's (that we are aware of) initial reaction to the question of the design of a policy for the ℓ_2 norm of flow for jobs with arbitrary speed-up curves, was that LAPS should be scalable. The intuition is understandable: [10] showed that for parallel work, favoring the most recent jobs is the right strategy to minimize the ℓ_1 norm of flow. Then [2] showed that considering general ℓ_k norms didn't change the "right" strategy, and [8] showed that allowing jobs with arbitrary speed-up curves didn't drastically change the right strategy. Intuitively it seems that considering arbitrary ℓ_k norms and arbitrary speed-up curves *together* shouldn't drastically change the right strategy. However, this intuition is misleading: we show in Section 3 that LAPS is not $O(1)$-speed $O(1)$-competitive for the ℓ_k norm of flow when $k \geq 2$.

1.1.1 *The Mistake in the Intuition, and Our Algorithm*

Considering why LAPS fails to be $O(1)$-competitive for the ℓ_2 norm, even with faster processors, offers insight into how to design such an algorithm. Just as the total flow time $\sum_i F_i$ is the integral over time of the number of jobs unfinished at that time, the sum of the squares of the flow times $\sum_i F_i^2$ is proportional to the integral over time, of the *ages* of the jobs at that time. The key observation in [2] was that if SETF, with a $(1+\epsilon)$-speed processor, has unfinished jobs of a particular age, then any schedule on a unit speed processor must have a comparable number of unfinished jobs that are at least as old: this observation implies the competitiveness of SETF for all ℓ_k norms rather directly. The lower bound instance in Section 3 shows that for jobs with arbitrary speed-up curves, LAPS may have devoted too much processing to sequential jobs in the past, and hence it may have many more old jobs than is necessary. And as LAPS focuses on new jobs, these old jobs will remain unfinished, driving up the ℓ_2 norm of flow for LAPS relative to optimal. This lower bound instance, and the simple instance of the stream of unit work jobs, suggests that the online algorithm must give a greater share of the processing power to older jobs for ℓ_2 norm of flow time for jobs with arbitrary speed-up curves.

Our algorithm addresses this by distributing the processors proportional to the age of the jobs, which is the rate at which that job is currently driving up the ℓ_2 norm of flow. Combining this intuition with an idea used in [8] to focus only on a fraction of the recent jobs, we design an algorithm WLAPS that is $(2+\epsilon)$-speed $O(1)$-competitive for the ℓ_2 norm of flow. Essentially, WLAPS distributes the processors to the latest arriving constant fraction of jobs. However, the proportion of resources a job gets is related to the age of the job.

1.1.2 Other Results: Broadcast Scheduling, and Other Norms

To explain broadcast scheduling, consider a web server serving static content on a broadcast channel. If the web server has multiple unsatisfied requests for the same file, broadcasting that file only once simultaneously satisfies all the users who issued these requests. [7] observed that broadcast scheduling in the ℓ_1 norm can be viewed as a special case of scheduling jobs with arbitrary speed-up curves in the ℓ_1 norm, in which all jobs consist of a sequential phase followed by a parallel phase. Indeed, the results of [7, 1] showed how to convert any s-speed c-competitive nonclairvoyant algorithm for scheduling jobs with arbitrary speedup curves into a $(1+\epsilon)s$-speed c-competitive nonclairvoyant algorithm for broadcast scheduling. Combining this reduction with the analysis of LAPS in [8], one obtains a scalable algorithm for broadcast scheduling for the ℓ_1 norm of flows. Unfortunately, this reduction from broadcast scheduling given in [7, 1] does not seem to work for the ℓ_2 norm of flow. However, in Section 4, we directly show that the broadcast version of WLAPS is indeed $(2+\epsilon)$-speed $O(1)$-competitive for the ℓ_2 norm of flow for broadcast scheduling.

Our techniques give algorithms for other ℓ_k norms as well. In particular, we can give nonclairvoyant algorithms that are $(k+\epsilon)$-speed $O(k^2)$-competitive for the objective of minimizing the ℓ_k norm of the flow times. We should remark that improving our results for larger values of k would be interesting, though mostly of academic interest, since as a practical matter, the interesting values of k are those in $[1,3] \cup \{\infty\}$.

1.2 Related Results

Consider first the case that all the work is fully parallelizable and and the ℓ_1 norm. It is well known that the online clairvoyant algorithm Shortest Remaining Processing Time is optimal. The competitive ratio of any deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$, and the competitive ratio of every randomized algorithm against an oblivious adversary is $\Omega(\log n)$ [12]. A randomized version of the Multi-Level Feedback Queue algorithm is $O(\log n)$-competitive [11, 3].

As for broadcast scheduling, a recent result of Im and Moseley [9] gives a scalable algorithm for the problem of minimizing the ℓ_1 norm of the requests, when all pages are of unit size. Subsequently [1] shows a scalable algorithm, inspired by LAPS, for the general problem with arbitrary page sizes.

Chan et al. [4] consider the problem of nonclairvoyant scheduling of jobs with varying degrees of parallelizability on a multiprocessor, where each machine can be scaled at a different speed. They give a $O(\log m)$-competitive algorithm for the problem of minimizing the sum of average flow time and energy, where the power function varies as s^{α} for constant α, under some assumptions about the jobs.

1.3 Formal Problem Statement and Notation

We now formally define the problem and introduce notation required for our algorithm and analysis. An arbitrary problem instance consists of a collection of jobs $\mathcal{J} = \{J_1, \dots, J_n\}$ where job J_i has a *release/arrival time* of a_i and a sequence of phases $\langle J_i^1, J_i^2, \dots, J_i^{q_i} \rangle$. Each phase is an ordered pair $\langle w_i^q, \Gamma_i^q \rangle$, where w_i^q is a positive real number that denotes the amount of *work* in the phase and Γ_i^q is a function, called the *speedup function*, that maps a nonnegative real number to a nonnegative real number. The function $\Gamma_i^q(p)$ represents the rate at which work is processed for phase q of job i when the job is run on p processors running at speed 1. Henceforth, we may interchangeably use job i and job J_i when the context is clear.

A *feasible schedule* $\mathcal{S}_s$ for the job set $\mathcal{J}$ with n jobs with sm available processors (one may think of s as a parameter) specifies for each time, and for each job, a nonnegative real number specifying the number of processors assigned to the job. Notice that we allow a job to be scheduled on a non-integral number of machines, and this can be translated to an actual scheduling on machines by having schedules which are preemptive and migratory. Such an assignment would be feasible as long as $\sum_{i=1}^{n} \mathcal{S}_s(i,t) \le sm$ for all time instants t, where $\mathcal{S}_s(i,t)$ is the number of processors schedule $\mathcal{S}_s$ allocates to job i at time t. In words, at any time, the total number of processors allocated to the jobs must not exceed sm.

For such a schedule $\mathcal{S}_s$, suppose a job i begins its q^{th} phase at time t_q. Then, the completion time of this stage (which is also when the subsequent stage begins) is the unique time t_{q+1} such that $\int_{t_q}^{t_{q+1}} \Gamma_i^q(\mathcal{S}_s(i,t))dt = w_i^q$. The completion time C_i of the job is then the completion time of its final phase q_i.

