Online Batch Scheduling for Flow Objectives

Sungjin Im [*] Benjamin Moseley [†]

January 16, 2013

Abstract

Batch scheduling gives a powerful way of increasing the throughput by aggregating multiple homogeneous jobs. It has applications in large scale manufacturing as well as in server scheduling. In batch scheduling, when explained in the setting of server scheduling, the server can process requests of the same type up to a certain number simultaneously. Batch scheduling can be seen as *capacitated* broadcast scheduling, a popular model considered in scheduling theory. In this paper, we consider an online batch scheduling model. For this model we address flow time objectives for the *first* time and give positive results for average flow time, the k-norms of flow time and maximum flow time. For average flow time and the the k-norms of flow time we show algorithms that are $O(1)$-competitive with a small constant amount of resource augmentation. For maximum flow time we show a 2-competitive algorithm and this is the best possible competitive ratio for any online algorithm.

[*]Department of Computer Science, Duke University, Durham NC 27708-0129. sungjin@cs.duke.edu. Partially supported by NSF grant CCF-1008065.

[†]Toyota Technological Institute, Chicago IL, 60637, USA. moseley@ttic.edu

1 Introduction

Using k-norms and not ℓ_k-nroms
should we stick to requests or jobs
should be say scheduled or broadcasted

A majority of works in scheduling literature studies the case where all jobs must be processed sequentially and only on a single machine at any point in time. However, more general problems arise practice. For instance, jobs could be parallelizable. In this setting a job can be be processed on multiple machine simultaneously. Scheduling parallelizable jobs has received a significant amount of attention in theoretical scheduling literature, for instance [13, 9, 15, 23, 14]. The reason to address the parallelizability of a job is to be able to process more work per unit time by using more machines.

Another way to increase the work being done per time unit is to batch homogeneous jobs. In a batch scheduling setting, jobs of the same type can be aggregated and processed together *simultaneously*. Thus, one unit of processing can decrease the work that needs to be performed on more than one job. Batch scheduling arises in many real work systems. For example, if jobs require the same code, then the server can let the jobs share the code loaded into the memory. Or in a multicast network, many different clients can receive a communication simultaneously. Batch scheduling is not only restricted to computers and networks. In particular, in manufacturing lines, such as semiconductor manufacturing, tasks are grouped together to be processed simultaneously. See [22], for a variety of industry applications of batched scheduling.

A special type of batch scheduling, known as broadcast scheduling, has received a significant amount of attention recently in scheduling literature. In the broadcast scheduling model, there are n pages of data stored at a server. Over time requests arrive for specific pages. When the server broadcasts a page p, *all* unsatisfied/outstanding requests are satisfied simultaneously. Broadcast scheduling finds applications in multicast systems, LAN and wireless networks [24, 1, 2]. Notice that here any number of requests can be satisfied simultaneously. This can be viewed as having an infinite batch size. In practice, however, there could be a limit on the number of requests that a server can handle simultaneously, as was pointed out in [7].

In this paper, we consider online batch scheduling where there is possibly some limit on the number of jobs that can be processed simultaneously. In the *online* setting, the scheduler is only aware of a request once it is sent to the system. Inspired by the broadcast model, we consider a generalization this model to the batched setting. As in the broadcast model, there are n pages of information stored at the sever. Requests arrive over time for different pages. The sever can satisfy up to B_p requests for page p simultaneously. Here B_p is different for each page p and takes some integral value in $(0, \infty]$. We note that the model we consider is not only restricted to applications in wireless networks. Indeed, one can think of the n pages as n different types of jobs. Over time requests come for a specific type of job p to be done and up to B_p jobs of type p can be done simultaneously. This model accurately captures a variety of batched settings, for instance batch scheduling tasks in an assembly line.

The *flow time* of a job is the amount of time it takes the sever to satisfy a job. A client is interested in having the flow time of their job minimized. Let $J_{p,i}$ be the ith request for page p and say that this request arrives at time $r_{p,i}$. Each page p has a size ℓ_p which specifies the amount of work on page p that is required to complete a request for page p. We assume that a page consists of a sequence of unit sized pieces $(p, 1), (p, 2), \ldots, (p, \ell_p)$. The sever can broadcast/schedule one piece of a page at each time step. A requests will need to be in a batch for each piece of a page it requested to be completed. For a given schedule, let $C_{p,i}$ be the time $J_{p,i}$ is completed in the system. The flow time of a job is $C_{p,i} - r_{p,i}$. The scheduler's goal is to determine how jobs should be processed so that the clients are given good service. That is, the schedule's goal is to optimize a quality of service metric. Perhaps the most popular quality of service metrics are based on the flow time of the jobs. In this paper, we study flow time objectives for the *first* time in batch scheduling when the batch size is between 1 and ∞.

The most popular flow time based quality of service metric is minimizing the total or *average flow time*.

Here the schedules goal is to minimize $\sum_{p,i} C_{p,i} - r_{p,i}$ and this objective essentially minimizes the average quality of service. Unfortunately, in by optimizing average flow time, the scheduler could give a few clients poor quality of service as to optimize the flow time of other jobs. Thus, it can be seen that optimizing average flow time is not necessarily 'fair' to all of the jobs. An objective that focuses on ensuring no job is given poor quality of service is minimizing the *maximum flow time*. Finally, an objective that is also used to ensure fairness while not being as stringent as maximum flow time is minimizing the *k-norms of flow time*. Here the objective is to minimize $\sqrt[k]{\sum_{p,i}(C_{p,i} - r_{p,i})^k}$ for some $k \in (1, \infty)$. By focusing on minimizing the k-norms for small values of k greater than one the scheduler optimizes the variance in the requests flow time, which enforces fairness [6]. It can be noted that average flow time is equivalent to setting $k = 1$ and maximum flow time is equivalent to setting $k = \infty$. Each of these objectives are widely considered in scheduling theory and it depends on the requirements of your system for which objective you would consider optimizing.

In theoretical scheduling literature, batch scheduling has been studied only for the objective of maximizing the total throughput [7, 16]. For this objective, each request $J_{p,i}$ has a release time $r_{p,i}$ and deadline $d_{p,i}$. If some request is completed by it deadline, then the scheduler receives some profit $w_{p,i}$. The goal of the scheduler is to maximize the profit obtained. Flow objectives are quite different from the throughput objective. For example, there exists a simple 2-competitive greedy algorithm for the global throughput [7].[1] However, it is known that any deterministic online algorithm is $\Omega(n)$-competitive for the average flow time objective [20]. This strong lower bound holds even in the special case of broadcast scheduling. Further any randomized algorithm is $\Omega(\sqrt{n})$-competitive [3].

To give a flavor of the difficulty of batch scheduling, it would be useful to consider the following algorithm. As discussed before, the main reason to consider batch scheduling is the ability to satisfy multiple jobs simultaneously. Hence, it is natural to think that the algorithm which processes a page with the most number of requests would perform well for batched scheduling. This algorithm is known as Most Requests First (MRF). It is known that MRF is 2-competitive for maximizing throughput [21, 11], however it is far from the optimum for minimizing average flow time, maximum flow time and the k-norms of flow time. Consider the following simple example where the batch size can be at most two for each page. At time 0, $n/2$ requests arrive, one for each distinct page. For simplicity, assume that all pages are unit-sized and thus can be completed in a unit time. In addition, at each integer time until time n^2, the same page p is requested twice. Then the algorithm MRF will broadcast page p at every time step until time n, thereby leaving many requests unsatisfied for a long time. However, one can broadcast page p every other time step and process other requests in between. This schedule's objective is a factor of $\Omega(n)$ less for each of the mentioned objectives over MRF.

As mentioned, there are strong lower bounds on the competitive ratio of any online algorithm in the batched scheduling setting for the problems of minimizing average flow time and the k-norms for $k \in (0, \infty)$. Due to these strong lower bounds, we consider analyzing algorithms in the popular resource augmentation [19]. In the resource augmentation model, an algorithm is given extra resources over the adversary and then the competitive ratio is bounded. In previous work, the resource augmentation usually comes in the form of being able to process jobs at a faster rate. An algorithm is said to be s-speed c-competitive if it can process jobs s times faster than the adversary. In our setting, we will consider algorithms that possibly have resource augmentation on how fast jobs can be processed as well as resource augmentation on the number of jobs that can be satisfied together. We will say that an algorithm is s-speed d-capacity c-competitive if the algorithm can process jobs s times faster than the adversary, the batches can be d times larger and the algorithm achieves a competitive ratio of c. The ultimate goal of a resource augmentation analysis is to find a constant competitive algorithm even with a minimum amount of extra resource augmentation.

[1] We note that [7] studies a more general batch scheduling setting in the offline setting.

Our Results: As mentioned, we consider minimizing flow time objectives for the first time in the batched scheduling setting. First we consider the problems of minimizing average flow time and the k-norms of flow time. Due to space constraints, the results for the k-norms can be found in the appendix. For both of these problems we consider a generalization of an algorithm that was considered in the broadcast scheduling setting [5]. Both of these results will follow a similar framework. We will specify two different cases depending on the size of pages. One is when pages are all unit sized (i.e. $\ell_p = 1$ for all p) and the other is when pages are varying sized. We are able to show the following theorems for the case of unit sized pages.

