© 2012 Benjamin J. Moseley

ONLINE SCHEDULING ALGORITHMS FOR BROADCASTING AND GENERAL COST FUNCTIONS

BY

BENJAMIN J. MOSELEY

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Doctoral Committee:

Associate Professor Chandra Chekuri, Chair
Associate Professor Sariel Har-Peled
Associate Professor Manoj Prabahakaran
Professor Kirk Pruhs, University of Pittsburgh

Abstract

In this thesis we study scheduling problems that occur in the client server setting. In this setting there are a set of jobs that are sent by clients over time to a sever. There is a scheduler at the sever that determines how the jobs should be processed. The goal of the scheduler is to process the jobs in a way that optimizes the quality of service given to the clients. The quality of service is determined by some metric, which is designed for the specific needs of the system. Several systems in the real world motivate the study of client server scheduling such as web servers, operating systems, and load balancing in distributed computing.

This thesis is on designing online scheduling algorithms and focuses mainly on broadcast scheduling, but also considers traditional scheduling on single and multiple machines. In the broadcast setting, clients send requests for pages of information. When the sever broadcasts a page, all clients requesting the page that was broadcasted are satisfied at the same time. The broadcast setting differs from standard models because multiple requests can be satisfied by a single broadcast.

We analyze our algorithms in a worst case analysis framework. One of the main contributions of this thesis is the design of algorithms for the online broadcast setting. We develop key algorithmic ideas and analysis techniques for broadcasting. We study the algorithm Longest-Wait-First (LWF) and its variants. We show that LWF and its extensions are $O(1)$-speed $O(1)$-competitive for average flow time and the ℓ_k norms of flow time for fixed k in the broadcast setting. We then extend and generalize the techniques developed in the analysis of LWF to show a $(1+\epsilon)$-speed $O(\text{poly}(1/\epsilon))$-competitive algorithm for minimizing average flow time in the broadcast setting.

We also consider the minimizing the maximum weighted flow time problem and show that a natural extension of LWF to this problem does not perform well. We introduce new algorithmic ideas for this problem. To better understand minimizing the maximum weighted flow time in the broadcast model, we first consider the problem in the standard single machine and multiple identical machines settings. Here we give $(1+\epsilon)$-speed $O(\text{poly}(1/\epsilon))$-competitive algorithms. Building on these algorithms we give a new algorithm for minimizing the maximum weighted flow time in the broadcast setting and show that it is $(1+\epsilon)$-speed $O(\text{poly}(1/\epsilon))$-competitive. We then consider improving the speed-up required to give an $O(1)$-competitive algorithm for the ℓ_k norms of flow time in the broadcast setting. Here we consider an extension of the Latest-Arrival-Processor-Sharing (LAPS) algorithm and show that it is $(1+\epsilon)$-speed $O(\text{poly}(1/\epsilon))$-competitive for all k.

It is typical in scheduling theory for algorithms and analysis to be tailored for specific objective functions. However, it is valuable to ask whether or not algorithms and analysis can be unified into a single algorithm or analysis framework. Motivated by this, we introduce the general-cost-function objective. In the general cost function, the objective is to minimize $\sum_{i\in[n]} w_i g(C_i - r_i)$ where w_i is the weight of the job, C_i is the completion time, r_i is the release time and g is some non-decreasing function of a job's flow time. The function g can be fixed to specific objectives and, hence, this framework captures most reasonable objective functions. For the general cost function, we give an algorithm that is $(2+\epsilon)$-speed $O(1/\epsilon)$-competitive in the standard single machine setting. The algorithm considered is oblivious to the function g. Thus, this shows that on a single machine, in the standard setting, there is indeed a single algorithm that performs well for most objectives. Further, since the algorithm is oblivious to g, it follows that the algorithm is $(2+\epsilon)$-speed $O(1/\epsilon)$-competitive for all objectives that fit into this framework *simultaneously*.

To my parents and grandparents, for their limitless love and support.

Acknowledgments

First and foremost I would like to thank my advisor Chandra Chekuri for his guidance through my graduate career. He has helped me mature academically and guide my interests. I especially appreciate his time, patience, and faith in me during the early years of my graduate studies. He is a excellent teacher and researcher and I am very grateful to have had him as my advisor. The work in this thesis would not have been possible without his support.

I also would like to thank Kirk Pruhs. He has been a second advisor to me. My visits to the University of Pittsburgh were fun and enjoyable experiences. He has been very influential on my work and my academic growth. Several chapters of this thesis was work done with Kirk or grew from our work together. I am very thankful to have had the chance to collaborate with him the past few years. I also would like to thank Ravi Kumar and Sergei Vassilvitskii. I had two fantastic internships at Yahoo! working with them. The insights and guidance I received during my internships with them has opened new research areas to me.

I owe a great deal of gratitude to Sungjin Im. It has been my pleasure to have gone through graduate school with him. He is an excellent collaborator and colleague, but also a wonderful friend. Much of the research I have done so far would not have been possible without him, including much of the work in this thesis.

I would like to thank everyone in the University of Illinois theory group. It has been a friendly and fun group to be in. Sariel Har-Peled has been very supportive over the years and working with him during my undergraduate studies is what inspired me to study theoretical computer science. I am thankful to Kyle Fox and

Alina Ene for being a good roommates during our internships in California and I enjoyed working with them on several research projects while at the University of Illinois.

I would like to thank all of my friends, especially Ian Griffin, William Arndt, and Evan Cooperman. I also would like to thank my family with a special thanks to my parents and grandparents, this thesis is dedicated to them. The encouragement and support from my family and friends has been invaluable and I am forever indebted to them.

I am very grateful to have had many collaborators and I would like to thank all of them.

Finally, I am grateful that the research in this dissertation was supported in part by a University of Illinois Fellowship and NSF grants CNS-0721899, CCF-0728782 and CCF-1016684.

Ben Moseley

Table of Contents

Notation Reference ix

Chapter 1 Introduction 1
1.1 Introduction to Scheduling Theory 2
1.2 Scheduling Models 8

Chapter 2 Problems, Results, and Related Work 12
2.1 Broadcast Scheduling 12
2.2 General Cost Function 20

Chapter 3 Minimizing the Maximum Flow Time and Delay Factor 22
3.1 Introduction 22
3.2 Standard Scheduling 24
3.3 Broadcast Scheduling 32
3.4 Minimizing the Maximum Weighted Flow Time and Weighted Delay Factor 45
3.5 Conclusions 49

Chapter 4 Longest Wait First and its Variants in the Broadcast Model . 50
4.1 Introduction 50
4.2 Minimizing Average Flow Time 54
4.3 Generalization to Delay-Factor and ℓ_k Norms 59
4.4 Lower Bound on LF 63
4.5 Conclusion 67

Chapter 5 A Scalable Algorithm for Minimizing Average Flow Time in the Broadcast Model 68
5.1 Introduction 68
5.2 Time Model and Algorithm 70
5.3 Analysis 72
5.4 Conclusion 89

Chapter 6 A Scalable Algorithm for Minimizing the ℓ_k-norms of Flow Time in the Broadcast Model 90
6.1 Introduction 90

6.2 Algorithm and Analysis 91
6.3 Conclusions 97

Chapter 7 Minimizing General Cost Functions 98
7.1 Introduction 98
7.2 Analysis of HDF 103
7.3 Lower Bounds 111
7.4 Analysis of FIFO for Unit Size Jobs 116
7.5 Analysis of WLAPS for Concave Functions 120
7.6 Conclusions and Discussions 124

Chapter 8 Conclusions and Future Directions 125

References 128

Notation Reference

OPT	The optimal solution and/or the value of the optimal solution's objective
ℓ_k	The ℓ_k-norm objective function

Standard Scheduling

J_i	The ith job
r_i	Release (arrival) time of job J_i
w_i	Weight of job J_i
d_i	Deadline of job J_i
l_i	Processing time or length of job J_i
C_i^A	Completion time of job J_i under schedule A

Broadcast Scheduling

$J_{p,i}$	The ith request for page p
$r_{p,i}$	Release (arrival) time of request $J_{p,i}$
$w_{p,i}$	Weight of request $J_{p,i}$
$d_{p,i}$	Deadline of request $J_{p,i}$
l_p	The time required to broadcast page p in the non-uniform page setting
$C_{p,i}^A$	Completion time of request $J_{p,i}$ under schedule A

Chapter 1

Introduction

This thesis considers online algorithms for scheduling (resource allocation) optimization problems. Succinctly, scheduling theory studies the allocation of a set of resources to a set of jobs. In an optimization problem, one would like to find an optimal solution for a given objective function over the set of all feasible solutions of the given problem instance. Generally, the goal is to design an efficient algorithm to find such a solution for any instance of the problem. An example optimization problem is: given a set of tasks, decide how to process them by a computer so that as many tasks as possible are completed by their deadline.