In the ℓ_k norm objective, the total cost incurred by this solution can then be expressed as

$$\mathsf{cost}(\mathcal{S}_s) = \left(\sum_{i \in [n]} (C_i - a_i)^k \right)^{1/k}$$

Recall that a nonclairvoyant algorithm only knows when jobs have been released and finished in the past, and which jobs have been run on each processor each time in the past. In particular, for any phase q, the algorithm does not know the values of w_i^q, and the speedup function Γ_i^q. In fact, it is not even aware of the progression of a job from one phase to the next.

Notice that, by having a parameter s to alter the number of available processors, the notion of *resource augmentation* we have (implicitly) assumed here is that of machine augmentation and not speed augmentation. However, since an s speed processor is as powerful as s unit speed processors when preemption is allowed, our results would translate to the speed augmentation model as well. This enables us to make the following simplification: for ease of analysis, we scale the number of processors by a factor of m, and assume that the optimal solution has a single unit speed processor and the online algorithm has s unit speed processors.

2. NON-CLAIRVOYANT SCHEDULING WITH ARBITRARY SPEEDUP CURVES

2.1 Restricted Instances are Sufficient

As by now is standard, we can show that we only need to focus on restricted instances where every job is composed of either *fully parallelizable* phases or *completely sequential* phases. A phase is said to be completely parallelizable if $\Gamma_i^q(p) = p$ for all p, and completely sequential if $\Gamma_i^q(p) = 1$

for all values of p. That is, sequential phases progress at the same rate regardless of the number of processors allocated.

To show this, we perform the following reduction from an arbitrary instance $\mathcal{I}$ of the problem to such a restricted instance $\mathcal{I}'$ with the following properties holding true: *(i)* the schedule produced by the non-clairvoyant algorithm *remains the same* for both instances $\mathcal{I}$ and $\mathcal{I}'$, and *(ii)* the cost of the optimal offline solution for the instance $\mathcal{I}'$ is at most the cost of an optimal offline solution for the first instance $\mathcal{I}$. This would ensure that if our algorithm is α-competitive on instance $\mathcal{I}'$, then it has a competitive ratio of at most α on instance $\mathcal{I}$ as well.

Let `NCAlg` denote any non-clairvoyant algorithm. Our reduction works in the following fashion: For each job i that is released in $\mathcal{I}$, we release the job i' in $\mathcal{I}'$ at the same time a_i. Now consider an infinitesimally small interval $[t, t+dt)$, and let `NCAlg` devote p_i^a processors towards j in this time interval. Also, let the job be in some phase with parallelizability Γ in this time interval. Therefore, the online algorithm effectively does a work of $w = \Gamma(p_i^a)dt$ for job i in time interval $[t, t+dt)$. Now, let us focus on the time interval $[t^*, t^*+dt^*)$ when the optimal solution works on this exact w amount of the job i (note that it could occur before or after t). Let the optimal solution devote p_i^o processors towards doing this work w. Notice that the definition of $[t^*, t^*+dt^*)$ and p_i^o imply that $\Gamma(p_i^o)dt^* = w = \Gamma(p_i^a)dt$, which in turn implies that

$$\frac{\Gamma(p_i^o)}{\Gamma(p_i^a)} = \frac{dt}{dt^*} \tag{1}$$

If $p_i^o \geq p_i^a$, then in the new instance $\mathcal{I}'$, we replace this w amount of work for job i with $w' = p_i^a dt$ amount of *fully parallelizable* work. Notice that by this change, when w amount of work was finished by the online algorithm in $\mathcal{I}$, an equivalent w' amount of work is done in $\mathcal{I}'$, and so the job progresses at the same rate for the online algorithm in either instance. Furthermore, since $p_i^o \geq p_i^a$, we have that

$$\frac{p_i^o}{p_i^a} \geq \frac{\Gamma(p_i^o)}{\Gamma(p_i^a)} = \frac{dt}{dt^*} \tag{2}$$

and therefore an optimal solution for $\mathcal{I}'$ can fit in the w' amount of fully parallelizable work at same time interval $[t^*, t^*+dt^*)$ when the optimal solution for $\mathcal{I}$ worked on the corresponding w amount of i. Here, the equation (2) follows from the sublinear nature of the speed-up function.

On the other hand, if $p_i^o < p_i^a$, then in our instance $\mathcal{I}'$, we replace this w amount of work for job i with $w' = dt$ amount of *fully sequential* work. Notice that by this change, when w amount of work was finished by the online algorithm in $\mathcal{I}$, an equivalent w' amount of work is done in $\mathcal{I}'$, and again the job progresses at the same rate for the online algorithm in either instance. Furthermore, since in this case $p_i^o < p_i^a$, we have that

$$1 \geq \frac{\Gamma(p_i^o)}{\Gamma(p_i^a)} = \frac{dt}{dt^*} \tag{3}$$

and therefore $dt^* \geq dt$. Therefore, an optimal solution for $\mathcal{I}'$ can fit in the $w' = dt$ amount of fully sequential work in same time interval $[t^*, t^*+dt^*)$ when the optimal solution for $\mathcal{I}$ worked on the corresponding w amount of i.

Hence, in either case, we see that the flow time of every job in the non-clairvoyant online algorithm is same for both instances, and the flow time in the optimal solution for $\mathcal{I}'$ is at most that for $\mathcal{I}$. Therefore, it is sufficient to design non-clairvoyant algorithms which are competitive against such extremal instances. Furthermore, since any phase of a job is either completely sequential or completely parallelizable, an algorithm working on s machines is equivalent to one working on a single machine with speed s. Hence, in the following section, we shall refer to s as the speed advantage the online algorithm has over the optimal offline adversary.

2.2 Non-clairvoyant Algorithm WLAPS

We first describe our non-clairvoyant preemptive algorithm WLAPS for Weighted Latest Arrival Processor Sharing. As can be deduced from its name, WLAPS is inspired by LAPS [8], a scalable algorithm for minimizing the total flow time of the jobs (i.e. when $k = 1$). Before we describe our algorithm, let us introduce some notation. First and foremost, we will assume that the algorithm WLAPS is given a speed-up of a factor of s. In other words, we can assume that WLAPS is given a s-speed processor while the optimal adversary is given only a unit-speed processor. Let β be a scaling parameter that determines the fraction of weight we consider at any instant of time. The speed-up s will depend on β, and we will fix this parameter later.

For each job $i \in [n]$, let us define its *weight* at time t to be $w_i(t) = k(t - a_i)^{k-1}$. Informally, $w_i(t)$ denotes the rate of increase of the k^{th} power of the flow time of job i at time t (which is also the incremental cost incurred by the algorithm due to job i being alive at time t). At any time t, let $N_a(t)$ denote the set of jobs that are alive in the queue of our algorithm, i.e. $N_a(t) := \{i \in [n] \mid a_i \leq t < C_i\}$, where C_i is the completion time of job i. Among the set of jobs $N_a(t)$, let $N'_a(t)$ denote the set of those jobs with the *latest arrival times* whose weights sum up to $\beta w(t)$, where $w(t) = \sum_{i \in N_a(t)} w_i(t)$.