Theorem 1.1. *For any $\epsilon > 0$, there exists an online algorithm that is $(1+\epsilon)$-speed 2-capacity $O(1/\epsilon^3)$-competitive algorithm for minimizing average flow time when pages are unit-sized.*

Theorem 1.2. *For any $\epsilon > 0$ and any fixed integer $k \in [1, \infty)$, there exists an online algorithm that is $(1+\epsilon)$-speed 2-capacity $O(1/\epsilon^4)$-competitive algorithm for minimizing the k-norms of flow time when pages are unit-sized.*

We extend these result for varying sized pages. Here we require a further relaxation on the capacity constraints.

Theorem 1.3. *For any $\epsilon > 0$, there exists an online algorithm that is $(1+\epsilon)$-speed 6-capacity $O(1/\epsilon^3)$-competitive algorithm for minimizing average flow time when pages are varying-sized.*

Theorem 1.4. *For any $\epsilon > 0$ and any fixed integer $k \in [1, \infty)$, there exists an online algorithm that is $(1+\epsilon)$-speed 6-times $O(1/\epsilon^4)$-competitive algorithm for minimizing the k-norms of flow time when pages are varying-sized.*

Finally, we consider the problem of minimizing the maximum flow time. In this case we do use resource augmentation at all, neither on the processing speed nor the capacity of the batch size. In this case the algorithm considered is the natural extension of the algorithm first-in-first out which always prioritizes the request that arrived the earliest.

Theorem 1.5. *There exists a 2-competitive algorithm for minimizing the maximum flow time.*

This result is tight, since any randomized online algorithm was shown to have competitive ratio larger than $2-\epsilon$ for any fixed constant $\epsilon > 0$ even in the special case of $B_p = \infty$ [10, 11]. Due to space constraints, this result can be found in Appendix C.

Our Technical Contributions: As mentioned before, our algorithm and analysis are inspired by the previous work in broadcast scheduling [5, 14, 12]. We first give a high-level view of the approach in [5, 14] for the k norms objectives, $1 \leq k < \infty$. The main difficulty of broadcast scheduling comes from the fact that an optimal schedule can be very effective in processing multiple requests simultaneously compared the the online scheduler. This is the case particularly since the optimal scheduler knows the future input while our algorithm does not. Hence it is challenging for an online algorithm to decide when and what to broadcast. That is, the online algorithm may want to wait more hoping for a chance to aggregate more requests, however, it increases the wait time of such waiting requests.

The key idea that was used in [5, 14] is to first obtain a fractionally good schedule. To illustrate the idea, we assume that all pages are unit sized where requests arrive at integer times, and it takes a unit time to broadcast a page p. In the real (aka integral) schedule, the scheduler is allowed to transmit only one page per time step. However, in the fractional schedule, at each instantaneous time, multiple pages can be broadcasted fractionally, up to a total of one unit. Then in fractional schedule, a request $J_{p,i}$ is satisfied, if page p is transmitted by one unit since its release time $r_{p,i}$. They then convert their schedule for the fractional case, to a valid real schedule, where only one page is broadcasted in a time step, in a online manner.

Until this, our algorithm and analysis are fairly similar to the previous work. We will also consider a fractional schedule and convert it to a valid schedule in an online manner. However, in batch scheduling, one has to decide which requests get satisfied in a batch; whereas, in the broadcast schedule all requests get satisfied together. Deciding which requests get satisfied makes the problem more challenging. Our algorithm at any point in time, will decide a *single* request that is wants to satisfy. However, now it also needs to decide which requests to group in a batch with this request. Our algorithm will define 'balls' around each request where a ball consists of which requests get satisfied if this request is chosen to be satisfied. The ball will be defined on requests for the same page that arrived the most closely to this request. Using this, we will show a fractionally good scheduler. However, the online conversion to a valid schedule is much more challenging in our case. For the online conversion, we will need use a more intricate analysis. The reason is that when pages have varying sizes the batch a request is in can change over time and we will need to be careful about how the algorithm and optimal solution group requests. To do this, we will determine some structural properties of the optimal solution that we can compare against.

Related Work: Broadcast scheduling for minimizing the average flow time is NP-hard [10]. It is also NP-hard to optimize the maximum flow objective [10]. Since broadcast scheduling is a special case of the problems considered in this paper, each of the problems we consider is NP-hard. The best known approximation for the average flow in broadcast scheduling is a $O(\log^2 n/\log\log n)$-approximation [4]. For the k-norms objective, [14] gave the first scalable algorithm. For the problem of minimizing the maximum flow time, [12] gave a 2-approximation.

2 Formal Problem Definition and Notation

There are n pages of information that are available at the server. Each request $J_{p,i}$ arrives at an integer time $r_{p,i}$, and this is the first time when the scheduler is aware of this request. Here we use $J_{p,i}$ to denote the request for page p with the ith earliest arrival time; ties are broken arbitrarily but a fixed way. For any p and i, we will say that the two requests $J_{p,i}$ and $J_{p,i+1}$ are adjacent. Each page p has size ℓ_p. That is, the page p consists of ℓ_p pieces of unit sized information, $(p,1),(p,2),...,(p,\ell_p)$. At each time t (or equivalently between time t and $t+1$), the server can transmit only one piece of information, and further can service up to B_p requests when transmitting a piece of information for page p. Note unlike in broadcast scheduling that the server needs to decide which requests to serve if there are more than B_p outstanding requests. A request $J_{p,i}$ is satisfied at the first time when the request, since its release time $r_{p,i}$, receives all pieces of information $(p,1),(p,2),...,(p,\ell_p)$ in the *sequential* order. However, such a sequence of transmissions does not have to be continuous, and can be interrupted by other transmissions. We let $C_{p,i}$ denote the completion time of request $J_{p,i}$. We will distinguish two cases depending on the pages sizes: unit sized pages and varying sized pages. In the unit sized pages case, $\ell_p = 1$ for all pages p, and hence does not have to decide which piece of information to transmit.

In all objectives we consider in this paper, the server is required to satisfy all requests. As mentioned before, k-norms of flow time is defined as $\sqrt[k]{\sum_{p,i}(C_{p,i}-r_{p,i})^k}$. This objective becomes the total (or equivalently average) flow time and the maximum flow time when $k=1$ and $k=\infty$, respectively.

When the algorithm is given speed $(1+\epsilon)$, we assume that the algorithm is allowed to transmit an extra piece of information at every $1/\epsilon$ time steps; here $1/\epsilon$ is assumed to be an integer. When the algorithm is of η-capacity, it can serve up to ηB_p requests of the same page p.

3 Preliminaries and Overview of the Analysis

For the objectives of minimizing the average flow time and the k-norms of flow time, we will first show that an online algorithm is fractionally scalable. The main analysis tool for this is potential functions. Potential functions have become popular for scheduling algorithms and their analysis. See [18] for a tutorial. Then we convert the fractional schedule into an integral one with additional speed augmentation. In this

section, we first define a fractional schedule, and give a quick overview of potential function based analysis. Also we will briefly discuss the online conversion from a fractional schedule into an integral one. The maximum flow time objective completely differs from the analysis for the k norms, and hence will be discussed on its own in Appendix C.

Fractional Flow Time: In the fractional schedule, there are no different pieces of information for a page. Hence when defining a fractional schedule, we only need to specify which page is being processed at a time together with the requests that are being satisfied. Up to B_p requests for page p can be grouped in a batch for page p. Then the (fractional) completion time $C_{p,i}$ of a request $J_{p,i}$ is defined as the first time $t > r_{p,i}$ when $J_{p,i}$ has been included in a batch for page for a total of ℓ_p during $[r_{p,i}, t]$. Note that a real (integral) schedule σ also defines the fractional completion time $C^{f,\sigma}_{p,i}$ of a request $J_{p,i}$, and is no bigger than its integral completion time $C^{\sigma}_{p,i}$. To distinguish fractional schedule from integral schedule, we will say that a page is 'processed' rather than 'broadcasted.'

The advantage of fractional flow time is two-fold. Firstly, this makes potential function based analysis more applicable–Particularly, in the analysis of the continuous changes of the potential function. Secondly, this allows us not to worry about restarting a broadcast due to newly arriving requests for the same page. To see this, consider an alive request for page p that has been partially processed. Now when new requests arrive for the same page p, we need to decide whether to start broadcasting page p from the beginning or resume from the piece just after that by which the old request has been satisfied.

Potential Functions: We show that our online algorithms are fractionally scalable for average flow and the k-norms of flow using potential functions. The potential function $\Phi(t)$ changes over time, and have non-continuous and continuous changes. Non-continuous changes can occur when request arrive or are completed by our algorithm or the optimal scheduler. Continuous changes can occur when our algorithm or the optimal scheduler process some pages. We can without loss of generality compare our algorithm's fractional cost to the optimal scheduler's fractional cost, rather than to the optimal scheduler's integral cost, since the fractional cost can be only smaller. The continuous change can occur also due to time elapse (this will only be the case for the k-norms of flow time). We will show that the potential function satisfies the following properties:

- $\Phi(0) = \Phi(\infty) = 0$.
- All non-continuous changes of $\Phi(t)$ are non-positive.
- For some constants $c_1, c_2 > 0$, at any time t, $\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq -c_1 \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{COST}_A(t) + c_2 \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{COST}_{\mathrm{OPT}}(t)$.