When considering an optimization problem in the *online* setting, a portion of the input is revealed over time to the algorithm. The online algorithm must make decisions at each time without the knowledge of the portion of the input that is yet to be revealed. This is in contrast to *offline* algorithms, which are assumed to know all of the properties of the entire input sequence in advance. Online algorithms have proven to be extremely useful because in many practical settings one may not know the entire problem instance in advance.

Unfortunately, for many problems it is known that no online algorithm can be optimal when compared to an offline algorithm that knows the input sequence in advance. The study of online algorithms focuses on determining the quality of the solutions that can be feasibly obtained in the online setting. For problems where there exists no online algorithm that produces an optimal solution on every input sequence, one popular form of analysis used is competitive analysis. This thesis focuses on using competitive analysis to bound the performance of online algorithms for scheduling optimization problems.

1.1 Introduction to Scheduling Theory

Scheduling theory is subject that is of interest in several areas of computer science and arises in many other fields. Employee tasks at a business, programs to be executed by a computer and communication in a network can all be seen as 'jobs' that need to be scheduled. In particular, any computer operating system includes a scheduler to determine when programs are executed. Due to the numerous applications of scheduling theory, it is an area with a significant amount of interest in both academia and industry. For more information on scheduling algorithms see [57, 60].

The motivation behind the work in this thesis is to determine 'good' scheduling algorithms in the client-sever scheduling setting. In the client sever setting, there is a set of clients and a server or job scheduler. The clients send jobs to the system over time. The sever decides how to process these jobs in the system. The client-sever scheduling setting has been extensively studied over the last few decades. The popularity of the client sever setting comes from the extensive applications of the model. The model captures jobs scheduling environments seen in web servers, operating systems, server farms, and load balancing in distributed computing settings.

In the client-server scheduling setting, n jobs or requests need to be processed. Each of these jobs requires some amount of processing from the system. The jobs arrive over time and can be processed after their arrival. In the *offline* setting the scheduler has complete knowledge of all the jobs in advance, including when they will arrive. This contrasts to the *online* setting where the scheduler learns of a job at the time it arrives. In this case, the scheduler makes its decisions with only partial knowledge of the job sequence. The goal of the scheduler is to optimize the quality of service which the jobs receive. Naturally, the definition of the quality of service depends on the needs of the specific system.

In the simplest setting, each of the jobs are to be scheduled on a single machine. In this case, the scheduler determines the jobs to be processed at each

time. There is a considerable amount of literature on this fundamental model. When scheduling on a single machine there are a variety of well-known scheduling algorithms with strong performance guarantees. For example:

- FIFO: It is known that the algorithm First-In-First-Out (FIFO) is an optimal algorithm for minimizing the maximum time any job has to wait to be satisfied. FIFO at each time schedules the job that has been unsatisfied the longest.

- SRPT: If one desires to minimize the average time that jobs have to wait to be satisfied, the algorithm Shortest-Remaining-Processing-Time (SRPT) is an optimal online algorithm. The algorithm SRPT at each time preemptively schedules the job with the least remaining processing time.

- EDF: If jobs have deadlines and all jobs can feasibly be scheduled by their deadline then the online algorithm Earliest-Deadline-First (EDF) will guarantee a schedule where all jobs get scheduled by their deadline. The algorithm EDF at each time preemptively schedules the job with the earliest deadline.

Now consider the case where jobs can be distributed on m machines or processors. In this situation, at any time, the scheduler must make the decision of which jobs to assign to which machines along with the decision of which jobs are to be processed. Scheduling on multiple machines has been studied extensively in scheduling theory. See, [58, 5, 15, 44, 4, 56, 45, 24, 49]. Designing algorithms to create good schedules in a multiple machine scheduling setting is typically much more challenging than scheduling in the single machine scheduling setting. This is because the scheduler not only needs to prioritize jobs, but it also must efficiently load balance the jobs on the machines.

The simplest multiple machines setting is where all machines are *identical*. That is, each job has the same processing time on all machines and any job can be scheduled on any machine. Several previous works have focused on the identical

machine model. For instance, [5, 58, 15, 4, 56]. See the survey [60] for other relevant work. However, in practice, machines may not be identical. Machines may have different speed processors. One model that captures this situation is the *related* machines model. Here, each machine x has some speed s_x. Job J_i requires l_i/s_x time to complete if it is assigned to machine x. Here l_i is the processing time of job J_i.

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

Algorithmic scheduling challenges may not only be related to having multiple machines. Indeed, many challenges arise in a model known as *broadcasting*. This model naturally arises in wireless and LAN networks. Here there is a single server where *pages* of information are stored. Clients send requests to the server for a page of information. In most situations, it is assumed that the sever must satisfy all of the requests. More than one client may send a request for the same page. The server can satisfy multiple requests by a single broadcast since the clients are assumed to be listening in on a multicast channel. Along with practical interest, this scheduling model has received a considerable amount of attention over the last decade in both the offline and online settings in scheduling theory [11, 2, 1, 13, 47, 6, 8, 27]. In the broadcast setting, a scheduler can satisfy multiple requests for the same page by a single broadcast. Notice that this implies that two different schedules can do a different amount of work to

satisfy the same set of requests. Not having conservation of work between two schedules makes scheduling problems considerably more challenging in worst case analysis, especially in the online setting. The difficulty is that it is hard design algorithms that balance saving work by grouping requests together and optimizing an objective.

1.1.1 Quality of Service Objectives

As mentioned, the goal of the scheduler is to optimize the quality of service the jobs receive. Generally, the client submitting a job would like the job completed as soon as possible. The *flow time* of a job is the amount of time that the scheduler takes to satisfy the job. A well known scheduling metric is minimizing the total flow time. The *total flow time* of a schedule is total amount of time that all of the jobs must wait to be satisfied in the system. By minimizing the total flow time the scheduler focuses on optimizing the average quality of service given to the jobs. Optimizing the total flow time is the most popular and well studied online scheduling metric.

Minimizing the total flow time is quite natural and useful, but this objective has the following disadvantage. A scheduler that optimizes the total flow time may focus on giving the majority of the jobs very good quality of service while giving a small number of jobs poor quality of service. This maybe unacceptable in some systems. A quality of service metric that is used to give worst case guarantees at the individual job level is *maximum flow time*. This objective focuses on reducing the maximum time any job has to wait to be satisfied. This more stringent objective ensures that no job can be ignored by the system. Hence, the objective enforces fairness between the jobs. Optimizing the fairness of a schedule is one of the highest priorities in several systems [63].

A popular objective that is used to balance the average quality of service and the fairness of the schedule that is not as rigid as the maximum flow time objective

is the ℓ_k *norms* of flow time. The ℓ_k norms of flow time objective was first studied in the influential paper of Bansal and Pruhs [9]. Let r_i denote the release time (arrival time) of job J_i and let C_i denote the time the request is completed under some schedule. The ℓ_k-norms of flow time is defined to be $\sqrt[k]{\sum_i (C_i - r_i)^k}$. Notice that the ℓ_1 norm is equivalent to the total flow time and the ℓ_∞ norm is equivalent to the maximum flow time objective. For the ℓ_k norm objective, k is usually assumed to be small, i.e. 2 or 3. In the ℓ_k norm objective, the algorithm is severely penalized when a job waits a substantial amount of time to be satisfied. This is because the flow time of the job to the kth power is considered. This forces the algorithm to minimize the variance of flow time between the jobs and, thereby, ensures the predictability of the system at the individual job level. Further, in the ℓ_k-norm, the flow time is still being considered and the algorithm must also focus on average quality of service. By optimizing the ℓ_k norm of flow time, the algorithm balances average quality of service and fairness. This makes online algorithms that perform well for the ℓ_k-norm of flow time highly desirable in practice. To see the difference between average flow time and the ℓ_k norms, consider the following well known example.

Average Flow Vs. the ℓ_k-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 flowtime, 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.*

In some systems jobs have deadlines. In this case, many system designers

focus on maximizing the number of jobs satisfied by their deadline. For instance, this maybe the case in real time systems [55]. This assumes that once a job's deadline has passed there is no longer a need to complete the job. Strict deadlines are necessary in several situations, however some systems may require that a job be completed even after the deadline has passed. This is known as a *soft deadline*. When jobs have soft deadlines, a metric that was recently suggested is called the *delay factor* [26]. Let d_i denote the deadline of job J_i. If a job is satisfied by its deadline then the delay factor of the job is 1. Otherwise the delay factor is $\frac{C_i - r_i}{d_i - r_i}$. Notice that the delay factor of a job is larger if the original amount of time the scheduler had to satisfy the job before the deadline was smaller. In the delay factor objective, it is assumed that the priority of a job is proportional to the time between the job's deadline and its arrival time.