It would be useful to observe that the objective function we are interested in is equivalent to (after raising the ℓ_k objective by a power of k) minimizing

$$\sum_{i \in [n]} (C_i - a_i)^k = \int_0^\infty \sum_{i \in N_a(t)} w_i(t) \mathsf{dt}$$

We are now ready to describe our algorithm: At any time t, the algorithm WLAPS simply distributes its processing power among the jobs in $N'_a(t)$, in proportion to their weights at time t. Let $x_i(t)$ denote the fraction of processing power job i receives at time t under the schedule of WLAPS. Then,

$$x_i(t) := s \cdot \frac{w_i(t)}{\beta w(t)}, \;\; \forall i \in N'_a(t)$$

Notice that the total processing power used at any time is exactly s. We remark that when $k = 1$ our algorithm WLAPS is exactly the same as LAPS, since the weights of all jobs are identically equal to 1.

A Simplifying Assumption: We assume that there exists a set of latest arriving jobs whose weights sum up to exactly $\beta w(t)$. Otherwise, a slight modification should be made to the algorithm. The set $N'_a(t)$ which WLAPS works on is now defined to be the minimal set of latest arriving jobs whose weights exceed $\beta w(t)$. Let j be the earliest arriving job in $N'_a(t)$. The amount of processing power that every job gets in $N'_a(t)$ except j stays the same. The job j receives a processing power of $x_j(t) := s \cdot \frac{\beta w(t) - (\sum_{i \in N'_a(t) \setminus \{j\}} w_i(t))}{\beta w(t)}$. In words, roughly speaking, the processing power the job j gets is proportional to its weight which "overlaps" the β fraction of weights. With this small elaboration, we can remove the assumption and the analysis easily follows. We however stick to the simplifying assumption to make our analysis more readable.

2.3 Analysis

Our analysis is based on a potential function argument, inspired by [8]. To formally describe the potential function, we need to introduce some more notation.

For any job i, let σ_i denote the total sequential work for job i and ρ_i denotes the total parallel work for job i. Recall (from Section 2.2) that for any job i and time t, $x_i(t)$ denotes the fraction of processing the algorithm WLAPS dedicates towards job i. Similarly, let $x_i^*(t)$ denote the fraction of processing the optimal offline schedule OPT allocates for i at time t. Without loss of generality, we can assume that $x_i^*(t) > 0$ only if job i is in parallel phase under the schedule by the adversary. Since we assumed that the total processing power for WLAPS is s and that for OPT is 1, it follows that $\sum_{i \in N'_a(t)} x_i(t) \leq s$ and that $\sum_{i \in N_o(t)} x_i^*(t) \leq 1$.

For any job $i \in [n]$, let $\texttt{On}(i, t_1, t_2)$ denote the *total amount of parallel work* for job i done by WLAPS during the time interval $[t_1, t_2]$. To quantify this formally, we need to define a variable $\mathbb{I}_i(t)$ which indicates whether job i is in a parallel phase for WLAPS at time t, (in which case $\mathbb{I}_i(t) = 1$) or in a sequential phase ($\mathbb{I}_i(t) = 0$). Then,

$$\texttt{On}(i, t_1, t_2) := \int_{t_1}^{t_2} x_i(t) \mathbb{I}_i(t) dt$$

Note that $\texttt{On}(i, t_1, t_2) \leq \rho_i$ for any job i and any time interval $[t_1, t_2]$.

Similarly, for each job $i \in [n]$, let $\texttt{Opt}(i, t_1, t_2)$ denote the total amount of parallel work for job i done by OPT during $[t_1, t_2]$. Since we had assumed that OPT works on a job only if it is in a parallel phase, we have that

$$\texttt{Opt}(i, t_1, t_2) := \int_{t_1}^{t_2} x_i^*(t) dt$$

We will then introduce the following variable that will be used in our potential function to keep track of parallel work for job i:

$$z_i(t) = \frac{\texttt{On}(i, t, \infty) \cdot \texttt{Opt}(i, a_i, t)}{\rho_i}$$

As for sequential work, let $y_i(t)$ denote how much the adversary OPT is ahead of WLAPS in the sequential work of job i at time t. If WLAPS is ahead of the adversary in the sequential work, $y_i(t)$ is set to zero.

Once again, recall that $N_a(t)$ denotes the set of jobs that are yet unfinished at time t by the algorithm WLAPS. Similarly, define $N_o(t)$ to denote the jobs alive under OPT's schedule.

Our potential function $\Phi(t)$ is defined as follows

$$\Phi(t) := \frac{1}{\beta} \sum_{i \in N_a(t)} z_i(t) \sum_{a_j \leq a_i, j \in N_a(t)} w_j(t) + \frac{8k}{\beta^3} \sum_{i \in N_a(t)} w_i(t) y_i(t)$$

For the remainder of this section, our goal is to show the following bound

$$\frac{d}{dt} A(t) + \frac{d}{dt} \Phi(t) \leq \frac{16k^3}{\beta^3} \frac{d}{dt} \mathrm{OPT}(t) \qquad (4)$$

Here $\frac{d}{dt} A(t) = \sum_{i \in N_a(t)} w_i(t)$ denotes the rate of increase of the objective function for WLAPS, and $\frac{d}{dt}\mathrm{OPT}(t)$ denotes the analogous quantity for the optimal schedule OPT. (some exceptional details should be taken care of). Since $\Phi(0) = \Phi(\infty) = 0$, this would suffice to analyze competitive ratio of WLAPS, by a simple amortized analysis (by integrating both sides of the above inequality from 0 to ∞).

In order to show equation (4), let $\Phi_1(t)$ and $\Phi_2(t)$ denote the first and the second term in $\Phi(t)$ respectively. For ease of analysis, we will separately investigate the change of $\Phi(t)$ for the following events, and then put the pieces together.

Change of the Potential Function: We will first consider time instants that induce discontinuities in Φ and show that it does not jump abruptly. It is easy to see that the only sources of discontinuity is when new jobs arrive, and when jobs are completed by WLAPS (in which case they leave the set $N_a(t)$). We consider these two situations now.

Job Arrival: We show that $\Delta\Phi = 0$. When a job i arrives at time t , $z_i(t) = y_i(t) = 0$ because the optimal solution has not had a chance to work on job i yet. It is easy to see that any new terms which appear are zero, since the jobs are indexed according to their arrival time. Thus, $\Delta\Phi = 0$.

Job Completion for WLAPS*:* When a job i is completed by the online algorithm, some terms may disappear from $\Phi(t)$. Since all terms are always non-negative, $\Delta\Phi(t) \leq 0$.

We now consider an infinitesimally small time interval $[t, t + dt)$ and show that the equation (4) holds. To show this, we individually consider the various changes that occur to Φ, and collect them all at the end.