Here $\mathrm{COST}_A(t)$ denotes the total cumulative cost of our algorithm at time t. The $A(t)$ be the unsatisfied requests for page p at time t in the schedule A. Here $\mathrm{COST}_A(t) = \sum_{J_{p,i} \in A(t)} (t - r_{p,i})$ for the total flow time objective, and note that $\mathrm{COST}_A(\infty)$ is the final objective. For the k-norm objective, we let $\mathrm{COST}_A(t) := \sum_{J_{p,i} \in A(t)} k(t - r_{p,i})^{k-1}$ and $\sqrt[k]{\mathrm{COST}_A(\infty)}$ is the final objective in this case. Likewise, $\mathrm{COST}_{\mathrm{OPT}}(t)$ denotes the optimal scheduler's cumulative cost at time t. For more details of the usage of potential functions, see the survey [18]. If the above constraints are satisfied, we can easily show that $\mathrm{COST}_A(\infty) \leq \frac{c_2}{c_1}\mathrm{COST}_{\mathrm{OPT}}(\infty)$.

Online Conversion into an Integral Schedule: We give an online algorithm that converts into a fractional schedule into an integral schedule that increases the flow time of each request only by a constant factor. To do this, when a request $J_{p,i}$ is fractionally completed, we insert the request into a queue together with its lag $\rho_{p,i} := C^f_{p,i} - r_{p,i}$. We will give an online algorithm that satisfies all requests $J_{p,i}$ within a constant times their lag since they enter the queue. Here the lags are carefully used to prioritize the pages to broadcast and which request should be included in a batch. We will give two conversion algorithms, one for unit-size pages, and one for varying-sized pages. The latter will require more relaxation on the batch size. We note that if the flow time of any request increases by at most a constant factor, for any of the objectives

we consider, the overall objective of the integral schedules is at most a constant factor larger than for the fractional schedule.

Optimal Schedule's Structure: We make the following observation regarding the fractional optimal schedule's structure .Note that the following lemma holds for the k-norms of flow time as well as maximum flow time and average flow time.

NOTE: I do not want to consider only fractional schedulers because the FIFO proof needs this lemma and the proof only works in the sequential (non-beffering or fractional) model

Lemma 3.1. *Consider any k norm of flow time for $k \in \{1, 2, 3, ..., \infty\}$. There exists an optimal solution in the integral model where each request $J_{p,i}$ receives the pieces (p, j) of page p before $J_{p,i+1}$ receives (p, x) for all p, i, x. That is, requests are processed in first-in-first-out order.*

Proof. Among all possible optimal schedules in the integral model, consider a schedule σ that minimizes the number of inversions against the first-in-first-order. Here we say that a pair of requests $J_{p,i}$ and $J_{p,j}$, $i < j$, and a piece of information (p, k) of page p, incur an inversion if the later arriving request $J_{p,j}$ receives the kth piece of information earlier than $J_{p,i}$. We show that there is in fact no inversion in σ. For the sake of contradiction, suppose not. Fix any pair of requests $J_{p,i}$, $J_{p,j}$, that have an inversion. Consider changing the way $J_{p,i}$ and $J_{p,j}$ are satisfied in this schedule. For each piece (p, x) of page p, let $J_{p,i}$ be included in the first batch where one of $J_{p,i}$ or $J_{p,j}$ receives (p, x). Let $J_{p,j}$ be in the latest batch either $J_{p.j}$ and $J_{p,i}$ receive (p, x). Notice that this is a valid schedule since $r_{p,i} \leq r_{p,j}$. Further, since the objective function is convex, this can only reduce the objective and reduces the number of inversion, which is a contradiction to the definition of σ. □

Thus, we can assume that the optimal solution satisfies requests in first-in-first-out order.

Organization: In Section 4, we present our results for average flow time and prove Theorem 1.1 and 1.3. To do this, we will first show that a natural variant of the algorithm Latest Arrival Processor Sharing (LAPS) [15, 5] is fractionally scalable. We complete the algorithm and analysis by providing the online conversions into an integral schedule. As mentioned, the two conversions are for the unit-sized pages and varying-sized pages cases, respectively. Due to the space constraints, we defer the analysis of latter conversion to Appendix A. The k-norms results and the maximum flow result are presented in Appendix B and C, respectively.

4 Average Flow Time

In this section we consider the problem of minimizing the average flow time of the requests. As mentioned, we will consider the objective of fractional flow time first. Later we will show how to convert this to an integral schedule. For the conversion we will consider the unit sized page case and the varying sized page case separately. However, for the fractional case, we will assume that pages can possibly have varying sizes.

4.1 Fractional Flow Time

4.1.1 Algorithm

The algorithm we use is an adaptation of the algorithm LAPS [15, 5]. Let $0 < \epsilon < 1/10$ be a fixed constant. We will assume that the algorithm is given $s = (1 + 10\epsilon)$-speed. The algorithm essentially distributes its processing power amongst the the latest ϵ fraction of the latest arriving requests. In particular, let $A(t)$ denote the set of unsatisfied requests at time t in LAPS's schedule. Let $A'(t)$ be the set of $\lceil \epsilon |A(t)| \rceil$ latest arriving requests in $A(t)$. Let $J_{q,k}$ be the earliest arriving request in $A'(t)$. Let $h_{p,i}(t) = s/(\epsilon |A(t)|)$ for each request $J_{p,i} \in A'(t) \setminus J_{q,k}$. For the request $J_{q,k}$, let $h_{q,k}(t) = s(\epsilon |A(t)| - \lceil \epsilon |A(t)| \rceil + 1)/(\epsilon |A(t)|)$. Intuitively, $h_{p,i}$ denotes the share of processing power request $J_{p,i}$ receives. Note that only the requests in

$A'(t)$ receive processing power at time t. It can be seen for all times t that $\sum_{J_{p,i} \in A'(t)} h_{p,i}(t) = s$, the speed the algorithm is given.

Now we need to define how the processing power is actually distributed at time t. We assume that our algorithm is 2-factor-capacity, i.e. can serve up to $2B_p$ requests for page p when broadcasting page p. For each request $J_{p,i} \in A'(t)$, let $Q_{p,i}(t) = \{J_{p,j} \in A(t) \mid i - B_p + 1 \leq j \leq i + B_p - 1\}$. The is a 'ball' of at most $2B_p$ requests that arrived the closed to request $J_{p,j}$. This is the set of requests $J_{p,i}$ can effect. Now each request in $Q_{p,i}$ is processed at a rate of $h_{p,i}(t)$ due to (the share of) request $J_{p,i}$. Thus, a request $J_{p,j} \in A(t)$ for page p is processed at a total rate of $\sum_{J_{p,i} \in A'(t), J_{p,j} \in Q_{p,i}(t)} h_{p,i}(t)$ at time t.

4.1.2 Analysis

In this section we analyze the algorithm LAPS. Let OPT denote some fixed optimal solution for the *fractional* setting. For the sake of analysis, we define a potential function that satisfies the three properties described in Section 3. To this end, we need to define some notation. Let $\textsc{On}(t_1, t_2, p, i) = \int_{t=t_1}^{t_2} h_{p,i}(t)$ be the total amount WLAPS devotes to request $J_{p,i}$ during $[t_1, t_2)$. Let $\tau_{p,i}$ be the first time that OPT works on request $J_{p,i}$. Let $\textsc{Opt}(t_1, t_2, p)$ denote the amount OPT processes page p during $[t_1, t_2)$ up to a maximum of ℓ_p. Now we create the following variable for each request $J_{p,i}$,

$$z_{p,i}(t) = \textsc{On}(t, \infty, p, i) \cdot \textsc{Opt}(\tau_{p,i}, t, p)$$

Now we are almost ready to define the potential function. Let $\textsc{rank}(p, i, t) = \sum_{J_{q,k} \in A(t), r_{q,k} \leq r_{p,i}} 1$ be the total number of alive requests in $A(t)$ that arrived earlier than $J_{p,i}$. Now our potential function is,

$$\Phi(t) = \frac{5}{\epsilon} \sum_{J_{p,i} \in A(t)} \textsc{rank}(p, i, t) \frac{z_{p,i}(t)}{\ell_p}$$

Recall from Section 3 the properties we need to show. The first property $\Phi(0) = \Phi(\infty) = 0$. We now bound each of the possible changes in the potential function separately below.

Arrival Condition: Consider when $J_{p,i}$ arrives at time t. Note that when a job arrives, the rank of every other job remains unchanged. Further the term $\textsc{rank}(p, i, t) \frac{z_{p,i}(t)}{\ell_p}$ is added to the potential function. However, it can be seen that $z_{p,i}(t) = 0$ at the time t when $J_{p,i}$ arrives since OPT has not had a chance to work on this request yet.

Completion Condition: When a request $J_{p,i}$ is completed at time t then the rank of all jobs that arrived later that $J_{p,i}$ could possibly decrease. However, since the potential function is always positive, it can be seen that this can only decrease the potential function. Also the term $\textsc{rank}(p, i, t) \frac{z_{p,i}(t)}{\ell_p}$ is removed from the potential. Again, this can only decrease the potential.

We have shown that no non-continuous change is positive. We now focus on continuous changes of $\Phi(t)$ that can occur due to our algorithm and OPT's processing.