1.1.2 Worst Case Competitive Analysis and Resource Augmentation

As mentioned, in this thesis our attention is focused on worst case analysis and the online setting. In the online setting, an algorithm is only given partial information about the set of jobs while it is making scheduling decisions. Due to this restriction, for most problems, the online algorithm cannot be optimal for every job sequence. For many online problems, instead of designing algorithms that are optimal, worst case analysis focuses on designing algorithms that are comparable with the optimal algorithm. A natural way to make this comparison is to bound the relative performance of the algorithm to that of the optimal solution for any possible job sequence. For an algorithm and a fixed objective function let A_I denote the algorithm's objective on an input sequence I. Let OPT_I denote the value of the objective for an optimal solution on the input I. We say that an algorithm is c-competitive if $A_I \leq c \cdot \mathrm{OPT}_I$ for any input I [22]. In worst case online analysis, our goal is to find an algorithm that minimizes the competitive

ratio.

Unfortunately, for several online scheduling problems, strong lower bounds exist on the competitive ratio. When there exists a super-constant lower bound on a problem, a popular form of analysis is a *resource augmentation* analysis [52, 59]. This form of analysis focuses on giving the algorithm extra resources over the optimal solution. This may come in the form of being able to process the jobs at a faster rate or by giving the algorithm more machines. The most well studied model is when the jobs are processed at a faster rate. We say that an algorithm is s-speed c-competitive if the competitive ratio of the algorithm is at most c and the algorithm can processes jobs s times faster than the optimal solution. An ideal resource augmentation result is to find an algorithm that is $(1+\epsilon)$-speed $O(f(\epsilon))$-competitive, where ϵ is any constant in $(0, 1]$ and f is some function of ϵ. That is, to find an algorithm that is constant competitive while using the minimum amount of extra resources over the adversary. We call all such algorithms *scalable*. When strong lower bounds exist on the competitive ratio, most previous work in the worst case analysis setting focuses on finding scalable algorithms.

Resource augmentation analysis has several motivations. For example, one could consider scheduling as a provisioning problem. A system designer may design a system such that it is possible for the system to give good performance. This could be determined by analyzing expected job sequences the system may incur. However, it could be the case that no online algorithm can have a small competitive ratio for the scheduling setting faced in the system. Now, if it is known that a scalable algorithm exists then by buying machines with slightly faster speeds, the designer will know that using the scalable algorithm will result in good performance.

1.2 Scheduling Models

This thesis will consider both the traditional scheduling model where requests (or jobs) are independent and the broadcast model where different requests can

be satisfied simultaneously. We will refer to the traditional scheduling model as the 'standard' scheduling model. In this setting we will refer to requests as jobs as this is more appropriate terminology. In the broadcast model, we will keep the terminology of requests, instead of jobs, to distinguish the model from the standard model. We will use this terminology throughout this thesis unless stated otherwise. In either the broadcast or standard setting, A_s will denote an algorithm A given speed s. We say that an algorithm is *non-clairvoyant* if the algorithm does not use the processing time of a job to make scheduling decision. This is in contrast to *clairvoyant* schedulers that are assume to know the processing time of a job when the job arrives. Throughout the thesis it is assumed that preemption is allowed. That is, a job can be stopped during processing and can be resumed from where it was left off later.

1.2.1 Standard Scheduling

Each job J_i has a processing time or length l_i which the scheduler must devote to the job to satisfy the job. The job is said to be released or arrive at time r_i. In the online setting, this is the first time that the scheduler is aware of the job. When the job is completed in some schedule A, we will say that it is completed at time C_i^A. In some cases, the job may have a deadline d_i and/or a weight w_i which is know to the scheduler when the job arrives.

In the traditional model, this thesis will be concerned with both the single machine and multiple machine settings. It is assumed that each job can be assigned to at most one machine at each moment in time; however, a job can possibly be migrated between machines over time. In the multiple machine setting, we will primarily consider the identical machine model where the processing time of a job is the same on all machines.

1.2.2 Broadcast Scheduling

There are n requests that arrive over time. Each request is for a page and there are at most n distinct pages or pieces of data that are available in the system. Multiple outstanding requests for the same page p are satisfied by a single transmission of the p. We use $J_{p,i}$ to denote i'th request for a page $p \in \{1, 2, \ldots, n\}$. We let $r_{p,i}$ denote the arrival time of the request $J_{p,i}$. Requests may also have deadlines, denoted $d_{p,i}$ and non-negative weights $w_{p,i}$. The finish time or completion time $C^A_{p,i}$ of a request $J_{p,i}$ in a given schedule A is defined to be the earliest time after $r_{p,i}$ when the page p is transmitted by the schedule A. Multiple requests for the same page can have the same finish time.

A significant portion of the broadcast scheduling literature focused on the case where all page sizes are the same which can be assumed to be one without loss of generality. We call this the uniform or the unit page size setting. We also consider the *non-uniform* or varying page size setting where pages have potentially different sizes. We will let l_p denote the time it takes to transfer page p. When page sizes are different, one has to carefully define when a request for a page is satisfied if it arrives midway through the transmission of that page. In this thesis we consider the sequential model [38], the most restrictive one, in which the server broadcasts each page *sequentially* and a client receives the page sequentially without buffering; see [61] on the relationship between different models. When pages have non-uniform sizes, each page p is divided into an ordered list of uniform sized pieces $(1, p), (2, p), \ldots, (\ell_p, p)$ where ℓ_p is the size of page p. In the sequential setting, a client must receive the pieces in sequential order.

1.2.3 Objective Functions

The following table in figure 1.1 shows the formal definition of most of the objectives considered in this thesis. The objectives are shown for the standard

Objective	Standard	Broadcast
Max Flow Time	$\max_i C_i - r_i$	$\max_{p,i} C_{p,i} - r_{p,i}$
Max Weighted Flow Time	$\max_i w_i(C_i - r_i)$	$\max_{p,i} w_{p,i}(C_{p,i} - r_{p,i})$
Max Delay Factor	$\max_i\{1, \frac{C_i - r_i}{d_i - r_i}\}$	$\max_{p,i}\{1, \frac{C_{p,i} - r_{p,i}}{d_{p,i} - r_{p,i}}\}$
Max Weighted Delay Factor	$\max_i\{w_i, \frac{w_i(C_i - r_i)}{d_i - r_i}\}$	$\max_{p,i}\{w_{p,i}, \frac{w_{p,i}(C_{p,i} - r_{p,i})}{d_{p,i} - r_{p,i}}\}$
Average Flow Time	$\sum_i C_i - r_i$	$\sum_{p,i} C_{p,i} - r_{p,i}$
Average Weighted Flow Time	$\sum_i w_i(C_i - r_i)$	$\sum_{p,i} w_{p,i}(C_{p,i} - r_{p,i})$
ℓ_k-norms of Flow Time	$\sqrt[k]{\sum_i (C_i - r_i)^k}$	$\sqrt[k]{\sum_{p,i} (C_{p,i} - r_{p,i})^k}$

Figure 1.1: Different objective functions discussed in this thesis. The goal is to minimize the objective.

model and broadcast model along with their weighted variants.

1.2.4 Organization

We begin in Chapter 2 by giving an overview of the results in this thesis. Here we also summarize the history of the broadcast model in the online client server setting. In Chapter 3 we consider the problem of minimizing the maximum delay factor and weighted flow time in both the standard single machine and multiple machine settings as well as the broadcast model. In Chapter 4 we study the algorithm Longest-Wait-First and its variants in the broadcast setting. In Chapter 5 a scalable algorithm for average flow time in the broadcast model is introduced. Chapter 6 focuses on designing a scalable algorithm for the ℓ_k-norms of flow time in the broadcast setting. One chapter of this thesis will focus on the standard single machine setting with the objective of minimizing the general cost function. Chapter 7 is devoted to this problem and related work can also be found in Chapter 7.

Chapter 2

Problems, Results, and Related Work

We now given an overview of the results covered in this thesis. As stated, most of this thesis will concentrate on broadcast scheduling. We consider several popular online scheduling objectives and show scalable algorithms for them. Although we consider several objective functions and different algorithms, many of the key underlying algorithmic ideas are similar. We now give an overview of the area of broadcast scheduling as well as the contributions of this thesis. Then we discuss the work on the general cost function presented in the thesis.

2.1 Broadcast Scheduling

The broadcast model is motivated by several applications such as multicast systems and wireless and LAN networks [66, 1, 2, 47]. Besides the practical interest in the model, broadcast scheduling has seen growing interest in algorithmic scheduling literature both in the offline and online settings [11, 2, 1, 13, 47]. Work has also been done in stochastic and queueing theory literature on related models [35, 34, 64, 65]. Algorithmic scheduling work was initiated by Bartal and Muthukrishanan [13]; see [53]. In addition to the applications, broadcast scheduling has sustained interest due to the significant technical challenges that basic problems in this setting have posed for algorithm design and analysis.