Processing by OPT*:* The amount of parallel work for job $i \in N_a(t)$ done by the adversary in $[t, t + dt)$ is at most $x_i^*(t) dt$. Thus,

$$\begin{aligned}
\Delta\Phi_1(t) &\leq \frac{1}{\beta} \sum_{i \in N_a(t)} \frac{\texttt{On}(i, t, \infty) \cdot x_i^*(t) dt}{\rho_i} \sum_{a_j \leq a_i, j \in N_a(t)} w_j(t) \\
&\leq \frac{1}{\beta} dt \sum_{i \in N_a(t)} x_i^*(t) \sum_{a_j \leq a_i, j \in N_a(t)} w_j(t) \\
&\leq \frac{1}{\beta} dt \sum_{i \in N_a(t)} x_i^*(t) \frac{d}{dt} A(t) \\
&\leq \frac{1}{\beta} \frac{d}{dt} A(t)\ dt \quad [\text{since } \textstyle\sum_i x_i^*(t) \leq 1]
\end{aligned}$$

We now consider $\Delta\Phi_2(t)$. The maximum increase occurs when all jobs in $N_o(t)$ are in sequential phase. Since $y_i(t)$ could increase by dt for each job $i \in N_o(t) \cap N_a(t)$, we have $\Delta\Phi_2(t) \leq \frac{8k}{\beta^3} \sum_{i \in N_o(t)} w_i(t) dt \leq \frac{8k}{\beta^3} \Delta\mathrm{OPT}(t)$. Hence,

$$\frac{d}{dt}\Phi(t) \leq \frac{1}{\beta} \frac{d}{dt} A(t) + \frac{8k}{\beta^3} \frac{d}{dt} \mathrm{OPT}(t)$$

Processing by WLAPS*:* We partition $N_a(t)$ into $\mathcal{S}(t)$ and $\mathcal{P}(t)$ depending on whether a job in $N_a(t)$ is in sequential phase or parallel phase at time t under the schedule by WLAPS. Also, let $\mathcal{P}'(t) := N'_a(t) \cap \mathcal{P}(t)$ denote the set of jobs that WLAPS is working on at time t, which are in their parallel phases.

Consider any job $i \in \mathcal{P}'(t) \setminus N_o(t)$. Since OPT has already finished job i, we have that $z_i(t) = \texttt{On}(i, t, \infty)$. Furthermore,

$z_i(t)$ decreases at a rate of $-s\frac{w_i(t)}{\beta w(t)}$ by definition of our algorithm. Also we have that $\sum_{j \in N_a, a_j \le a_i} w_j(t) \ge (1-\beta)w(t)$ from the fact that $i \in N'_a(t)$. Hence,

$$\begin{aligned}\frac{d}{dt}\Phi_1(t) &\le -\frac{s}{\beta}\sum_{i\in\mathcal{P}'(t)\setminus N_o(t)}\frac{w_i(t)}{\beta w(t)}\sum_{j\in N_a(t),a_j\le a_i} w_j(t)\\ &\le -\frac{s(1-\beta)}{\beta^2}\sum_{i\in\mathcal{P}'(t)\setminus N_o(t)} w_i(t)\end{aligned}$$

For any job $i \in \mathcal{S}(t)\setminus N_o(t)$, whether WLAPS works on i or not, the rate of change of $y_i(t)$ is -1, since i is in a sequential phase and OPT has completed the job. Thus $\frac{d}{dt}\Phi_2(t) \le -\frac{8k}{\beta^3}\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t)$. In sum, we have that

$$\frac{d}{dt}\Phi(t) \le -\frac{s(1-\beta)}{\beta^2}\sum_{i\in\mathcal{P}'(t)\setminus N_o(t)} w_i(t) - \frac{8k}{\beta^3}\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t)$$

In the remaining case, the following lemma will be used.

Lemma 2.1 *For any job $j \in [n]$, $\sum_{i\in[n],a_i\ge a_j} \texttt{Opt}(i,a_i,t) \le t-a_j$.*

Proof: Consider any job i such that $a_i \ge a_j$. Note that OPT did $\texttt{Opt}(i,a_i,t)$ amount of parallel work for job i during $[a_i,t]$, therefore during $[a_j,t]$. The lemma immediately follows from the fact that the adversary is given only speed 1. ■

Time Elapse: We investigate the increase rate of the potential function only due to time elapsing.

$$\begin{aligned}\frac{d}{dt}\Phi_1(t) &= \frac{1}{\beta}\sum_{i\in N_a(t)} z_i(t)\sum_{j\in N_a(t),a_j\le a_i} k(k-1)(t-a_j)^{k-2}\\ &= \frac{1}{\beta}k(k-1)\sum_{j\in N_a(t)}(t-a_j)^{k-2}\sum_{i\in N_a(t),a_i\ge a_j} z_i(t)\end{aligned}$$

But notice that $z_i(t) = \frac{\texttt{On}(i,t,\infty)\cdot\texttt{Opt}(i,a_i,t)}{\rho_i} \le \texttt{Opt}(i,a_i,t)$, since $\texttt{On}(i,t,\infty) \le \rho_i$. Therefore, we get that $\frac{d}{dt}\Phi_1(t)$ is at most

$$\begin{aligned}&\frac{1}{\beta}k(k-1)\sum_{j\in N_a(t)}(t-a_j)^{k-2}\sum_{i\in N_a(t),a_i\ge a_j}\texttt{Opt}(i,a_i,t)\\ \le\ &\frac{1}{\beta}k(k-1)\sum_{j\in N_a(t)}(t-a_j)^{k-2}(t-a_j) \quad \text{[By Lemma 2.1]}\\ =\ &\frac{1}{\beta}k(k-1)\sum_{j\in N_a(t)}(t-a_j)^{k-1} = \frac{1}{\beta}(k-1)\frac{d}{dt}A(t)\end{aligned}$$

Before addressing $\frac{d}{dt}\Phi_2(t)$ due to time, notice that for any job i, it holds that $(t-a_i) \ge y_i(t)$ because $y_i(t)$ amount of time is required to complete $y_i(t)$ amount of sequential work of job i.

$$\begin{aligned}\frac{d}{dt}\Phi_2(t) &= \frac{8k}{\beta^3}\sum_{i\in N_a(t)} k(k-1)(t-a_i)^{k-2}y_i(t)\\ &\le \frac{8k^2}{\beta^3}(k-1)\sum_{i\in N_a(t)}(t-a_i)^{k-2}y_i(t)\end{aligned}$$

In sum we obtain

$$\begin{aligned}\frac{d}{dt}\Phi(t) \le\ &\frac{1}{\beta}(k-1)\frac{d}{dt}A(t) +\\ &\frac{8k^2}{\beta^3}(k-1)\sum_{i\in N_a(t)}(t-a_i)^{k-2}y_i(t) \qquad (5)\end{aligned}$$

We now want to bound $\sum_{i\in N_a(t)}(t-a_i)^{k-2}y_i(t)$ by $\frac{d}{dt}A(t)$. Now, since we know that $y_i(t) \le (t-a_i)$, it is easy to see that $k\sum_{i\in N_a(t)}(t-a_i)^{k-2}y_i(t) \le \frac{d}{dt}A(t)$. However, this bound does not suffice for the rest of the analysis, because of the magnitude of the constant sitting in front of this term (which is $\frac{8k^2}{\beta^3}(k-1)$). To handle this issue, we only consider the jobs which are "*old*" when compared to $\frac{8k^2}{\beta^3}\sigma_i$, for this sum, and terms due to all other "*young*" jobs will be accounted for, separately. The trick to do this is simple. For all $i \in N_a(t)$, we charge $(t-a_i)^{k-2}y_i(t)$ *directly* to the optimal solution as long as $t-a_i \le \frac{8k^2}{\beta^3}\sigma_i$; this can be done since OPT requires at least σ_i amount of time to get job i done.