Running Condition: We first consider the case where OPT processes pages. We can without loss of generality assume that OPT processes at most one page at time t. Say that OPT processes page p. Let $O^*(t)$ be the set of requests which OPT satisfies by working on page p. Note that $|O^*(t)| \leq B_p$ by definition of page p's batch size. Further, note that O^* contains at most B_p 'adjacent' requests due to the FIFO-nature of OPT proven in Lemma 3.1. Also note that when page p is processed by OPT, the $z_{p,i}$ variable remains unaffected for all requests $J_{p,i}$ where $t < \tau_{p,i}$ and for all requests $J_{p,i}$ that have been satisfied by OPT. The first case is because OPT does not start working on such requests by definition of $\tau_{p,i}$, and hence $\textsc{Opt}(\tau_{p,i}, t, p)$ for those requests does not increase. The second case is because $\textsc{Opt}(\tau_{p,i}, t, p)$ can be at most ℓ_p and it must be ℓ_p if OPT completed request $J_{p,i}$, so $\textsc{Opt}(\tau_{p,i}, t, p)$ does not increase at time t.

Now, for any request $J_{p,i} \in A(t) \cap O^*(t)$ where $\tau_{p,i} \leq t$ the variable $z_{p,i}$ will increase at a rate of $\textsc{On}(t, \infty, p, i)$. We bound this total increase in the following lemma.

Lemma 4.1. *If* OPT *processes* p *at time* t *then* $\sum_{J_{p,i}\in A(t)\cap O^*(t),\tau_{p,i}\le t} \textsc{On}(t,\infty,p,i) \le \ell_p$.

Proof. We know that $O^*(t)$ contains at most B_p adjacent requests. Further, by definition of LAPS, if LAPS works on any request in $O^*(t)$ then it satisfies any other request in $O^*(t)$; here the relaxation that LAPS is of 2-factor-capacity is crucially used. However, this implies that $\sum_{J_{p,i}\in A(t)\cap O^*(t),\tau_{p,i}\le t} \textsc{On}(t,\infty,p,i)$ can be at most ℓ_p, because all the requests in $O^*(t)$ will be completed by WLAPS schedule if ℓ_p amount of processing is devoted to the requests in $O^*(t)$ for page p. □

With this lemma and by the fact that the rank of any job is at most $|A(t)|$, we have the total increase in the potential due to OPT's processing is at most

$$\frac{5}{\epsilon}\sum_{J_{p,i}\in A(t)\cap O^*(t),\tau_{p,i}\le t}\frac{\textsc{On}(t,\infty,p,i)}{\ell_p}\textsc{rank}(p,i) \le \frac{5}{\epsilon}|A(t)| \tag{1}$$

Now we bound the change in the potential due to the algorithm's processing of pages. Notice that for each request $J_{p,i} \in A'(t)$ it is the case that $\textsc{On}(t,\infty,p,i)$ decreases at a rate of $h_{p,i}(t)$. Thus, $z_{p,i}(t)$ decreases at a rate of $h_{p,i}(t)\textsc{Opt}(\tau_{p,i},t,p)$. Also, notice that for any request $J_{p,i}$, it is the case that $\textsc{Opt}(\tau_{p,i},t,p) = \ell_p$ if OPT has competed job $J_{p,i}$ by time t. Let $O(t)$ be the set of alive jobs in OPT's schedule at time t. The decrease in $\Phi(t)$ due to the algorithm's processing is the following.

$$\begin{aligned}
& \frac{5}{\epsilon}\sum_{J_{p,i}\in A'(t)} \textsc{rank}(p,i,t)\frac{h_{p,i}(t)\textsc{Opt}(\tau_{p,i},t,p)}{\ell_p} \\
\ge\ & \frac{5}{\epsilon}\sum_{J_{p,i}\in A'(t)} (1-\epsilon)|A(t)|\frac{h_{p,i}(t)\textsc{Opt}(\tau_{p,i},t,p)}{\ell_p} \quad [(1-\epsilon)|A(t)| \le \textsc{rank}(p,i,t) \text{ for all } J_{p,i}\in A'(t)] \\
\ge\ & \frac{5}{\epsilon}(1-\epsilon)|A(t)|\sum_{J_{p,i}\in A'(t)\setminus O(t)} h_{p,i}(t) \quad [\ \textsc{Opt}(\tau_{p,i},t,p) = \ell_p \text{ for all requests } J_{p,i}\notin O(t)] \\
\ge\ & \frac{5}{\epsilon}(1-\epsilon)|A(t)|\left(\sum_{J_{p,i}\in A'(t)} h_{p,i}(t) - \sum_{J_{p,i}\in O(t)\cap A'(t)} h_{p,i}(t)\right) \\
\ge\ & \frac{5}{\epsilon}(1-\epsilon)|A(t)|\left(s - \sum_{J_{p,i}\in O(t)\cap A'(t)} h_{p,i}(t)\right) \quad [\text{Since } \textstyle\sum_{J_{p,i}\in A'(t)} h_{p,i}(t) = s] \\
\ge\ & \frac{5}{\epsilon}(1-\epsilon)|A(t)|\left(s - \sum_{J_{p,i}\in O(t)\cap A'(t)} \frac{1}{\epsilon|A(t)|}\right) \quad [h_{p,i}(t) \le 1/(\epsilon|A(t)|) \text{ by definition of LAPS}] \\
\ge\ & \frac{5}{\epsilon}(1-\epsilon)s|A(t)|s - \frac{5s}{\epsilon^2}|O(t)|
\end{aligned}$$

By combining this with (1), we derive the last property we want to show.

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) &\le \frac{5}{\epsilon}|A(t)| - \frac{5}{\epsilon}(1-\epsilon)s|A(t)|s + \frac{5s}{\epsilon^2}|O(t)| \\
&\le \frac{5}{\epsilon}|A(t)| - \frac{5}{\epsilon}(1-\epsilon)(1+10\epsilon)|A(t)| - \frac{10}{\epsilon^2}|O(t)| \quad [s = (1+10\epsilon) \text{ by definition and } s \le 2] \\
&\le -|A(t)| + \frac{10}{\epsilon^2}|O(t)|
\end{aligned}$$

Since $|A(t)$ and $|O(t)|$ are the instantaneous increase in the LAPS and OPT' objectives, respectively, we conclude that our algorithm LAPS is $(1+10\epsilon)$-speed $\frac{10}{\epsilon^2}$-competitive.

4.2 Unit Sized Page Conversion

In this section we show how to take the algorithm we presented for minimizing average flow time in fractional batch scheduling and construct an online algorithm for the integral batched scheduling such that the integral schedule has a small amount of extra resource augmentation and the no request's flow time increases by more than a constant factor. We note that this conversion also applies to our algorithm we introduce for the k-norms of flow time presented in Appendix B. To show this, let A be the algorithm for fractional batch scheduling. Say that A is given $(1+\epsilon)$ speed where $0 < \epsilon < 1$. Note that we assumed that A has $(1+10\epsilon)$ in the previous section, but we can assume it has $1+\epsilon$ speed by scaling ϵ. We construct an algorithm A' for integral batch scheduling. We will assume that A' can schedule one unit sized pages in a time step and four extra unit size pages every $\lceil 1/\epsilon \rceil$ time steps. Note that this is essentially equivalent to A' having $1+4\epsilon$ speed. For the conversion, we do not consider any arbitrary algorithm that performs well for the fractional objective because we will need some key properties of A to perform the conversion. In particular, the algorithm A is required to group two requests $J_{p,i}$ and $J_{p,j}$ in the same batch only if $i - B_p + 1 \leq j \leq i + B_p - 1$. More precisely, if this condition is not satisfied, no share that one of the two requests receive cannot be used to satisfy the other request. This is the way our algorithms are described for both average flow time and the k-norms of flow time.

For our online conversion algorithm, we maintain a queue Q. Whenever a request $J_{p,i}$ is finished by A in the fractional schedule, it is added to the queue Q, and waits to be completed in the integral schedule. We let $\rho_{p,i}$ denote the flow time of request $J_{p,i}$ in A's schedule, and call $\rho_{p,i}$ the lag of request $J_{p,i}$. At any time, the algorithm A' finds (if any) a request $J_{p,i}$ with the smallest lag $\rho_{p,i}$, and transmits page p. The batch satisfied by this transmission will be all the requests $J_{p,j}$ such that $i - B_p + 1 \leq j \leq i + B_p - 1$. Note that the batch size is at most $2B_p$. We will say that $J_{p,i}$ *forces* A' to schedule page p at this time.

Now our goal is to show that any request $J_{p,i}$ is completed within $\frac{10}{\epsilon}\rho_{p,i}$ time steps in A''s schedule. This will then imply that the average flow time of the requests in the schedule A' is at most a factor $\frac{10}{\epsilon}$ larger than that for A.

Theorem 4.2. *Any request $J_{p,i}$ is satisfied by time $r_{p,i} + \frac{10}{\epsilon}\rho_{p,i}$ in A'.*

For the sake of contradiction, suppose that there exists a request $J_{q,k}$ that has a flow time greater than $\frac{10}{\epsilon}\rho_{p,i}$ in the scheduler of A'. Let t' be the first time that $J_{q,k}$ has waited more than $\frac{10}{\epsilon}\rho_{p,i}$ time steps. Let t_1 be the earliest time before time t' such that every request that forces A' to schedule during $[t_1, t']$ has lag at most $\rho_{p,i}$. Let N_p be the set of requests that forced A' to schedule page p during $[t_1, t]$. Our first goal is to show that no two requests in N_p can be satisfied at any point simultaneously in A.