2.1.1 Minimizing the Maximum Flow Time and Delay Factor

In chapter 3 we consider the objectives of minimizing the maximum flow time and delay factor. Interestingly, the maximum response time metric was studied in the (short) paper of Bartal and Muthukrishnan [13] where they claimed that the online algorithm FIFO (for First In First Out) is 2-competitive for broadcast scheduling, and moreover that no deterministic online algorithm is $(2 - \epsilon)$-competitive. (As mentioned previously FIFO is optimal in the standard single machine setting). Despite the claim, no proof was published. Almost a decade later, Chang et al. [26] gave formal proofs for these claims for unit-sizes pages. The upper bound proof for FIFO in [26] is short but delicate. In fact, [13] claimed 2-competitiveness for FIFO even when pages have different sizes. As stated, in this thesis we consider the sequential model [38], the most restrictive one, in which the server broadcasts each page sequentially and a client receives the page sequentially without buffering. The claim in [13] regarding FIFO for different pages is in a less restrictive model in which clients can buffer and take advantage of partial transmissions and the server is allowed to preempt. The FIFO analysis in [26] for unit-sized pages does not appear to generalize for different page sizes. Our contribution to the maximum flow time problem is the following.

- FIFO is 2-competitive for minimizing maximum response time in broadcast scheduling even with different page sizes.

A competitive ratio of 2 is the best positive result that can be achieved; for any fixed $\epsilon > 0$ there is a lower bound of $(2 - \epsilon)$ on the competitive ratio for minimizing the maximum flow time, even if randomization is allowed [28]. Note that FIFO, whenever the server is free, picks the page p with the earliest request and *non-preemptively* broadcasts it. Our bound matches the lower bound shown even for unit-sized pages, thus closing one aspect of the problem. Our proof differs from that of Chang et al.; it does not explicitly use the unit-size assumption and this is what enables the generalization to different page sizes. The analysis is inspired by work on the maximum delay factor [32] which we discuss next.

The delay factor of a schedule is a metric recently introduced in [26] (and implicitly in [18]) when requests have deadlines. Delay factor is motivated by a variety of applications, in particular real-time systems, where requests naturally have deadlines associated with them. In real-time systems, a *hard* deadline implies that it cannot be missed, while a *soft* deadline implies some flexibility in violating it. In online settings it is difficult to respect hard deadlines. Previous work has addressed hard deadlines by either considering periodic tasks or other restrictions [23], or by focusing on maximizing throughput (the number of requests completed by their deadline) [54, 25, 67].

We are interested in online algorithms that minimize the maximum delay factor. We also consider a related metric, namely minimizing the maximum *weighted* response time. Delay factor and weighted response time have syntactic similarity if we ignore the 1 term in the definition of delay factor — one can think of the weight as the inverse of the slack. Although the metrics are somewhat similar we note that there is no direct way to reduce one to the other. On the other hand, we observe that upper bounds for one appear to translate to the other.

Surprisingly, the maximum weighted response time metric appears to not have been studied formally even in standard scheduling; however a special case, namely maximum *stretch* has received attention. The stretch of a job is its response time divided by its processing time; essentially the weight of a job is the inverse of its processing time. Bender et al. [17, 19], motivated by applications to web-server scheduling, studied maximum stretch and showed very strong lower bounds in the online setting.

To understand the complexity of the delay faction problem we first consider the problem in the standard scheduling setting. Before the work presented in this thesis, no non-trivial positive results were known about this objective in any online scheduling model. We first consider algorithms for optimizing this objective in the standard scheduling setting. Here we showed that strong lower bounds exist on the competitive ratio of any online algorithm even if all jobs are unit sized. We say that the slack of job J_i is $d_i - r_i$.

- For standard setting on a single machine no online algorithm is $\Delta^{0.4}/2$-competitive where Δ is the ratio between the maximum and minimum slacks.

Due to this strong lower bound, we focus on using a resource augmentation analysis. We give the following positive results in the single machine and multiple machine models.

- For standard setting, for any $\epsilon \in (0, 1]$, there are $(1 + \epsilon)$-speed $O(1/\epsilon)$-competitive algorithms in both single and multiple machine cases.

Next we consider this objective in the broadcast setting. The broadcast model poses several different challenges over the standard scheduling setting. The algorithms that we give for the standard scheduling setting do not extend to the broadcast setting. Exemplifying the challenge of the broadcast setting, we show the following strengthened lower bound on the competitive ratio of any online algorithm.

- For broadcast scheduling with n unit-sized pages there is no $n/4$-competitive algorithm.

We then introduce new algorithms for the broadcast model and show the following positive results.

- For broadcast setting, for any $\epsilon \in (0, 1]$, there is a $(1 + \epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm for unit-sized pages. For non-uniform sized pages and any $\epsilon \in (0, 1]$, there is a $(1 + \epsilon)$-speed $O(1/\epsilon^3)$-competitive algorithm.

We then show how these results can be extended to weighted versions of the maximum flow time and maximum delay factor problems.

We complement our upper bound result above with a lower bound. Recall that FIFO is 2-competitive for maximum response time in broadcast scheduling and is optimal for job scheduling. A natural algorithm that extends FIFO for delay

factor or weighted response time is to schedule the request in the queue that has the largest current delay factor (or weighted flow time). This natural greedy algorithm was labeled LF (longest first) since it can be seen as an extension of the well-studied LWF (longest-wait-first) for average flow time. LWF is known to be $O(1)$-competitive with $O(1)$-speed for average flow time, it was suggested in [31] that LF may be $O(1)$-speed $O(1)$-competitive for maximum delay factor. LWF and LF will be discussed further in the next section. We show that this is not the case even for standard scheduling in chapter 4.

- For any constants $s, c > 1$, LF is not c-competitive with s-speed for minimizing maximum delay factor (or weighted response time) in unicast scheduling of unit-time jobs.

2.1.2 Minimizing Average Flow Time

In chapters 4 and 5 we study the problem of minimizing average flow time. When considering average flow time, we will focus on the unit page time case. It was shown that without resource augmentation any *online* deterministic algorithm is $\Omega(n)$-competitive [53]. Further, any randomized online algorithm has a lower bound of $\Omega(\sqrt{n})$ on the competitive ratio [6].

A major difficultly of algorithmic development for average flow time is that previous work has shown that the existence of a $O(1)$-speed $O(1)$-competitive online algorithm cannot be proved using standard techniques. An algorithm A is said to be *locally* competitive if the number of requests in A's queue is comparable to the number of requests in the adversary's queue at each time. In [53] it was shown that no online algorithm can be locally competitive with an adversary. Local competitiveness has been one of the most popular methods of analysis in standard scheduling [52, 16, 53].

Average flow time in broadcast scheduling has been studied extensively over the last decade. In the *offline* setting, minimizing average flow time was first studied using non-trivial linear programming techniques coupled with resource

augmentation [53, 42, 43]. It was not until later that a complex reduction showed that this problem was in fact NP-Hard [41]. Recently, a simpler proof of this fact was found [26]. Following this line of work, a $(1+\epsilon)$-speed $O(1)$-approximation algorithm was eventually given in [6]. Here, resource augmentation was used even though it is still open if the problem admits an $O(1)$-approximation. The problem is substantially more difficult without resource augmentation. No non-trivial analysis was shown without resource augmentation until Bansal et al. gave a $O(\sqrt{n})$-approximation in [6]. More recently, a $O(\log^2 n/\log\log(n))$-approximation was shown in [7]. We note that this result relies on highly non-trivial algorithmic techniques.

In the online setting, the strong lowerbound without resource augmentation has led previous work to focus on finding $O(1)$-speed $O(1)$-competitive algorithms. The ultimate goal of this line of work is to find a scalable algorithm. Previously, there have been two main approaches used to avoid a local argument. The first was given by Edmonds and Pruhs in [38]. They showed a non-trivial reduction from the problem of minimizing average flow time in broadcast scheduling to a non-clairvoyant scheduling problem. Their reduction takes an algorithm A that is s-speed c-competitive for the non-clairvoyant scheduling problem and creates an algorithm B that is $2s$-speed c-competitive for the broadcast scheduling problem. Using this reduction, they were able to show an algorithm which is $(4+\epsilon)$-speed $O(1)$-competitive for minimizing the average flow time in broadcast scheduling [36, 38]. More recently, the same authors used this reduction to show another algorithm is $(2+\epsilon)$-speed $O(1)$-competitive [40]. Both of these algorithms can be extended to the case where pages have varying sizes.