Lemma 2.2 *The total contribution of all the young jobs, integrated over time is at most $(\frac{8k^2}{\beta^3})^k$OPT*

Proof: Consider any job i. We now show that that the *total* contribution of the term $(t-a_i)^{k-2}y_i(t)$ over all times at which i remains young (i.e. $(t-a_i) \le \frac{8k^2}{\beta^3}\sigma_i$) can be bounded by the cost it incurs in OPT's schedule. This can then be charged to the optimal solution by adding an extra factor to the competitive ratio. More specifically, the total increase (summed over all jobs) can be at most

$$\begin{aligned}&\sum_{i\in[n]}\int_{a_i}^{a_i+\frac{8k^2}{\beta^3}\sigma_i}\frac{8k^2}{\beta^3}(k-1)(t-a_i)^{k-2}y_i(t)\,dt\\ \le\ &\sum_{i\in[n]}(\frac{8k^2}{\beta^3}\sigma_i)^k\\ \le\ &(\frac{8k^2}{\beta^3})^k\text{OPT}.\end{aligned}$$

The last inequality holds knowing that $\text{OPT} \ge \sum_{i\in[n]}(\sigma_i)^k$ because each job i takes at least σ_i time units to complete in the optimal solution's schedule. ■

Since we can handle all young jobs in the above fashion, let us only consider old jobs, such that $t-a_i \ge \frac{8k^2}{\beta^3}\sigma_i \ge \frac{8k^2}{\beta^3}y_i$. In the worst case, all the jobs are old. In this case, equation (5) can be simplified to,

$$\frac{d}{dt}\Phi(t) \le (\frac{1}{\beta}+1)(k-1)\frac{d}{dt}A(t).$$

Final Step of the Analysis:

Recall that throughout the analysis, our main goal has been to show that

$$\frac{d}{dt}A(t) + \frac{d}{dt}\Phi(t) \le \frac{16k^3}{\beta^3}\frac{d}{dt}\text{OPT}(t) \qquad (6)$$

We first complete this proof, and then show how this leads us to the desired guarantees. By summing up the change (rate) of $\Phi(t)$ for all the cases, it is easy to see that showing the following inequality, is sufficient for showing that in equation (6) holds.

$$\begin{aligned}\frac{8k^3}{\beta^3}\frac{d}{dt}\mathrm{OPT}(t) \quad &\geq \quad (\frac{1}{\beta}+1)k\frac{d}{dt}A(t) - \\ &\frac{s(1-\beta)}{\beta^2}\sum_{i\in\mathcal{P}'(t)\setminus N_o(t)} w_i(t) - \\ &\frac{8k}{\beta^3}\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t)\end{aligned}$$

To this end, we consider the following three cases.

(a) $\frac{d}{dt}\mathrm{OPT}(t) \geq \frac{\beta^2}{4}\frac{d}{dt}A(t)$*:* This is the simplest case. Since the adversary has unfinished jobs whose total weight is significant, we can charge the positive term involving $\frac{d}{dt}A(t)$ to $\frac{d}{dt}\mathrm{OPT}(t)$. Formally, $(\frac{1}{\beta}+1)k\ \frac{d}{dt}A(t) \leq \frac{2k}{\beta}\frac{4}{\beta^2}\ \frac{d}{dt}\mathrm{OPT}(t) \leq \frac{8k^2}{\beta^3}\ \frac{d}{dt}\mathrm{OPT}(t)$.

(b) $\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t) \geq \frac{\beta^2}{4}\frac{d}{dt}A(t)$*:* In this case, the second negative term for jobs in $\mathcal{S}(t)$ will be used to offset the positive term. Indeed, we have $(\frac{1}{\beta}+1)k\frac{d}{dt}A(t) - \frac{8k}{\beta^3}\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t) \leq \frac{2k}{\beta}\frac{d}{dt}A(t) - \frac{2k}{\beta}\frac{d}{dt}A(t) \leq 0$

(c) $\frac{d}{dt}\mathrm{OPT}(t) < \frac{\beta^2}{4}\frac{d}{dt}A(t)$ *and* $\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t) < \frac{\beta^2}{4}\frac{d}{dt}A(t)$*:* This is the final case, where the first negative term involving parallel-phase jobs will override the positive term. Since $\mathcal{P}'(t)\setminus N_o(t) = (N'_a(t)\cap\mathcal{P}(t))\setminus N_o(t) = N'_a(t)\setminus\mathcal{S}(t)\setminus N_o(t)$, it follows that $\sum_{i\in\mathcal{P}'(t)\setminus N_o(t)} w_i(t) \geq \sum_{i\in N'_a(t)} w_i(t) - \sum_{i\in\mathcal{S}(t)} w_i(t) - \sum_{i\in N_o(t)} w_i(t) \geq \beta(1-\frac{1}{2}\beta)\frac{d}{dt}A(t)$. Substituting $s = k(1+16\beta)$ we obtain,

$$\begin{aligned}&(\frac{1}{\beta}+1)k\frac{d}{dt}A(t) - \frac{s(1-\beta)}{\beta^2}\sum_{i\in\mathcal{P}'(t)\setminus N_o(t)} w_i(t) \\ \leq\ & \Big((\frac{1}{\beta}+1)k - \frac{k(1+16\beta)(1-\beta)(1-\frac{1}{2}\beta)}{\beta}\Big)\frac{d}{dt}A(t) \\ \leq\ & \frac{k}{\beta}\Big((1+\beta)-(1+16\beta)(1-\frac{3}{2}\beta)\Big)\frac{d}{dt}A(t) \\ \leq\ & 12k(-1+2\beta)\frac{d}{dt}A(t) \leq 0 \quad [\text{By } 0<\beta\leq\tfrac{1}{2}]\end{aligned}$$

And this completes the proof that equation (6) holds at all times. To complete the analysis, let WLAPS_s denote the total cost incurred by WLAPS (when given a speed of s), and OPT_1 denote the cost of the optimal schedule (operating at unit speed). We have that the cost incurred by WLAPS_s is

$$\begin{aligned}\int_0^\infty \frac{d}{dt}A(t)dt \quad &= \quad \int_0^\infty (\frac{d}{dt}A(t)+\frac{d}{dt}\Phi(t))dt \\ &\leq \quad \int_0^\infty \Big(\frac{16k^3}{\beta^3}\frac{d}{dt}\mathrm{OPT}(t)dt\Big) + (\frac{8k^2}{\beta^3})^k\mathrm{OPT}_1 \\ &= \quad (\frac{16k^3}{\beta^3}+(\frac{8k^2}{\beta^3})^k)\mathrm{OPT}_1 \leq 2(\frac{8k^2}{\beta^3})^k\mathrm{OPT}_1\end{aligned}$$

Here, the second equality easily follows from the fact that $\Phi(0)=\Phi(\infty)=0$. The third inequality follows from inequality (6). The additional term is due to the fact that we had not accounted for the contribution of "young" jobs towards $\frac{d}{dt}\Phi(t)$ in (6), but showed (in Lemma 2.2) that the total cost integrated over time is at most $(\frac{8k^2}{\beta^3})^k\mathrm{OPT}_1$. As a result, by scaling the speed WLAPS is given, we obtain the following theorem.