Lemma 4.3. *For all p, any two requests in N_p cannot be satisfied simultaneously at any point in A.*

Proof. Consider any page p and two distinct requests $J_{p,i}, J_{p,j}$ in N_p. We assume without loss of generality that $J_{p,i}$ forces A' to schedule p before $J_{p,j}$. Let t^* be the time A' satisfies $J_{p,i}$. There are two cases. In the first case, say that $r_{p,j} \leq t^*$. In this case, the only reason $J_{p,j}$ is not satisfied is that either $j < i - B_p + 1$ or $j > i + B_p - 1$. That is, $J_{p,j}$ is not included in the batch with $J_{p,i}$. However, we know that in this case $J_{p,i}$ cannot satisfied simultaneously at any point in A by definition of A. For the second case, assume that $r_{p,j} > t^*$. Note that $t^* \geq r_{p,i} + \rho_{p,i}$ since $J_{p,i}$ is added to to the queue Q at the time $r_{p,i} + \rho_{p,i}$ when $J_{p,i}$ is completed in A. Hence no processing of $J_{p,i}$ in A cannot be used to satisfy $J_{p,j}$. □

Next we show that any request that forces A' to schedule during $[t_1, t']$ did not arrive too early.

Lemma 4.4. *For any request $J_{p,i}$ that forces A' to schedule during $[t_1, t']$, it is the case that $r_{p,i} \geq t_1 - \rho_{q,k}$.*

Proof. For the sake of contradiction, suppose that there is a request $J_{p,i}$ that arrived before $t_1 - \rho_{q,k}$ and forced A' to schedule during $[t_1, t']$. By definition of A', $J_{p,i}$ is available to be scheduled in A' at time $r_{p,i} + \rho_{p,i}$. Note from the definition of t_1 that $\rho_{p,i} \leq \rho_{p,k}$, and hence $r_{p,i} + \rho_{p,i} \leq r_{p,i} + \rho_{p,k} < t_1$. However, this implies that every request that forced to schedule a page during $[r_{p,i} + \rho_{p,i}, t_1]$ has lag less than $\rho_{p,i} \leq \rho_{q,k}$. This contradicts the definition of t_1. □

Finally, we will show the following theorem about A'.

Proof of [Theorem 4.2] We know that A' schedules a total of $(1+4\epsilon)(t'-t_1)$ pages during $t'-t_1$. Further, by Lemma 4.3 these requests cannot be satisfied simultaneously in A'. We know by Lemma 4.4 that these requests arrive no earlier than $t_1 - \rho_{q,k}$. Finally, we know that these requests must be completed by time t', since no request can force A' to schedule until they are complete in A. Thus, A must do a volume of $(1+4\epsilon)(t'-t_1)$ work during $[t_1 - \rho_{q,k}, t']$. This implies that $(1+4\epsilon)(t'-t_1) \leq (t'-t_1+\rho_{q,k})$. However, we know that $t_1 \leq r_{q,k} + \rho_{q,k}$ by definition of t_1 and $t' - r_{q,k} \geq \frac{10}{\epsilon}\rho_{q,k}$ by definition of $J_{q,k}$. However, this is a contradiction and we derive the theorem. □

4.3 Varying Size Page Conversion

In this section we show how to convert a fractionally good schedule into an integral schedule with a small amount of extra speed augmentation so that no request's flow time increases by more than a constant factor. Like the conversion presented in Section 4.2, this conversion can be used for the average flow objective and also for the more general k-norms of flow objective. However, this conversion is more general in that it works for varying sized pages. However, this requires more relaxation concerning batch size–we will allow our online conversion can use a batch of size up to $6B_p$ for all pages p. Furthermore, we restrict that in the input fractional schedule, two requests $J_{p,i}$ and $J_{p,j}$ can be in the same batch only if $i - B_p + 1 \leq j \leq i + B_p - 1$, which will be crucial in performing the conversion. Due to the space constraints, we here present only the conversion algorithm. The full analysis can be found in Appendix A.

To describe this new conversion, we first set up some notation. Let A denote the fractional input schedule. For simplicity, the reader may assume that A refers to the fractional algorithm we give in Section 4.1 or Appendix B (or more precisely the resulting fractional schedule). Say that A is given $(1+\epsilon)$ speed where $0 < \epsilon < 1$ (We assumed that A has $(1+10\epsilon)$ previously, but we can assume it has $1+\epsilon$ speed by scaling ϵ). We construct an algorithm A' for integral batch scheduling. We will assume that A' can schedule one unit sized page in a time step and four extra unit size pages every $\lceil 1/\epsilon \rceil$ time steps. Note that this is essentially equivalent to A' having $1+4\epsilon$ speed. We will show in in Appendix A that any request $J_{p,i}$ is completed within $\frac{10}{\epsilon}\rho_{p,i}$ time steps in A''s schedule.

For each request $J_{p,i}$ let $\rho_{p,i}$ denote the flow time of request $J_{p,i}$ in A's schedule. We will call $\rho_{p,i}$ the *lag* of request $J_{p,i}$. We group requests for page p seamlessly into groups of size $2B_p$ based on the order in which they arrive. More precisely, for each page p, let $\mathcal{B}_p^i$ contain the requests $J_{p,j}$ for page p such that $2iB_p + 1 \leq j \leq 2(i+1)B_p$. For each request $J_{p,i}$, we define the *start time* for $J_{p,i}$ to be the last time $J_{p,i}$ was included in a batch where the first piece of page p was transmitted. Note that a request's start time can 'restart' if the first piece of page p is transmitted again and $J_{p,i}$ is included in the batch. Starting times will be crucial in our analysis. Also we need to make sure that each request keeps track of the next piece of information it needs to receive.

More formally, each request $J_{p,i}$ is associated with a tuple $\langle p, i, s, a\rangle$ where s denotes the start time for $J_{p,i}$ and a denotes the next piece of page p that $J_{p,i}$ will need to receive. Initially, $s = \infty$ and $a = 1$ when $J_{p,i}$ arrives. Now at time t, the algorithm A' finds the request $J_{p,i}$ which has the smallest lag amongst those requests which have been completed in the schedule A. The algorithm is going to broadcast a piece of page p at this time. However, we need to be careful about which requests are satisfied in the current batch as well as which piece of page p is broadcasted. Let $\langle p, i, s, a\rangle$ be the tuple associated with $J_{p,i}$. Then, the ath piece of page p is broadcasted at time t. If $a \geq 2$ then the requests in the batch are all those for page p with start

time s. For each of these requests, we increment the a value in the tuple so that it wait for the following piece of information of page p. If $a = 1$ then say that $J_{p,i} \in \mathcal{B}_p^h$ for some h. The batch will be all those requests for p in $\mathcal{B}_p^{h-1} \cup \mathcal{B}_p^h \cup \mathcal{B}_p^{h+1}$ that are currently alive and unsatisfied in A'. Any request $J_{p,j}$ that is in the batch, sets its start time s to t and the next piece a to receive to 2 (here 'restart' can happen). Note that a request in the batch could already have a finite start time, but in this case it just resets its start time no matter how much of page p it has already received. At time t, we say that $J_{p,i}$ forces A' to schedule.

See Appendix A for the analysis of this algorithm.

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. R x W: 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. In *ICALP '10: Proceedings of the Thirty-seventh International Colloquium on Automata, Languages and Programming*, pages 324–335. Springer, 2010.

[6] Nikhil Bansal and Kirk Pruhs. Server scheduling to balance priorities, fairness, and average quality of service. *SIAM J. Comput.*, 39(7):3311–3335, 2010.

[7] Amotz Bar-Noy, Sudipto Guha, Yoav Katz, Joseph (Seffi) Naor, Baruch Schieber, and Hadas Shachnai. Throughput maximization of real-time scheduling with batching. *ACM Trans. Algorithms*, 5(2):18:1–18:17, March 2009.

[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. SIAM, 2000.

[9] Ho-Leung Chan, Jeff Edmonds, and Kirk Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *SPAA '09: Proceedings of the Twenty-first Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–10, 2009.

[10] 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, 2008.

[11] Chandra Chekuri, Avigdor Gal, Sungjin Im, Samir Khuller, Jian Li, Richard Matthew McCutchen, Benjamin Moseley, and Louiqa Raschid. New models and algorithms for throughput maximization in broadcast scheduling - (extended abstract). In *WAOA '10: Proceedings of the Eighth Workshop on Approximation and Online Algorithms*, pages 71–82. Springer, 2010.

[12] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Online scheduling to minimize maximum response time and maximum delay factor. *Theory of Computing*, 8(7):165–195, 2012.

[13] 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.

[14] Jeff Edmonds, Sungjin Im, and Benjamin Moseley. Online scalable scheduling for the ℓ_k-norms of flow time without conservation of work. In *SODA '11: Proceedings of the Twenty-first Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 109–119, 2011.

[15] 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, 2009.

[16] Regant Y. S. Hung and Hing-Fung Ting. Design and analysis of online batching systems. *Algorithmica*, 57(2):217–231, 2010.

[17] 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.

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

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

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

[21] Jae-Hoon Kim and Kyung-Yong Chwa. Scheduling broadcasts with deadlines. *Theor. Comput. Sci.*, 325(3):479–488, 2004.