The algorithm Longest-Wait-First (LWF) was first introduced in [53]. LWF uses a natural scheduling policy which always schedules the page with the highest flow time. Edmonds and Pruhs showed that LWF is 6-speed $O(1)$-competitive using a direct analysis that avoided the use of the reduction [39]. In this work, new novel techniques were introduced to avoid a local argument. The techniques presented in the paper were quite complex. In joint work with Chekuri, we were

able to simplify these techniques to make the key ideas more transparent. Using this, we were able to show LWF is $(3.4+\epsilon)$-speed $O(1)$-competitive [30]. However, LWF was shown to be $n^{\Omega(1)}$-competitive when given speed less than 1.618 [39].

In this thesis, we first discuss an improved analysis of LWF. We study LWF because it is a simple and easily implementable algorithm that has been shown to work well in practice and outperforms other policies in experiments [2]. We will show the following in chapter 4,

- LWF is 5-speed $O(1)$-competitive

Although there is a proof showing that LWF is $(3.4+\epsilon)$-speed $O(1)$-competitive, we present the 5 speed proof to emphasize the analysis techniques. We emphasize the techniques because they will be used for the following result shown in chapter 5.

- There is a scalable algorithm for average flow time in broadcast scheduling.

This was previously one of the main open question concerning scheduling in the broadcast model. This algorithm, like LWF, is a simple greedy algorithm. The algorithm itself keeps the main spirit of LWF, while overcoming the disadvantages of the LWF algorithm. To bound the performance of the algorithm, we build on the analysis techniques that we introduced in our analysis of LWF. Rather than using the reduction given in [38] we use a direct global charging scheme.

2.1.3 Minimizing the ℓ_k-norms of Flow Time

As mentioned in the previous section, average flow time in the broadcast setting has received a significant amount of attention recently. Minimizing the total flow time is a natural objective; unfortunately, in practice algorithms which optimize the flow time are not implemented due to a lack of fairness. Indeed, one can easily construct examples where an optimal algorithm for minimizing the total flow time may starve individual jobs of processing power for an arbitrary amount of time.

Implementing a fair algorithm is one of the highest priorities in most systems [63]. As stated previously, a popular and practical method to enforce the fairness of a schedule is to optimize the ℓ_k-norm of flow time for some fixed $k > 1$. In practice, it is usually the case that $k \in [2, 3]$. Notice that average flow time is equivalent to the ℓ_1 norm of flow time. By optimizing the ℓ_k norm of flow time, the algorithm minimizes the variance of the jobs, thus ensuring fairness while reducing the flow time. Although a schedule that optimizes the ℓ_k norm of flow time for $k > 1$ may have a larger average flowtime over a schedule that optimizes the ℓ_1 norm, the variance of flow time will be reduced. This will make the system more predictable on average for the individual jobs, which is generally more desirable [63, 62].

The algorithmic study of the ℓ_k-norms of flow time was initiated by Bansal and Pruhs. They showed that any online deterministic algorithm is $n^{\Omega(1)}$-competitive for the ℓ_k norms of flow time without resource augmentation in the standard single machine scheduling setting where $k > 1$ [9]. This contrasts with the fact that the algorithm SRPT is optimal in the ℓ_1 norm in this setting. The same authors showed that SRPT is a scalable algorithm for the ℓ_k norms of flow time on a single machine in the standard scheduling setting [9]. For the broadcast model we present the following in chapter 4,

- The algorithm LF, a variation of LWF, is $O(k)$-speed $O(k)$-competitive for the ℓ_k-norms of flow time

This was the first positive result for the ℓ_k-norms of flow time in the broadcast model. To show this result, we extend the analysis showing that LWF is 5-speed $O(1)$-competitive for average flow time, which is also presented in chapter 4.

Later, Gupta et al. , introduced the algorithm Broadcast-Weighted-Latest-Arrival-Processor-Sharing (BWLAPS) and showed that the algorithm is $(k + \epsilon)$-speed $O(k)$-competitive for any fixed $\epsilon > 0$ [46]. This result was shown by using a potential function analysis to bound the competitiveness of BWLAPS. In this thesis we will show the following in chapter 6 ,

- BWLAPS is scalable for the ℓ_k norms of flow time in the broadcast setting.

This result was originally given in joint with Edmonds and Im [37]. To show this result, we give an improved potential function analysis that better captures the performance of BWLAPS over the potential function given in [46]. Further, this is the first potential function based analysis used to show an algorithm is scalable for the objective of the ℓ_k-norms of flow time in any scheduling setting. Beyond showing that BWLAPS is scalable, this work also shows how one can use potential functions to analyze a scalable algorithm for the ℓ_k-norms of flow time. In chapter 6 the analysis of BWLAPS showing that the algorithm is scalable given in [37] is presented.

2.2 General Cost Function

A challenge faced in systems today is to determine the appropriate scheduling objective function, which can be difficult to define. For instance, it could be the case that fairness is important, but so is average performance. Fairness and average performance are two objective functions that inherently are competing. Due to this, it is not clear how to define one global objective.

To address this and related questions, we study the problem of optimizing the following general objective function on a single machine in the standard model. Let F_i denote the flow time of job J_i. The goal of the scheduler is to optimize $\sum_{i\in[n]} w_i g(F_i)$ where g is an arbitrary non-decreasing function and w_i is the weight of job J_i. Here one can think of $g(F_i)$ as being the penalty for making job J_i wait F_i time units. The function g is assumed to be non-decreasing because there should be no incentive for making a job wait longer to be satisfied. This framework captures most reasonable scheduling objectives such as total flow time and ℓ_k norms of the delay factor. Chapter 7 will be devoted to the general cost function problem.

For this objective function, we present the following result from [51],

- The algorithm Highest-Density-First (HDF) is $(2+\epsilon)$-speed $O(1/\epsilon)$-

competitive for any fixed $\epsilon > 0$

This result is fairly surprising because the algorithm HDF is oblivious to the objective function g. Further, since HDF is oblivious to g, HDF will optimize any objective that this framework captures simultaneously. We will also show a scalable algorithm for the special case of the problem where all jobs have uniform processing times and uniform weights. We also show a scalable algorithm when the function g is concave.

Further we complement these upper bounds by showing lower bounds on any randomized online algorithm for several variants of the problem. We show that HDF is essentially the best algorithm that one can consider for the problem by showing that no algorithm that is oblivious to the cost function g can be $(2-\epsilon)$-speed $O(1)$-competitive for any constant $\epsilon > 0$. Also we show that there exists a function g such that no algorithm can be scalable for this function even if it knows the function g. Further, if the problem is extended to the case where each job J_i has its own cost function g_i and the goal is to minimize $\sum_{i\in[n]} g_i(F_i)$ we show that no algorithm can be $O(1)$-speed $O(1)$-competitive.

Chapter 3

Minimizing the Maximum Flow Time and Delay Factor

3.1 Introduction

In this chapter we consider two related performance metrics, namely flow time (also referred to as response time) and a recently suggested performance metric called delay factor [26]. We also address their weighted versions. See chapter 2 for motivation and related work. In particular, we are interested in scheduling to minimize the *maximum* (weighted) flow time (over all requests[1]) or to minimize the maximum delay factor. We consider both the standard setting and the broadcast setting.

We give the first non-trivial results for online scheduling to minimize the maximum delay factor and weighted flow time in both the standard and broadcast settings. We remark that weighted flow time and delay factor, though formally not equivalent, behave similarly in terms of algorithmic development. At a heuristic level, one can interpret the term $\frac{1}{d_i - r_i}$ in the delay factor metric as the weight w_i. For this reason, we mainly discuss results for delay factor below and point out how they generalize to weighted flow time.

We first prove strong lower bounds on online competitiveness for delay factor.

- For standard setting no online algorithm is $\Delta^{0.4}/2$-competitive where Δ is the ratio between the maximum and minimum slacks.
- For broadcast scheduling with n uniform sized pages there is no $n/4$ - competitive algorithm.

[1]In this chapter we use requests instead of jobs since we address both the broadcast and standard scheduling models.

We resort to resource augmentation analysis to overcome the above lower bounds.

- For standard setting, for any $\epsilon \in (0, 1]$, there are $(1 + \epsilon)$-speed $O(1/\epsilon)$-competitive algorithms in both single and multiple machine cases. Moreover, the algorithm for the multiple machine case immediately dispatches an arriving request to a machine, and is non-migratory. An algorithm is non-migratory if it processes each request on a single machine to which it is first assigned.

- For broadcast setting, for any $\epsilon \in (0, 1]$, there is a $(1 + \epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm for uniform pages. For non-uniform sized pages, for any $\epsilon \in (0, 1]$, there is a $(1 + \epsilon)$-speed $O(1/\epsilon^3)$-competitive algorithm.