Theorem 2.3 *Let* $0<\beta\leq\frac{1}{2}$ *be a constant. Then* WLAPS *is* $k(1+16\beta)$*-speed* $\frac{16k^2}{\beta^3}$*-competitive for the problem of minimizing* ℓ_k *norm flow time with arbitrary speed up curves.*

Remark 2.4 *The assumption* $0<\beta\leq\frac{1}{2}$ *that is used in Theorem 2.3 is not essential in the analysis. Rather, it is to make our analysis relatively simpler.*

3. LIMITATION OF LAPS FOR ℓ_K-NORM SCHEDULING

In this section, we show that for minimizing ℓ_k-norm flow time LAPS performs poor in the non-work-preserving setting. More specifically, we will test LAPS in the scheduling setting with arbitrary speedup curves and broadcast scheduling setting.

We first show that LAPS can be arbitrarily bad even with any constant speed given for jobs with arbitrarily speedup curves. The main idea of constructing the adversarial example is to repeatedly request fully sequential jobs to prevent LAPS from working on parallel jobs. Consequently, LAPS wastes its processing power procrastinating parallel jobs substantially; unlike in L_1-norm flow time, these delayed jobs will cause a huge penalty.

Theorem 3.1 *Let* $k\geq 2$ *be an integer. For any* $0<\beta\leq 1$*, the algorithm* LAPS_β *is not* $O(1)$*-competitive even with any constant speed given for the problem of minimizing* ℓ_k *norm flow time where jobs have arbitrarily speed up curves.*

Proof: Recall that LAPS works on only β fraction of alive jobs which arrived most recently. Let σ denote the adversarial instance. For simplicity of our argument, suppose that LAPS is given an integer speed $s>1$. Let $\mathrm{LAPS}_s(\sigma)$ and $\mathrm{OPT}_1(\sigma)$ denote the k^{th} power of flow time for the given instance σ; the subscript s and 1 are used to denote the speed LAPS and OPT are given, respectively. Let $M>0$ be a sufficiently large integer which will be defined later. The instance σ is constructed as follows.

- At time 0, one fully parallelizable job j_0 having size M arrives.
- At each integer time $t\in[0,M^2-1]$, sM sequential unit-sized jobs arrive. Let $\mathcal{J}_t$ denote the set of sequential jobs that arrive at time t.

Note that all sM jobs in $\mathcal{J}_t$ are unsatisfied by LAPS during $[t,t+1)$, since they are unit-sized sequential jobs. Therefore during $[0,M^2]$, as long as j_0 is alive, it is processed at a rate of at most $\frac{1}{M}$, since even in the best case $\beta=1$, it equally shares the processors with other sM sequential jobs. Thus job j_0 is not finished until time M^2, which implies that $\mathrm{LAPS}_s(\sigma)\geq(M^2)^k$. To the contrary, let OPT work on only job j_0. Then job j_0 is finished at time M, and all sequential jobs are finished in one time step. Hence, $\mathrm{OPT}_1(\sigma) \leq M^k + sM^3$. It is easy to check that $\mathrm{LAPS}_s(\sigma)/\mathrm{OPT}_1(\sigma) \to \infty$ as $M \to \infty$. ■

Using a similar idea, we can show a negative result for broadcast scheduling.

Theorem 3.2 *Let $k \geq 2$ be an integer. For any $0 < \beta \leq 1$, the algorithm* LAPS *is not $O(1)$-competitive even with any constant speed for the problem of minimizing ℓ_k norm flow time in broadcast scheduling where pages are varying sized.*

Proof: Let σ denote the adversarial instance. For simplicity, suppose that LAPS is given an integer speed $s > 1$. Let $\mathrm{LAPS}_s(\sigma)$ and $\mathrm{OPT}_1(\sigma)$ denote the k^{th} power of flow time for the given instance σ; the subscript s and 1 are used to denote the speedup LAPS and OPT are given respectively. Let M be a sufficiently large integer which will be defined later. The instance σ is defined as follows.

- At time 0, a request r_0 for page p having size M, arrives.
- At each integer time $t \in [0, M^2 - 1]$, sM requests for page q having size s, arrive.

We first investigate the schedule by LAPS. We observe that during the interval $[0, M^2]$ there are at least sM alive requests. This is because the new requests which are released at each integer time $t \in [0, M^2-1]$ cannot be satisfied within unit time, since they are asking for pages having size s. As a result, the request r_0 is processed at a rate of at most $\frac{1}{M}$ for each unit time slot. Hence request is not satisfied until time M^2 and we obtain $\mathrm{LAPS}(\sigma)_s \geq M^{2k}$.

We now shift our attention to the schedule by OPT. We let OPT work on r_0 during $[t, t+1)$ for every odd integer t until request r_0 is finished. Thus, r_0 will be satisfied at time $2M$. For other empty time slots, we let OPT broadcast page q sequentially in a round robin fashion. Notice that this implies that a request for page q is satisfied in at most $4s$ time steps. Hence we have that $\mathrm{OPT}_1(\sigma) \leq (2M)^k + sM^3(4s)^k$. It is easy to see that $\mathrm{LAPS}_s(\sigma)/\mathrm{OPT}_1(\sigma)$ goes to infinity as $M \to \infty$. ■

4. BROADCAST SCHEDULING FOR VARYING SIZED PAGES

In this section, we address the problem of minimizing ℓ_k-norm flow time in broadcast scheduling where pages have non-uniform sizes. Recently, Bansal et al. [1] gave a very clean solution for the same problem when $k = 1$, which is the inspiration of our algorithm and analysis. The problem is formally defined as follows: There are n pages stored at the server and each page p has an integer size σ_p; it is composed of σ_p unit-sized parts, $\{(p, i) \mid i \in [\sigma_p]\}$. For each integer time t, the server can transmit one part of a specific page p during $[t-1, t)$. Each request $r \in [m]$ asking for a page p_r arrives at the server at time a_r in online fashion. Request r is satisfied when it receives all points $\{(p, i) \mid i \in [\sigma_p]\}$ *in the order* of $(p, 1), (p, 2), ..., (p, \sigma_p)$, but not necessarily contiguously. Let C_r denote the time when the request r is satisfied. The goal is to give a scheduling of the server which minimize the ℓ_k-norm of the flow times, i.e. $(\sum_r (C_r - a_r)^k)^{1/k}$. Again, like in the previous sections, this is equivalent to minimizing $(\sum_r (C_r - a_r)^k)$, and this is the objective function which we will focus on optimizing.

4.1 The Fractional Algorithm WLAPS for Broadcast Scheduling

We use the same algorithm WLAPS with some small modifications. However, our algorithm is a *fractional* algorithm in the sense that it is allowed to transmit an infinitesimal amount of data for more than one pages during any arbitrarily short interval. Furthermore, the notion of when a request is *fractionally completed* is different from our original requirement: suppose that our algorithm WLAPS broadcasts each page $p \in [n]$ at rate of $y_p(t)$. Then, in the fractional setting, we say that a request r is completed at time

$$C_r := \inf\{t : \int_{a_r}^{t} y_p(t)dt \geq \sigma_p\}.$$

In words, C_r is the earliest time when σ_p units of page p are broadcast after a_r. Notice that we require nothing about the order of the unit-sized components in this definition of fractional completion. To fix this issue, we can then apply the rounding technique of Bansal et al. [1] (Section 5.3) to convert such a fractional solution to one which is integral with an additional loss in the competitive ratio.