[22] M. Mathirajan and A.I. Sivakumar. A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology*, 29(9-10):990–1001, 2006.

[23] Kirk Pruhs, Julien Robert, and Nicolas Schabanel. Minimizing maximum flowtime of jobs with arbitrary parallelizability. In *WAOA '10: Proceedings of the Eighth Workshop on Approximation and Online Algorithms*, pages 237–248, 2010.

[24] J. Wong. Broadcast delivery. *Proceedings of the IEEE*, 76(12):1566–1577, 1988.

A Analysis for the Online Conversion to an Integral Schedule for Varying Sized Pages

In this section, we analyze the online conversion algorithm we presented in Section 4.3, and prove the following theorem.

Theorem A.1. *In the schedule A', any request $J_{p,i}$ is satisfied by time $r_{p,i} + \frac{10}{\epsilon}\rho_{p,i}$.*

For the sake of contradiction, suppose that there exists a request $J_{q,k}$ that waits longer than $\frac{10}{\epsilon}\rho_{q,k}$ time in A' to be satisfied. Let t' be the first time that $J_{q,k}$ has waited more than $\frac{10}{\epsilon}\rho_{q,k}$ time steps. Let t_1 be the earliest time before time t such that every request that forces A' to schedule during $[t_1, t']$ has lag at most $\rho_{q,k}$. Let N_p be the set of requests that forced A' to schedule page p during $[t_1, t]$. Let S_p be the set of start times for page p which occur during $[t_1, t']$. Notice that $|N_p| = |S_p|$. It is important to note that not every request which forces A' to process a page during $[t_1, t']$ has a start time during this interval. In particular, some requests can start being satisfied in A''s schedule before they are completed in A's schedule. Recall that they cannot force A' to schedule until they are completed by A. It is this set of requests that most of the proof will need to focus on. Let N'_p be the set of requests that force A' to schedule during $[t_1, t']$ whose start time is before t_1, and let S'_p contain this start time. Note that S'_p particularly contains the last time a request in N'_p was started before t_1.

Our proof can be summarized as follows. We will first show that the total volume of work A' does during $[t_1, t']$ is $(1 + 4\epsilon)(t' - t_1) \leq \sum_p |S_p|\ell_p + \sum_p |S'_p|\ell_p$. We will then show that the total amount of work A must do during $[t_1 - \rho_k, t']$ is $\sum_p |S_p|\ell_p$, and $\sum_p |S'_p|\ell_p \leq 8\rho_k$. Given that A' has more speed than A this will be sufficient to derive a contradiction to $J_{q,k}$'s large waiting time.

Proposition A.2. *Consider any start time s in $S_p \cup S'_p$, and count the total number of pieces of page p that are scheduled during $[t_1, t']$ when a request with the final start time s forces A' to broadcast. Then the total number is at most ℓ_p.*

Proof. This proposition easily follows from definition of A'. All requests for page p having the same (final) start time s are started (or restarted) at the same time s, and continue to be satisfied altogether when one of those requests force A' to broadcast a piece of page p. In other words, the piece is 'right' one for all those requests. □

Lemma A.3. *Any request that forces A' to schedule during $[t_1, t']$ arrives at earliest $t_1 - \rho_{q,k}$.*

Proof. For the sake of contradiction say that there is a request $J_{p,i}$ that arrived before $t_1 - \rho_{q,k}$ and forced A' to schedule during $[t_1, t']$. By definition of A', $J_{p,i}$ is available to be scheduled in A' at time $r_{p,i} + \rho_{p,i} \leq r_{p,i} + \rho_{q,k} < t_1$. However, this implies that every request scheduled during $[r_{p,i} + \rho_{p,i}, t_1]$ has lag less than $\rho_{p,i} \leq \rho_{q,k}$. This contradicts the definition of t_1. □

Lemma A.4. *Any two distinct requests $J_{p,i}$ and $J_{p,j}$ in N_p cannot be satisfied at any point simultaneously in A.*

Proof. Fix some page p and consider any two distinct requests $J_{p,i}, J_{p,j} \in N_p$. Without loss of generality, assume that the first time $J_{p,i}$ forces A' to schedule p is before the first time $J_{p,j}$ forces A' to schedule. Let t^* be the first time A' is forced to be scheduled by $J_{p,i}$. There are two cases. In the first case, say that $r_{p,j} \leq t^*$. Say that $J_{p,i}$ is in $\mathcal{B}_p^{i'}$. In this case, the only reason $J_{p,j}$ is not started at time t^* is that $J_{p,j} \notin \mathcal{B}_p^{i'-1} \cup \mathcal{B}_p^{i'} \cup \mathcal{B}_p^{i'+1}$. However this implies that either $j < i - B_p + 1$ or $j > i + B_p - 1$. That is, $J_{p,j}$ is not included in the batch with $J_{p,i}$ at any point in A's schedule.

Now consider the second case that $r_{p,j} > t^*$. We know that $t^* \geq r_{p,i} + \rho_{p,i}$ since no request can force A' to schedule until it is completed in A. However, we know that A completes $J_{p,i}$ at time $r_{p,i} + \rho_{p,i}$. This implies that $J_{p,i}$ is completed in A before $J_{p,j}$ arrives. Hence the claim follows. □

Corollary A.5. *The schedule A must broadcast $|S_p|\ell_p$ units of page p during $[t_1 - \rho_{q,k}, t']$ for all pages p.*

Proof. By Lemma A.3 all the requests in N_p arrive at earliest $t - \rho_{q,k}$ and since these requests force A' to schedule during $[t_1, t']$ it must be the case that these requests are completed by time t' in A, by definition of A'. By Lemma A.4 each request in N_p must be satisfied separately in A. The corollary follows from the fact that $|S_p| = |N_p|$. □

Lemma A.6. *It is the case that A schedules at least $|S'_p|\ell_p/2$ units of p during $[t_1 - \rho_{q,k}, t_1 + \rho_{q,k}]$.*

Proof. For each start time $s_j \in S'_p$, let R_p contain the request $J_{p,i}$ if $J_{p,i}$ is the first request to force A' to broadcast at a time t during $[t_1, t']$ which had start time s_j at time t. Our goal is to show that all all times, at most two requests in R_p can be satisfied simultaneously in A and these requests must all be satisfied during $[t_1 - \rho_{q,k}, t_1 + \rho_{q,k}]$. This will then imply the lemma.

Consider any three start times $s_x, s_y, s_z \in S_p$ for any fixed page p and let $J_{p,x}$, $J_{p,y}$ and $J_{p,z}$ be the requests in R_p associated with these start times. First notice that $r_{p,x}, r_{p,y}, r_{p,z} < t_1$ since these requests start being satisfied before t_1. Further, these requests arrive after $t_1 - \rho_{q,k}$ by Lemma A.3. Finally, these requests have lag at most $\rho_{q,k}$ since they force A' to schedule during $[t_1, t']$. These facts imply that these three requests are satisfied by A during $[t_1 - \rho_{q,k}, t_1 + \rho_{q,k}]$.

To prove the lemma, it suffices to show that these three requests cannot be satisfied at any point simultaneously in A. There are two cases. In the first case say that it is not the case that that there exists some integer j such that $j - B_p + 1 \leq x, y, z \leq j + B_p - 1$. In this case, by definition of A all of these requests cannot be satisfied simultaneously at any point. For the second case say there exists some integer j such that $j - B_p + 1 \leq x, y, z \leq j + B_p - 1$. This implies that there exists some integer j' such that at least two of these requests are in $B_p^{j'}$. Without loss of generality, say that these two requests are $J_{p,x}$ and $J_{p,y}$ and say that $s_x > s_y$. Note that this implies that both $J_{p,x}$ and $J_{p,y}$ are alive and unsatisfied at time s_x. Since these two requests are in $B_p^{j'}$, A' would update the start time of both requests at time s_x. However, then $J_{p,y}$ would not have start s_y when it forces A' to schedule during $[t_1, t']$, a contradiction. □

Corollary A.7. $\sum_p |S'_p|\ell_p \leq 8\rho_{q,k}$.

Proof. By the previous lemma, A schedules at least $|S'_p|\ell_p/2$ units of p during $[t_1 - \rho_{q,k}, t_1 + \rho_{q,k}]$. Knowing that A has speed $(1+\epsilon) \leq 2$, it follows that $\sum_p |S'_p|\ell_p/2 \leq 2(t_1 + \rho_{q,k} - (t_1 - \rho_{q,k})) \leq 4\rho_{q,k}$. □

Finally we are ready to prove the theorem.

Proof of [Theorem A.1] By Proposition A.2 the total volume of work scheduled by A' during $[t_1, t']$ is less than $\sum_p |S_p \cup S'_p|\ell_p$. Knowing that A' is always busy during $[t_1, t']$ by definition of t_1 and A' has speed $(1+4\epsilon)$, we know that $(1+4\epsilon)(t' - t_1) \leq \sum_p |S_p \cup S'_p|\ell_p \leq \sum_p (|S_p| + |S'_p|)\ell_p$. By Corollary A.7, we derive $(1+4\epsilon)(t' - t_1) - 8\rho_{q,k} \leq \sum_p |S_p|\ell_p$.