Our results for minimizing maximum delay factor can be easily extended to the problems of minimizing maximum weighted flow time and minimizing maximum weighted delay factor. We also address the problem of minimizing the maximum flow time in broadcast scheduling. We already mentioned that FIFO is 2-competitive when pages have uniform sizes [26]. In this chapter we show the following.

- FIFO is 2-competitive for minimizing the maximum flow time for non-uniform sized pages in the broadcast model.

Our results for the standard setting are related to, and borrow ideas from, previous work on minimizing ℓ_k norms of flow time and stretch [9] in the single machine and parallel machine settings [3, 29]. Our main results are for broadcast scheduling. Our algorithm and analysis are direct and explicitly demonstrate the value of making requests wait for some duration to take advantage of potential future requests for the same page. We mention that prior to our work, even in the offline setting, the only algorithm known for minimizing the maximum delay factor was a 2-speed optimal algorithm that was based on rounding a linear-programming

relaxation [26]. Our algorithm, when viewed as an offline algorithm, gives a $(1+\epsilon)$-speed $O(1/\epsilon^2)$-approximation for uniform page sizes (and $O(1/\epsilon^3)$-approximation for non-uniform page sizes) and is quite simple.

Additional Notation: We let $S_i = d_i - r_i$ denote the slack of J_i in the standard setting. We assume without loss of generality that $S_i \geq l_i$. We assume that the requests for a page are ordered by arrival time and hence $r_{p,i} \leq r_{p,j}$ for $i < j$. We use Δ to denote the ratio of maximum slack to the minimum slack in a given request sequence. For an interval $I = [a, b]$ on the real line we use $|I|$ for the length of the interval $(b - a)$. We say a request is alive at t if has arrived by t but has not yet finished.

3.2 Standard Scheduling

In this section we address the standard setting where requests are independent and consider the delay factor metric. For a request J_i, recall that r_i, d_i, l_i, C_i denote the arrival time, deadline, processing time/size, and finish time, respectively. An instance with all $l_i = 1$ (or more generally the processing times are uniform) is referred to as a uniform processing time instance. It is easy to see that preemption does not help much for uniform processing time instances when the maximum delay factor metric is considered. Assuming that the processing times are integer valued, in the single machine setting one can reduce an instance with non-uniform processing times to an instance with uniform processing times as follows: replace J_i, with processing time l_i by l_i uniform sized requests with the same arrival time and deadline as that of J_i.

We remarked earlier that scheduling to minimize the maximum stretch is a special case of scheduling to minimize the maximum delay factor. In [17] a lower bound of $P^{1/3}$ is shown for minimizing the maximum stretch where P is the ratio of the maximum processing time to the minimum processing time. They show that this lower bound holds even when P is known to the algorithm. This

implies a lower bound of $\Delta^{1/3}$ for minimizing the maximum delay factor. Here we improve the lower bound for minimizing maximum stretch to $P^{0.4}/2$ when the online algorithm is not aware of P. The lower bound example is similar to that given in [17].

Theorem 1. *Any 1-speed deterministic algorithm has competitive ratio at least $\frac{P^{0.4}}{2}$ for minimizing the maximum stretch when P is not known in advance to the algorithm.*

Proof. P will denote the ratio of the maximum to minimum processing time. For the example created, P will depend on the decisions the online algorithm makes. For the sake of contradiction, assume that some online algorithm $\mathcal{A}$ achieves a competitive ratio better than $\frac{P^{0.4}}{2}$. Now consider the following example, where L is chosen so that the parameters in the following example are integral.

> `Type 1`: At time 0 a request with processing time L arrives.
>
> `Type 2`: At times $L - L^{0.6} + L^{0.6} \cdot i$ for integral $i \in [0, L^{0.56} - L^{0.4}]$, let a request with processing time $L^{0.6}$ arrive.

Consider time $L^{1.16}$. If this is the entire sequence of requests, then the optimal schedule can be described as follows. It schedules these requests in a first-in-first-out fashion. The optimal schedule finishes the request of type 1 by time L, and hence the stretch of this request is 1. The requests of type 2 finish $2L^{0.6}$ time units after their arrival. Thus, the maximum stretch of any request in the optimal schedule is $2L^{0.6}/L^{0.6} = 2$.

The ratio of maximum to minimum processing time of the requests seen so far is $P = \frac{L}{L^{0.6}} = L^{0.4}$. Thus, the maximum stretch algorithm $\mathcal{A}$ can have is $(L^{0.4})^{0.4}/2 = L^{0.16}/2$, by assumption. Suppose $\mathcal{A}$ does not finish the request of type 1 by time $L^{1.16}$. In this case, the stretch of this request in the algorithm's schedule will be at least $\frac{L^{1.16}}{L} = L^{0.16}$. Thus, the algorithm has a competitive ratio at least $\frac{L^{0.16}}{2} = \frac{P^{0.4}}{2}$, a contradiction. Therefore, by time $L^{1.16}$ the algorithm $\mathcal{A}$

must have finished the request of type 1. Immediately after this time, requests of type 3 arrive.

> `Type 3`: Starting at time $L^{1.16}$ a set of $L^{1.2} - L^{0.6}$ unit processing time requests arrive as follows. These requests arrive one after another; one such request arrives at each integer time step during $[L^{1.16}, L^{1.16} + L^{1.2} - L^{0.6}]$.

This is the entire request sequence. We now analyze the behaviour of an optimum schedule for this sequence and compare it to that of $\mathcal{A}$. An optimal schedule for this sequence schedules the request of type 1 from time 0 until time $L - L^{0.6}$. At time $L - L^{0.6}$, the type 1 request has $L^{0.6}$ remaining processing time left; the optimal solution schedules requests of type 2 and type 3 requests as they arrive. It is easy to verify that the stretch of requests of type 2 and 3 is 1 in this schedule. At time $L^{1.16} + L^{1.2} - L^{0.6}$ the optimum schedule finishes the request of type 1. Thus the maximum stretch of this schedule is for the type 1 request and is equal to $\frac{L^{1.16}+L^{1.2}}{L} \leq 2L^{0.2}$.

As we argued earlier, $\mathcal{A}$ must have completely scheduled the request of type 1 by time $L^{1.16}$. Thus the last request it finishes is either of type 2 or type 3. $\mathcal{A}$ has a volume of at least $L^{0.6}$ of type 2 requests left to complete at time $L^{1.16}$. If the last request completed by $\mathcal{A}$ is of type 2 then this request must have waited for all requests of type 3 to finish. Since the arrival time of a type 2 request is at most $L^{1.16} - L^{0.6}$ and the request is completed by time $L^{1.16} + L^{1.2} - L^{0.6}$ at the earliest (this time is the latest arrival of a type 3 request), the total time this request waits to be satisfied is $L^{1.16} + L^{1.2} - L^{0.6} - (L^{1.16} - L^{0.6}) = L^{1.2}$. Thus the stretch of this request is at least $\frac{L^{1.2}}{L^{0.6}} = L^{0.6}$. If the last request satisfied by the algorithm is of type 3, then this request must have waited for $L^{0.6}$ time since a $L^{0.6}$ volume of processing time of type 2 requests remained in the algorithm's queue when type 3 requests began arriving. The stretch of this request is therefore at least $L^{0.6}$. Notice that the ratio of maximum to minimum processing time is $P = L$. In

either case, the competitive ratio of the algorithm is at least $\frac{L^{0.6}}{2L^{0.2}} = \frac{L^{0.4}}{2} = \frac{P^{0.4}}{2}$, a contradiction. □

Recall that Δ is the ratio of the maximum slack to the minimum slack in a given sequence of requests. From the above we have the following corollary for delay factor.

Corollary 2. *Any deterministic online algorithm has competitive ratio at least* $\frac{\Delta^{0.4}}{2}$ *for minimizing the maximum delay factor when requests have uniform sizes and* Δ *is not known in advance to the algorithm.*

In the next two subsections we show that with $(1+\epsilon)$ resource augmentation simple algorithms achieve an $O(1/\epsilon)$ competitive ratio.

3.2.1 Single Machine Scheduling

In this section we consider a simple greedy algorithm for minimizing the maximum delay factor on a single machine when requests have non-uniform sizes. We analyze the simple shortest-slack-first (SSF) algorithm which at any time t schedules the request with the shortest slack. Recall that the slack of a request J_i is $d_i - r_i$ and that we have assumed without loss of generality that all requests have uniform sizes.

Algorithm: SSF

- Let $Q(t)$ be the set of alive requests at t.
- Let J_i be the request with the minimum slack among requests in $Q(t)$, ties broken by arrival time.
- Preempt the current request and schedule J_i if it is not being processed.