4.2 Notation

Our notation is similar to that used in earlier sections. Let $N_a(t)$ denote the set of unsatisfied requests at time t, i.e. all those requests r such that $t < C_r$. For each request $r \in N_a(t)$, let us define its *weight* at time t to be $w_r(t) = k(t - a_r)^{k-1}$. Among the set of requests $N_a(t)$, let $N'_a(t)$ denote those requests with the *latest arrival times* whose weights sum up to $\beta w(t)$, where $w(t) = \sum_{r \in N_a(t)} w_r(t)$; here we, as did in Section 2.2, rely on the same simplifying assumption that there exists a set of latest arriving requests whose total weight is exactly $\beta w(t)$.

4.3 Algorithm

The algorithm is similar to WLAPS, except that it shares its processing power based on the requests which are yet unsatisfied. At any time t, the algorithm WLAPS distributes its processing power among the requests in $N'_a(t)$, in proportion to their weights at time t. Let $x_r(t)$ denote the fraction of processing power request r receives at time t under the schedule of WLAPS. Then,

$$x_r(t) := s \cdot \frac{w_r(t)}{\beta w(t)}, \ \ \forall r \in N'_a(t)$$

What this means is the following: whenever the algorithm devotes processing towards a request r, it broadcasts the page p_r at a rate of $x_r(t)$ at time t. Note that if there are several unsatisfied requests for the same page p, the total rate at which p is broadcast at time t is then $y_p(t) = \sum_{r \in N'_a(t), p_r = p} x_r(t)$. It is important to note that although request r is processed at a rate of $y_{p_r}(t)$ due to other requests for the same page p_r, the fraction of processing power dedicated to request r is only $x_r(t)$. For this reason, sometimes we will say that $x_r(t)$ is the *share* of request r at time t.

4.4 Analysis

The analysis again employs a potential function based argument. Let $y^*_p(t)$ denote the transmission rate of page p by OPT at time t. For each request $p \in [n]$, let $\mathtt{Opt}(p, t_1, t_2)$

denote the total amount for page p transmitted by OPT during $[t_1, t_2]$, i.e.

$$\texttt{Opt}(p, t_1, t_2) := \int_{t_1}^{t_2} y_p^*(t)dt$$

For any request $r \in [m]$, define the *total share* for request r during $[t_1, t_2]$, denoted by $\texttt{On}(r, t_1, t_2)$. Formally,

$$\texttt{On}(r, t_1, t_2) := \int_{t_1}^{t_2} x_r(t)dt$$

Again, notice that $\texttt{On}(r, a_r, C_r)$ could be much smaller than σ_p if there are multiple pending requests for the same page p_r. We now define a variable z_r for each request r which will be used to define the potential function $\Phi(t)$.

$$z_r(t) = \frac{\texttt{On}(r, t, \infty) \cdot \texttt{Opt}(p_r, a_r, t)}{\sigma_{p_r}}$$

The potential function $\Phi(t)$ is then defined as follows.

$$\Phi(t) := \frac{2}{\epsilon} \sum_{r \in N_a(t)} z_r(t) \sum_{a_{r'} \le a_r, r' \in N_a(t)} w_{r'}(t)$$

We start by investigating the change of the potential function for all possible events.

Change of the Potential Function

`Request Arrival:` When a request r arrives at time t, $z_r(t) = 0$ because the optimal solution has not had a chance to work on request r yet. It is easy to see that any new terms which appear are zero, since the requests are indexed according to their arrival time. Thus $\Delta\Phi = 0$.

`Request Completion by` WLAPS`:` When a request r is completed, r is removed from $N_a(t)$. As a result, some terms may disappear from $\Phi(t)$. This can only decrease Φ, since all terms are non-negative in $\Phi(t)$. Thus $\Delta\Phi(t) \le 0$.

`Processing by` OPT`:` Fix a sufficiently small interval $[t, t + dt]$. Consider any page $p \in [n]$. The amount that OPT broadcasts for page p is $y_p^*(t)\ dt$. Let $\mathcal{R}(t, p)$ denote all requests for page p which are not satisfied yet under the schedule by WLAPS at time t. Formally,

$$\mathcal{R}(t, p) := \{r \in [m] \mid p_r = p, a_r \le t < C_r\}$$

The total increase of the potential function is at most

$$\frac{2}{\epsilon} \sum_p \sum_{r \in \mathcal{R}(t,p)} \frac{\texttt{On}(r, t, \infty) \cdot y_p^*(t)dt}{\sigma_p} \sum_{a_{r'} \le a_r, r' \in N_a(t)} w_{r'}(t)$$

Following the argument of [1] (Section 5.2), we can show that $\sum_{r \in \mathcal{R}(t,p)} \texttt{On}(r, t, \infty) \le \sigma_p$ as in [1]. The idea is the following: all the requests in $\mathcal{R}(t, p)$ are yet unsatisfied at time t, but none of them will be unsatisfied, once a collective total of σ_p units of this page are transmitted *in the future*. Hence, it must be that total share of processing devoted to these requests in the future cannot exceed σ_p.

Now, by combining this observation with the fact that $\sum_p y_p^*(t) \le 1$ (since the optimal offline solution is only given unit speed), we obtain

$$\begin{aligned}
\Delta\Phi(t) &\le \frac{2}{\epsilon} dt \sum_p y_p^*(t) \sum_{r \in \mathcal{R}(t,p)} \frac{\texttt{On}(r, t, \infty)}{\sigma_p} \sum_{a_{r'} \le a_r, r' \in N_a(t)} w_{r'}(t) \\
&\le \frac{2}{\epsilon} dt \sum_p y_p^*(t) \sum_{r \in \mathcal{R}(t,p)} \frac{\texttt{On}(r, t, \infty)}{\sigma_p} \sum_{r' \in N_a(t)} w_{r'}(t) \\
&\le \frac{2}{\epsilon} \Delta A(t)
\end{aligned}$$

`Processing by` WLAPS`:` Consider any request $r \in N'_a(t) \setminus N_o(t)$. Since OPT has already finished the request r, we have that $\frac{\texttt{Opt}(p_r, a_r, t)}{\sigma_{p_r}} \ge 1$. Thus, $z_r(t) \ge \texttt{On}(r, t, \infty)$. Moreover, since $r \in N'_a(t)$, $\texttt{On}(r, t, \infty)$ *decreases* at a rate of $x_r(t) = s\frac{w_r(t)}{\beta w(t)}$. Therefore, $z_r(t)$ also decreases at at least the same rate. Additionally we also have that $\sum_{r' \in N_a, a_{r'} \le a_r} w_{r'}(t) \ge (1 - \beta)w(t)$, again because $r \in N'_a(t)$. As a consequence, we have

$$\begin{aligned}
\frac{d}{dt}\Phi(t) &\le -\frac{2}{\epsilon} \sum_{r \in N'_a(t) \setminus N_o(t)} \frac{s\ w_r(t)}{\beta w(t)} \sum_{r' \in N_a(t), a_{r'} \le a_r} w_{r'}(t) \\
&\le -\frac{2}{\epsilon} \sum_{r \in N'_a(t) \setminus N_o(t)} \frac{s\ w_r(t)}{\beta w(t)} (1 - \beta)w(t) \\
&= -\frac{2s(1-\beta)}{\beta\epsilon} \sum_{r \in N'_a(t) \setminus N_o(t)} w_r(t) \\
&\le -\frac{2s(1-\beta)}{\beta\epsilon} \Big(\sum_{r \in N'_a(t)} w_r(t) - \sum_{r \in N_o(t)} w_r(t) \Big) \\
&= -\frac{2s(1-\beta)}{\epsilon} \frac{d}{dt} A(t) + \frac{2s(1-\beta)}{\beta\epsilon} \frac{d}{dt} \mathrm{OPT}(t)
\end{aligned}$$