Also, by Lemma A.5 we know that A does at least a total amount of work $\sum_p |S_p|\ell_p$ during $[t_1 - \rho_{q,k}, t']$. From the fact that A has speed $(1+\epsilon)$, it follows that $(1+\epsilon)(t' - t_1 + \rho_{q,k}) \geq \sum_p |S_p|\ell_p$. Combining this with the above inequality, we obtain that $(1+4\epsilon)(t' - t_1) - 8\rho_{q,k} \leq (1+\epsilon)(t' - t_1 + \rho_{q,k})$, which simplifies to $t' - t_1 \leq \frac{10}{3\epsilon}\rho_{q,k}$. This contradicts to the assumption that $t' - t_1 \geq \frac{10}{\epsilon}\rho_{q,k}$. □

B k-norms of Flow Time

In this section we study the k-norms of flow time. As we did for the average flow objective, we will first show that an online algorithm is scalable in the fractional batch scheduling, and will convert it using the online conversion presented either in Section 4.2 or 4.3. We will deal with the total kth power of flow time, $\sum_{p,i}(C_{p,i} - r_{p,i})^k$, and will take kth root at the end of analysis. We begin with presenting our algorithm for the fractional kth power of flow time.

B.1 Algorithm

We use an adaption of the algorithm Weighted Latest Arrival Processor Sharing (WLAPS) [14]. Let $\beta = \epsilon^{2k-1} > 0$ and $0 < \epsilon < \frac{1}{10}$ be fixed constants. We assume that our algorithm is given $s = 1 + 10\epsilon$ speed. Let $A(t)$ denote the set of unsatisfied requests at time t in WLAPS's schedule. For request $J_{p,i}$, we define $J_{p,i}$'s weight at time t as $w_{p,i}(t) = k(t - a_{p,i})^{k-1}$. Note that $J_{p,i}$'s contribution to the kth power of flow time increase at a rate of $w_{p,i}(t)$ (if $J_{p,i}$ is still alive at time t). Let $w(t) = \sum_{J_{p,i} \in A(t)} w_{p,i}(t)$. Let $A'(t)$ denote the minimum set of alive requests in $A(t)$ with the most recent release times whose total weight is no smaller than $\beta w(t)$. Except the oldest request in $A'(t)$, each request $J_{p,i} \in A'(t)$ is processed at a rate of $h_{p,i}(t) = s\frac{w_{p,i}(t)}{\beta w(t)}$. We will refer to $h_{p,i}(t)$ as $J_{p,i}$'s share. The total share is s, the speed A is given, and the oldest request in $A'(t)$ receives the remaining share. To make our analysis more transparent, we assume that $\sum_{p,i} w_{p,i}(t) = \beta w(t)$, and for all requests $J_{p,i} \in A'(t)$, $h_{p,i}(t) = s\frac{w_{p,i}(t)}{\beta w(t)}$. One can easily remove this simplifying analysis. The only difference of WLAPS from LAPS in Section 4 is that WLAPS distributes the processing power to the recent requests in proportion to their weights.

How the 'share' of each request is used to satisfy other requests remains the same as for LAPS. We assume that our algorithm is 2-capacity, i.e. can serve up to $2B_p$ requests for page p when processing page p. For each request $J_{p,i} \in A'(t)$, let $Q_{p,i}(t) = \{J_{p,j} \in A(t) \mid i - B_p + 1 \leq j \leq i + B_p - 1\}$. This is the set of requests $J_{p,i}$ can effect. Now each request in $Q_{p,i}$ is processed at a rate of $h_{p,i}(t)$ due to (the share of) request $J_{p,i}$. Thus, a request $J_{p,j} \in A(t)$ for page p is processed at a total rate of $\sum_{J_{p,i} \in A'(t), J_{p,j} \in Q_{p,i}(t)} h_{p,i}(t)$ at time t.

B.2 Analysis

In this section we analyze the algorithm WLAPS. The analysis of WLAPS will be very similar to that of LAPS. Let OPT denote some fixed optimal solution for the *fractional* setting. By Lemma 3.1, we can assume that OPT processes requests for the same page in a first-in-first-out order. For the sake of analysis, we define a potential function that satisfies the three properties described in Section 3. To this end, we need to define some notation. Let $\textsc{On}(t_1, t_2, p, i) = \int_{t=t_1}^{t_2} h_{p,i}(t)$ be the total amount WLAPS devotes to request $J_{p,i}$ during $[t_1, t_2)$. Let $\tau_{p,i}$ be the first time that OPT works on request $J_{p,i}$. Let $\textsc{Opt}(t_1, t_2, p)$ denote the amount OPT processes page p during $[t_1, t_2)$ up to a maximum of ℓ_p. Now we create the following variable for each request $J_{p,i}$,

$$z_{p,i}(t) = \textsc{On}(t, \infty, p, i) \cdot \textsc{Opt}(\tau_{p,i}, t, p)$$

Now our potential function is,

$$\Phi(t) = \sum_{J_{p,i} \in A(t)} (t - r_{p,i} + \frac{1}{\epsilon} \sum_{\substack{J_{q,j} \in A(t) \\ r_{q,j} \geq r_{p,i}}} z_{q,j}(t))^k$$

Recall from Section 3 the properties we need to show. The first property $\Phi(0) = \Phi(\infty) = 0$ is satisfied trivially. We now bound each of the possible changes in the potential function separately below.

Arrival Condition: When a request $J_{p,i}$ arrives at time $t = r_{p,i}$, the potential function does not change since $t - r_{p,i} = 0$ and $z_{p,i}(t) = 0$. It follows that $z_{p,i}(t) = 0$ since OPT has not had a chance to work on this request yet.

Completion Condition: When the optimal schedule completes a request, the potential function makes no change. When WLAPS completes a request $J_{p,i}$ the potential function can only decrease, since all terms are positive.

We have shown that no non-continuous change is positive. We now focus on continuous changes of $\Phi(t)$ that can occur due to our algorithm and OPT's processing. Unlike for the average flow objective, we

also need to consider the effect of time elapse. For the sake of notation, define $W_{p,i}(t) := k(t - r_{p,i} + \frac{1}{\epsilon}\sum_{r_{q,j}\in A(t), r_{q,j}\geq r_{p,i}} z_{q,j}(t))^{k-1}$.

Running Condition: Recall that we deal with the kth power of flow time first. Let $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) = \sum_{J_{p,i}\in A(t)} k(t - r_{p,i})^{k-1}\mathrm{dt}$ and let $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) = \sum_{J_{p,i}\in O(t)} k(t - r_{p,i})^{k-1}\mathrm{dt}$. Note that the quantities $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)$ and $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$ are the increase rate of the kth power flow time of BLAPS' schedule and OPT's, respectively.

Our goal is to show that during a time interval $[t, t + \mathrm{dt}]$ when no jobs arrive or are completed:

$$\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq (1+\frac{1}{\epsilon}\sum_{J_{p,i}\in A(t)} W_{p,i}(t) - \frac{s(1-\epsilon)}{\epsilon}(1-\beta(1+\frac{1}{\epsilon})^{k-1})\sum_{J_{p,i}\in A(t)} W_{p,i}(t) + -\frac{s(1-\epsilon)}{\epsilon}(\frac{2}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) \tag{2}$$

Then by a simple algebra, and the following lemma and proposition, we can show that BLAPS $\leq \frac{2^{k+1}}{\epsilon^{3k}}$OPT. By taking the kth root, and applying the online conversions, we derive Theorem 1.2 and 1.4.

Before we proceed, we introduce two facts that follow immediately from proofs given in [14].

Lemma B.1. *[14] For any request $J_{p,i} \in A(t)$ it is the case that $\sum_{r_{q,j}\in A(t), r_{q,j}\geq r_{p,i}} z_{q,j}(t) \leq (t - r_{p,j})$.*

Proposition B.2. *[14]* $\sum_{J_{p,i}\in A(t)} W_{p,i}(t) \leq \sum_{J_{p,i}\in A(t)}(1 + \frac{1}{\epsilon})^{k-1} w_{p,i}(t) \leq (\frac{2}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)$.

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 $\sum_{J_{p,i}\in A(t)} W_{p,i}(t)$.

We now address the change in $\Phi(t)$ due to OPT's processing. We can without loss of generality assume that OPT processes at most one page at time t. Let $O^*(t)$ be the set of requests which OPT satisfies by working on page p. Note that $|O^*(t)| \leq B_p$ by definition of page p's batch size. Further, note that O^* contains at most B_p 'adjacent' requests due to the FIFO-nature of OPT proven in Lemma 3.1. Also note that when page p is processed by OPT, recall that the $z_{p,i}$ variable remains unaffected for all requests $J_{p,i}$ where $t < \tau_{p,i}$ and for all requests $J_{p,i}$ that have been satisfied by OPT. Now, for any request $J_{p,i} \in A(t) \cap O^*(t)$ where $\tau_{p,i} \leq t$ the variable $z_{p,i}$ will increase at a rate of $\textsc{On}(t, \infty, p, i)$. Let $S_p(t)$ be such requests. By Lemma 4.1, and due to the fact that $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(\tau_{p,j}, t, p) \leq 1$, for any request $J_{p,i} \in A(t)$ we can bound the change in $\sum_{J_{p,j}\in A(t), r_{p,j}\geq r_{p,i}} z_{p,j}(t)$ by $\sum_{J_{p,j}\in S_p(t), r_{p,j}\geq r_{p,i}} \frac{\textsc{On}(t,\infty,p,j)}{\sigma_p} \cdot \frac{\mathrm{d}}{\mathrm{dt}}\textsc{Opt}(\tau_{q,j}, t, p) \leq 1$. This allows us to upper bound the change in $\Phi(t)$ due to OPT's processing to be $\frac{1}{\epsilon}\sum_{J_{p,i}\in A(t)} W_{p,i}(t)$.