Theorem 3. *The algorithm* SSF *is an* $(1+\epsilon)$*-speed* $\frac{1}{\epsilon}$*-competitive online algorithm for minimizing the maximum delay factor in the standard setting.*

Proof. Consider an arbitrary request sequence σ and let α^{SSF} be the maximum delay factor achieved by SSF on σ. If $\alpha^{\text{SSF}} = 1$ there is nothing to prove, so assume that $\alpha^{\text{SSF}} > 1$. Let J_i be the request that witnesses α^{SSF}, that is $\alpha^{\text{SSF}} = (C_i - r_i)/S_i$. Note that SSF does not process any request with slack more than S_i in the interval $[r_i, C_i]$. Let t be the minimum value less than or equal to r_i such that SSF was busy processing only requests with slack at most S_i in the interval $[t, C_i]$. It follows that SSF had no requests with slack $\leq S_i$ just before t. The total work that SSF processed in $[t, C_i]$ on requests with slack less than equal to S_i is $(1+\epsilon)(C_i - t)$ and all these requests arrive in the interval $[t, C_i]$. An optimal offline algorithm with 1-speed can do total work of at most $(C_i - t)$ in the interval $[t, C_i]$ and hence the earliest time by which it can finish these requests is $C_i + \epsilon(C_i - t) \geq C_i + \epsilon(C_i - r_i)$. Since all these requests have slack at most S_i and have arrived before C_i, it follows that $\alpha^* \geq \epsilon(C_i - r_i)/S_i$ where α^* is the maximum delay factor of the optimal offline algorithm with 1-speed machine. Therefore, we have that $\alpha^{\text{SSF}}/\alpha^* \leq 1/\epsilon$. □

Remark 1. *For uniform processing time requests, the algorithm that* non-preemptively *schedules requests with the shortest slack is a* $(1+\epsilon)$*-speed* $\frac{2}{\epsilon}$*-competitive online algorithm for minimizing the maximum delay factor.*

3.2.2 Multiple Machine Scheduling

We now consider minimizing the maximum delay factor when there are m machines. In the multiple machine setting, at any time a machine can choose to process at most one request and a request can be processed by at most one machine at a given time. A request is allowed to migrate and be processed by different machines at different times; as we remarked earlier, a schedule or algorithm is non-migratory if it does not migrate any request. To adapt SSF to this

setting we take intuition from previous work on minimizing ℓ_k norms of flow time and stretch [9, 3, 29]. We develop an algorithm that immediately dispatches an arriving request to a machine, and further does not migrate an assigned request to a different machine once it is assigned. Each machine essentially runs the single machine SSF algorithm and thus the only remaining ingredient to describe is the dispatching rule. For this purpose the algorithm groups requests into classes based on their slack. A request J_i is said to be in class k if $S_i \in [2^k, 2^{k+1})$. The algorithm maintains the total processing time of requests (referred to as *volume*) that have been assigned to machine x in each class k. Let $U^x_{=k}(t)$ denote the total processing time of requests assigned to machine x by time t of class k. With this notation, the algorithm SSF-ID (for SSF with immediate dispatch) can be described.

Algorithm: SSF-ID

- When a new request J_i of class k arrives at time t, assign it to a machine x where $U^x_{=k}(t) = \min_y U^y_{=k}(t)$.
- Use SSF on each machine separately.

The rest of this section is devoted to the proof of the following theorem.

Theorem 4. SSF-ID *is a* $(1+\epsilon)$*-speed* $O(\frac{1}{\epsilon})$*-competitive online algorithm for minimizing the maximum delay factor on* m *identical machines.*

We need a fair amount of notation. For each time t, machine x, and class k we define several quantities. For example $U^x_{=k}(t)$ is the total volume assigned to machine x in class k by time t. We use the predicate "$\leq k$" to indicate classes 1 to k. Thus $U^x_{\leq k}(t)$ is the total volume assigned to machine x in classes 1 to k. We let $R^x_{=k}(t)$ to denote the remaining processing time on machine x at time t and let $P^x_{=k}(t)$ denote the total volume that x has finished on requests in class k by time t. Note that $P^x_{=k}(t) = U^x_{=k}(t) - R^x_{=k}(t)$. All these quantities refer to the algorithm

SSF-ID. We use $V^*_{=k}(t)$ and $V_{=k}(t)$ to denote the remaining volume of requests in class k in an optimal offline algorithm with speed 1 and SSF-ID with speed $(1+\epsilon)$, respectively. Observe that $V_{=k}(t) = \sum_x R^x_{=k}(t)$. The quantities $V^*_{\leq k}(t)$ and $V_{\leq k}(t)$ are defined analogously.

The algorithm SSF-ID balances the amount of processing time for requests with similar slack. Note that the assignment of requests is not based on the current volume of unfinished requests on the machines, rather the assignment is based on the volume of requests that were assigned in the past to different machines. We begin our proof by showing that for any k the total volume of requests of class at most k that is assigned is balanced across the machines. This easily follows from the dispatching policy of the algorithm. Several of the lemmas below and their proofs follow from the work in [3, 29].

Proposition 5. *For any time t and two machines x and y, $|U^x_{=k}(t) - U^y_{=k}(t)| \leq 2^{k+1}$. This also implies that $|U^x_{\leq k}(t) - U^y_{\leq k}(t)| \leq 2^{k+2}$.*

Proof. The first inequality holds since all of the requests of class k are of size $\leq 2^{k+1}$. The second inequality follows easily from the first. □

Lemma 6. *Consider any two machines x and y. At any time t, the difference in volume of requests that already have been processed is bounded as $|P^x_{\leq k}(t) - P^y_{\leq k}(t)| \leq 2^{k+2}$.*

Proof. Suppose the lemma is false. Then there is the first time t_0 when $P^x_{\leq k}(t_0) - P^y_{\leq k}(t_0) = 2^{k+2}$ and small constant $\delta t > 0$ such that $P^x_{\leq k}(t_0 + \delta t) - P^y_{\leq k}(t_0 + \delta t) > 2^{k+2}$. Let $t' = t_0 + \delta t$. For this to occur, x processes a request of class $\leq k$ during the interval $I = [t_0, t']$ while y processes a request of class $> k$. Since each machine uses SSF, it must be that y had no requests in classes $\leq k$ during I which implies that $U^y_{\leq k}(t') = P^y_{\leq k}(t')$. Therefore,

$$U^y_{\leq k}(t') = P^y_{\leq k}(t') < P^x_{\leq k}(t') - 2^{k+2} \leq U^x_{\leq k}(t') - 2^{k+2},$$

since $P^x_{\leq k}(t') \leq U^x_{\leq k}(t')$. However, this implies that

$$U^y_{\leq k}(t') < U^x_{\leq k}(t') - 2^{k+2},$$

a contradiction to Proposition 5. □

Lemma 7. *At any time t the difference between the residual volume of requests that needs to be processed, on any two different machines x and y, is bounded as $|R^x_{\leq k}(t) - R^y_{\leq k}(t)| \leq 2^{k+3}$.*

Proof. Combining Proposition 5, Lemma 6, and the fact that $R^x_{\leq k}(t) = U^x_{\leq k}(t) - P^x_{\leq k}(t)$ for any x and k by definition then,

$$|R^x_{\leq k}(t) - R^y_{\leq k}(t)| \leq |U^x_{\leq k}(t) - U^y_{\leq k}(t)| + |P^x_{\leq k}(t) - P^y_{\leq k}(t)| \leq 2^{k+3}.$$

□

Corollary 8. *At any time t, $V^*_{\leq k}(t) \geq V_{\leq k}(t) - m2^{k+3}$.*

We are now ready to upper bound the competitiveness of SSF-ID when given $(1+\epsilon)$-speed in a similar fashion to the single machine case. Consider an arbitrary request sequence σ and let J_i be the request that witnesses the maximum delay factor $\alpha^{\text{SSF-ID}}$ of SSF-ID. Let k be the class of J_i and let x be the machine J_i was processed on by SSF-ID. We know that $\alpha^{\text{SSF-ID}} = (C_i - r_i)/S_i$ by definition of J_i. We use α^* to denote the delay factor of some fixed optimal algorithm that uses m machines of speed 1.

Let t be the last time before r_i when machine x processed a request of class $> k$ under SSF-ID's schedule. Note that $t \leq r_i$ since x does not process any request of class $> k$ in the interval $[r_i, C_i]$. At time t we know by 8 that $V^*_{\leq k}(t) \geq V_{\leq k}(t) - m2^{k+3}$. If $C_i \leq r_i + 2^{k+4}$ then SSF-ID achieves a competitive ratio of 16 since J_i is in class k. Thus we will assume from now on that $C_i > r_i + 2^{k+4}$.