`Time Elapse:` Recall that $\mathcal{R}(t, p)$ denotes the set of all requests that are yet unsatisfied for page p under the schedule of WLAPS at time t.

$$\begin{aligned}
\frac{d}{dt}\Phi(t) &= \frac{2}{\epsilon} \sum_{r \in N_a(t)} z_r(t) \sum_{r' \in N_a(t), a_{r'} \le a_r} k(k-1)(t - a_{r'})^{k-2} \\
&= \frac{2}{\epsilon} k(k-1) \sum_{r' \in N_a(t)} (t - a_{r'})^{k-2} \sum_{r \in N_a(t), a_r \ge a_{r'}} z_r(t)
\end{aligned}$$

But again, notice that in the inner summation, we have $z_r(t) = \frac{\texttt{On}(r,t,\infty) \cdot \texttt{Opt}(p, a_r, t)}{\sigma_p} \le \frac{\texttt{On}(r,t,\infty) \cdot \texttt{Opt}(p, a_{r'}, t)}{\sigma_p}$, since $a_{r'} \le a_r$. Thus, we can obtain the following set of inequalities

$$\begin{aligned}
\sum_{r \in N_a(t), a_r \ge a_{r'}} z_r(t) &= \sum_p \sum_{r \in \mathcal{R}(p,t), a_r \ge a_{r'}} z_r(t) \\
&\le \sum_{r \in cR(p,t), a_r \ge a_{r'}} \frac{\texttt{On}(r, t, \infty) \cdot \texttt{Opt}(p, a_{r'}, t)}{\sigma_p} \\
&\le \texttt{Opt}(p, a_{r'}, t)
\end{aligned}$$

The last inequality is because $\sum_{r \in \mathcal{R}(p,t)} \texttt{On}(r, t, \infty) \le \sigma_p$, following the arguments in [1] (Section 5.2). Moreover, recall that $\texttt{Opt}(p, a_{r'}, t)$ is the amount of processing OPT spent on page p during $[a_{r'}, t]$. Since the optimal solution has one speed, the total amount of all pages transmitted in the interval $[a_{r'}, t]$ can be at most $(t - a_{r'})$. Therefore, we get

$$\begin{aligned}\frac{d}{dt}\Phi(t) &\leq \frac{2}{\epsilon}k(k-1)\sum_{r'\in N_a(t)}(t-a_{r'})^{k-2}(t-a_{r'}) \\ &= \frac{2}{\epsilon}(k-1)\frac{d}{dt}A(t)\end{aligned}$$

Final Step of Analysis

We show that $\frac{d}{dt}A(t) + \frac{d}{dt}\Phi(t) \leq \frac{8k}{\epsilon^2}\frac{d}{dt}\mathrm{OPT}(t)$. Let $\beta = \frac{\epsilon}{4k}$. Recall that $s = k + \epsilon$. By summing up all the change (rate) of $\Phi(t)$ for all the events, we obtain

$$\begin{aligned}\frac{d}{dt}A(t) + \frac{d}{dt}\Phi(t) &\leq \Big(1 + \frac{2k}{\epsilon} - \frac{2s(1-\beta)}{\epsilon}\Big)\frac{d}{dt}A(t) + \\ &\quad \frac{2s(1-\beta)}{\beta\epsilon}\frac{d}{dt}\mathrm{OPT}(t) \\ &= (-\frac{1}{2} + \frac{\epsilon}{2k})\frac{d}{dt}A(t) + \frac{16k^2}{\epsilon^2}\frac{d}{dt}\mathrm{OPT}(t) \\ &\leq \frac{16k^2}{\epsilon^2}\frac{d}{dt}\mathrm{OPT}(t)\end{aligned}$$

With the fact that $\Phi(0) = \Phi(\infty) = 0$, the following theorem easily follows.

Theorem 4.1 *For any $0 < \epsilon \leq 1$, there exists a $(k+\epsilon)$-speed $(\frac{4k}{\epsilon})^{2/k}$-competitive deterministic online algorithm which gives a fractional broadcast schedule for minimizing ℓ_k norm flow time.*

4.5 Rounding to Integer Broadcast Schedule

Bansal et al. [1] (Section 5.3) show a very elegant deterministic rounding algorithm which takes a 1-speed fractional broadcast schedule and yields $(1+\delta)$-speed integer broadcast schedule satisfying the following property: for any request r, $C_r^I - a_r \leq \frac{3}{\delta}(C_r - a_r) + \frac{5}{\delta}$, where C_r^I denotes the finish time of r under the integer broadcast schedule. By applying this rounding technique to Theorem 4.1, we obtain an integer broadcast schedule, which completes our analysis.

Theorem 4.2 *Let $k \geq 1$ be a constant . For any $0 < \epsilon \leq 1$, there exists a $(k+\epsilon)$-speed $O((\frac{1}{\epsilon})^{1+2/k})$-competitive deterministic online algorithm in broadcast scheduling for minimizing ℓ_k norm flow time.*

5. CONCLUSION

In this paper we showed how to obtain a $(2+\epsilon)$-speed $O(1)$-competitive algorithm for scheduling jobs with arbitrary speed-up curves for the ℓ_2 norm of flow. Currently we do not know whether WLAPS is scalable or not. To find an scalable algorithm, either showing that WLAPS is scalable or finding another algorithm, would be a good problem worthy of attention.

Acknowledgments: The authors thank Nikhil Bansal for his helpful discussion about this problem. Sungjin Im and Benjamin Moseley thank Chandra Chekuri for his support and encouragement.

6. REFERENCES

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

[2] Nikhil Bansal and Kirk Pruhs. Server scheduling in the lp norm: a rising tide lifts all boat. In *ACM Symposium on Theory of Computing*, pages 242–250, 2003.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA*, pages 685–692, 2009.

[9] Sungjin Im and Benjamin Moseley. An online scalable algorithm for average flow time in broadcast scheduling. In *ACM-SIAM Symposium on Discrete Algorithms*, page To Appear, 2010.

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

[11] Bala Kalyanasundaram and Kirk Pruhs. Minimizing flow time nonclairvoyantly. *J. ACM*, 50(4):551–567, 2003.

[12] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theorertical Computer Science*, 130(1):17–47, 1994.

[13] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.

[14] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. In *Handbook on Scheduling*. CRC Press, 2004.

[15] Julien Robert and Nicolas Schabanel. Non-clairvoyant batch sets scheduling: Fairness is fair enough. In *European Symposium on Algorithms*, pages 741–753, 2007.

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

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