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 case. The case that $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) \leq \frac{1}{\beta\epsilon}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$ is easy, so assume that $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) > \frac{1}{\beta\epsilon}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$. In this case, the decrease in $\Phi(t)$ due to our algorithm's processing will play a crucial role to offset other increases. By a simple calculation, the total change rate of $\Phi(t)$ due to our algorithm's processing is at most,

$$\frac{1}{\epsilon}(\sum_{J_{p,i}\in A'(t)\setminus O(t)} \frac{\mathrm{d}}{\mathrm{dt}} z_{p,i}(t)) \sum_{J_{p,i}\in A(t)\setminus A'(t)} W_{p,i}(t) \tag{3}$$

Note that for any $J_{p,i} \in A'(t) \setminus O(t)$, it is the case that $\frac{\mathrm{d}}{\mathrm{dt}} z_{p,i}(t) \leq \frac{\mathrm{d}}{\mathrm{dt}}\textsc{On}(t, \infty, p, i) = -\frac{w_{p,i}(t)}{\beta w(t)}$ by definition of WLAPS. Using the assumption that $\frac{1}{\beta\epsilon}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) < \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)$, we have that

$$\sum_{J_{p,i}\in A'(t)\setminus O(t)} \frac{\mathrm{d}}{\mathrm{dt}} z_{p,i}(t) = -\sum_{J_{p,i}\in A'(t)\setminus O(t)} \frac{s w_{p,i}(t)}{\beta w(t)} \leq -\sum_{i\in A'(t)} \frac{s w_{p,i}(t)}{\beta w(t)} + \sum_{J_{p,i}\in O(r)} \frac{s w_{p,i}(t)}{\beta w(t)} \leq -s(1-\epsilon).$$

By simple algebra and Proposition B.2 we have that

$$\sum_{J_{p,i}\in A(t)\setminus A'(t)} W_{p,i}(t) = \sum_{J_{p,i}\in A(t)} W_{p,i}(t) - \sum_{J_{p,i}\in A'(t)} W_{p,i}(t) \geq \sum_{J_{p,i}\in A(t)} W_{p,i}(t) - (1+\frac{1}{\epsilon})^{k-1} \sum_{i\in A'(t)} w_{p,i}(t)$$

$$\geq \quad (1-\beta(1+\frac{1}{\epsilon})^{k-1}) \sum_{J_{p,i}\in A(t)} W_{p,i}(t) \quad [\text{Since } \sum_{J_{p,i}\in A'(t)} w_{p,i}(t) = \beta \sum_{J_{p,i}\in A(t)} \leq \beta \sum_{J_{p,i}\in A(t)} W_{p,i}(t)]$$

Thus we obtain $(3) \leq -\frac{s(1-\epsilon)}{\epsilon}(1-\beta(1+\frac{1}{\epsilon})^{k-1}) \sum_{J_{p,i}\in A} W_{p,i}(t)$.

Hence the change rate of $\Phi(t)$ due to WLAPS's processing is bounded by

$$-\frac{s(1-\epsilon)}{\epsilon}(1-\beta(1+\frac{1}{\epsilon})^{k-1}) \sum_{J_{p,i}\in A(t)} W_{p,i}(t)$$

Now consider the easier case that $\frac{\mathrm{d}}{\mathrm{dt}}\text{WLAPS}(t) \leq \frac{1}{\beta\epsilon}\frac{\mathrm{d}}{\mathrm{dt}}\text{OPT}(t)$. In this case using Proposition B.2, we can easily show that the change rate of $\Phi(t)$ due to WLAPS's processing is bounded by

$$-\frac{s(1-\epsilon)}{\epsilon}(\frac{2}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\text{OPT}(t)$$

By aggregating all the changes due to WLAPS's and OPT's processing and time elapse, we obtain (2).

C Maximum Flow Time

In this section we consider the problem of minimizing the maximum flow time. For this problem, we consider the algorithm First-In-First-Out (FIFO). In the analysis in this section, we will not use fractional flow time, but rather address the integral objective directly. The definition of the algorithm FIFO we consider is the following. Let $A(t)$ denote the set of unsatisfied requests at time t. At a fixed integral time t let $J_{p,i}$ be the oldest request in $A(t)$. Here ties are broken in an arbitrary but consistent way. Let j be as small as possible such that $J_{p,i}$ has not received the jth piece of page p, (p,j). To describe how FIFO batches requests, say FIFO processes the (p,j) piece of page p at time t. Here we will say that $J_{p,i}$ *forces* FIFO to schedule at time t. The requests in the batch satisfied in this time step are the earliest arriving requests for page p that have received all preceding pieces of page p, and are waiting for the piece (p,j). We do not relax the batch size, and hence the batch can contain at most B_p requests. Note that the request $J_{p,i}$ that forces FIFO to schedule at time t is the oldest one in the batch at time t. Notation-wise, for an interval I, we let $R(I)$ denote the set of requests that forced FIFO to schedule during I. One important property of FIFO is that it is non-preemptive, which will be useful in our analysis. That is, once any request forces FIFO to schedule then that request will be the request which forces FIFO to schedule until it is satisfied.

We now show that FIFO is 2-competitive. Consider any fixed sequence of requests and let OPT denote the optimal solution (or its maximum flow time depending on the context). As discussed in Section 3, we consider the optimal schedule OPT in the buffer model. This allows us to assume that OPT processes requests for the same page in the first-in-first-out manner. Let t' be the first time that some request $J_{q,k}$ in FIFO's schedule has response time larger than 2OPT. Let t_1 be as small as possible such that all requests in $R([t_1,t'])$ have flow time no smaller than OPT in FIFO's schedule. We begin by showing that all of the requests in $R([t_1,t'])$ cannot be satisfied simultaneously in any optimal solution.

Lemma C.1. *Any two distinct requests in $R([t_1,t'])$ cannot be satisfied simultaneously in* OPT.

Proof. Consider any two distinct requests $J_{p,i}$ and $J_{p,j}$ in $R([t_1,t'])$. We assume without loss of generality that $i<j$, which implies that $r_{p,i} \leq r_{p,j}$ and $J_{p,i}$ forces FIFO to schedule before $J_{p,j}$. Let t^* be the time FIFO is first forced to schedule because of request $J_{p,i}$. There are two cases. First say that $r_{p,j} > t^*$.

Due to the non-preemptive nature of FIFO, $J_{p,i}$ is satisfied at time $t^* + \ell_p$. However, by definition of the interval $[t_1, t']$, the request $J_{p,i}$ has flow time at least OPT. It follows that if OPT satisfies $J_{p,i}$ and $J_{p,j}$ simultaneously at any point, then OPT has to start satisfying $J_{p,i}$ later than FIFO, thereby making $J_{p,i}$'s flow time greater that OPT, which is a contradiction.

We consider the other case is when $t^* \leq r_{p,j}$. In this case, the only reason $J_{p,j}$ and $J_{p,i}$ are not satisfied together is that $J_{p,j}$ did not 'fit' into the batch that was satisfied when $J_{p,i}$ forced FIFO to schedule. That is, there must be $B_p - 1$ requests that arrive between $J_{p,i}$ and $J_{p,j}$ for page p which were satisfied simultaneously with request $J_{p,i}$. However, by assumption OPT processes requests in a first-in-first-out fashion for each page, hence $J_{p,i}$ and $J_{p,j}$ cannot be in the same batch in OPT. □

Next we show that no request in $R([t_1, t'])$ arrives too early.

Lemma C.2. *All of the requests in $R([t_1, t'])$ arrive no earlier than $t_1 - \text{OPT}$.*

Proof. For the sake of contradiction, say there exists a request $J_{p,i}$ in $R([t_1, t'])$ that arrives before time $t_1 - \text{OPT}$. Then it follows that every request in $R([r_{p,i} + \text{OPT}, t_1])$ has flow time at least OPT in FIFO's schedule, which contradicts the definition of t_1. □

Finally we are ready to prove that FIFO is 2-competitive, proving Theorem 1.5.

Proof of [Theorem 1.5] We consider the pieces of information that are processed by FIFO during requests in $R([t_1, t'])$. The total amount of this information is $(t' - t_1)$. By Lemma C.1, we know that all requests in $R([t_1, t'])$ cannot be merged in OPT. We now argue that OPT must complete all these requests during $[t_1 - \text{OPT}, r_{q,k} + \text{OPT}]$. The starting point immediately follows from Lemma C.2. The ending point follows from the fact that all the requests in $R([t_1, t'])$ arrive before time $r_{q,k}$. Since OPT has to do at least $(t' - t_1)$ amount of work during $[t_1 - \text{OPT}, r_{q,k} + \text{OPT}]$, and we derive $(t - t_1) \leq (r_{q,k} + \text{OPT}) - (t_1 - \text{OPT})$. This contradicts the fact that $t' > r_{q,k} + 2\text{OPT}$. □