During the interval $I = [t, C_i)$, SSF-ID completes a total volume of $(1+\epsilon)(C_i - t)$ work on machine x. Using 6, any other machine y also processes a volume of

$(1+\epsilon)(C_i - t) - 2^{k+3}$ work during I of requests in classes at most k . Thus the total volume processed by SSF-ID during I of requests in classes at most k is at least $m(1+\epsilon)(C_i - t) - m2^{k+3}$. During I, the optimal algorithm schedules at most a $m(C_i - t)$ volume of work for requests in classes at most k. Combining this with 8, we see that

$$\begin{aligned} V^*_{\leq k}(C_i) &\geq V_{\leq k}(t) - m2^{k+3} + m(1+\epsilon)(C_i - t) - m2^{k+3} \\ &\geq V_{\leq k}(t) + m(1+\epsilon)(C_i - t) - m2^{k+4} \geq \epsilon m(C_i - t). \end{aligned}$$

In the last inequality we use the fact that $C_i - t \geq C_i - r_i \geq 2^{k+4}$. Without loss of generality assume that no requests arrives exactly at C_i. Therefore $V^*_{\leq k}(C_i)$ is the total volume of requests in classes 1 to k that the optimal algorithm has left to finish at time C_i and all these requests have arrived before C_i. The earliest time that the optimal algorithm can finish all these requests is $C_i + \epsilon(C_i - t)$ and therefore it follows that $\alpha^* \geq \epsilon(C_i - t)/2^{k+1}$. Since $\alpha^{\text{SSF-ID}} \leq (C_i - r_i)/2^k$ and $t \leq r_i$, it follows that $\alpha^{\text{SSF-ID}} \leq 2\alpha^*/\epsilon$.

Thus $\alpha^{\text{SSF-ID}} \leq \max\{16, 2\alpha^*/\epsilon\}$ which completes the proof of Theorem 4.

3.3 Broadcast Scheduling

We now move our attention to the broadcast model where multiple requests can be satisfied by the transmission of a single page. Most of the literature in broadcast scheduling is concerned with the case where all pages have uniform sizes which is assumed to be unit. Here we consider both the case where pages have uniform and non-uniform sizes. We start by focusing on minimizing the maximum flow time of a schedule and then shift our focus to minimizing the maximum delay factor.

3.3.1 Minimizing the Maximum Flow Time

In this section we analyze FIFO for minimizing maximum response time when page sizes are non-uniform. As mentioned previously, it is known that FIFO is 2-competitive when pages have uniform sizes [26]. We first describe the algorithm FIFO. FIFO broadcasts pages *non-preemptively*; the optimal solution is allowed to use preemption. Consider a time t when FIFO finished broadcasting a page or a request arrives when FIFO has no unsatisfied requests just before time t. Let $J_{p,i}$ be the request in FIFO's queue with earliest arrival time breaking ties arbitrarily. FIFO begins broadcasting page p at time t. At any time during this broadcast, we will say that $J_{p,i}$ *forced* FIFO to broadcast page p at this time. When broadcasting a page p all requests for page p that arrived before or at the start of the broadcast are simultaneously satisfied when the broadcast completes. Any request for page p that arrives during the broadcast are not satisfied until the next full transmission of p. Recall that we are assuming the sequential model where the client does not buffer. The rest of this section will be devoted to proving the following theorem.

Theorem 9. *FIFO is a 2-competitive online algorithm for minimizing the maximum flow time in broadcast scheduling when pages have non-uniform sizes.*

We do not assume speed augmentation when analyzing FIFO. Let σ be an arbitrary sequence of requests. Let OPT denote some fixed optimum schedule and let ρ^* denote the optimum maximum flow time and ρ^{FIFO} denote FIFO's maximum flow time. We will show that $\rho^{\text{FIFO}} \leq 2\rho^*$. For the sake of contradiction, assume that FIFO witnesses a flow time $c\rho^*$ by some request $J_{q,k}$ for some $c > 2$. Let t^* be the time $J_{q,k}$ is satisfied, that is $t^* = C_{q,k}$. Let t_1 be the smallest time less than t^* such that at any time t during the interval $[t_1, t^*]$ the request which forces FIFO to broadcast a page at time t has flow time at least ρ^* when satisfied. Note that $t_1 \leq t^* - l_q$ where l_q is the length of page q. We let I denote the interval $[t_1, t^*]$. Let $\mathcal{J}_I$ denote the requests which forced FIFO to broadcast during I. Notice that during the interval I, all requests in $\mathcal{J}_I$ are completely satisfied during

this interval. In other words, any request in $\mathcal{J}_I$ starts being satisfied during I and is finished during I.

We say that OPT *merges* two distinct requests for a page p if they are satisfied by the same broadcast.

Lemma 10. OPT *cannot merge any two requests in* $\mathcal{J}_I$ *into a single broadcast.*

Proof. Let $J_{p,i}, J_{p,j} \in \mathcal{J}_I$ such that $i < j$. Note that $J_{p,i}$ is satisfied before $J_{p,j}$. Let t' be the time that FIFO *starts* satisfying request $J_{p,i}$. By the definition of I, request $J_{p,i}$ has flow time at least ρ^*. The request $J_{p,j}$ must arrive after time t', that is $r_{p,j} > t'$, otherwise request $J_{p,j}$ is satisfied by the same broadcast of page p that satisfied $J_{p,i}$. Therefore, it follows that if OPT merges $J_{p,i}$ and $J_{p,j}$ then the finish time of $J_{p,i}$ in OPT is strictly greater than its finish time in FIFO which is already at least ρ^*; this is a contradiction to the definition of ρ^*. □

Lemma 11. *All requests in* $\mathcal{J}_I$ *arrived no earlier than time* $t_1 - \rho^*$.

Proof. For the sake of contradiction, suppose some request $J_{p,i} \in \mathcal{J}_I$ arrived at time $r_{p,i} < t_1 - \rho^*$. During the interval $[r_{p,i} + \rho^*, t_1]$ the request $J_{p,i}$ must have wait time at least ρ^*. However, then any request which forces FIFO to broadcast during $[r_{p,i}+\rho^*, t_1]$ must have response time at least ρ^*, contradicting the definition of t_1. □

We are now ready to prove Theorem 9, stating that FIFO is 2-competitive.

Proof. Recall that all requests in $\mathcal{J}_I$ are completely satisfied during I. Thus we have that the total size of requests in $\mathcal{J}_I$ is $|I|$. By definition $J_{q,k}$ witnesses a flow time greater than $2\rho^*$ and therefore $t^* - r_{q,k} > 2\rho^*$. Since $J_{q,k} \in \mathcal{J}_I$ is the last request done by FIFO during I, all requests in $\mathcal{J}_I$ must arrive no later than $r_{q,k}$. Therefore, these requests must be finished by time $r_{q,k} + \rho^*$ by the optimal solution. From Lemma 11, all the requests $\mathcal{J}_I$ arrived no earlier than $t_1 - \rho^*$. Thus OPT must finish all requests in $\mathcal{J}_I$, whose total volume is $|I|$, during $I_{opt} = [t_1 - \rho^*, r_{q,k} + \rho^*]$. Thus it follows that $|I| \leq |[t_1 - \rho^*, r_{q,k} + \rho^*]|$,

which simplifies to $t^* \leq r_{q,k} + 2\rho^*$. This is a contradiction to the assumption that $t^* - r_{q,k} > 2\rho^*$. □

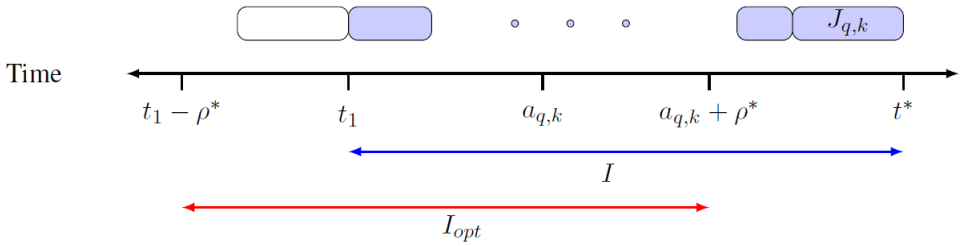


Figure 3.1: Broadcasts by FIFO satisfying requests in $\mathcal{J}_I$ are shown in blue. Note that $r_{q,k}$ and $r_{q,k} + \rho^*$ are not necessarily contained in I.

We now discuss the differences between our proof of FIFO for non-uniform sized pages and the proof given by Chang et al. in [26] showing that FIFO is 2-competitive for uniform sized pages. In [26] it is shown that at anytime t, $F(t)$, the set of *unique* pages in FIFO's queue satisfies the following property: $|F(t) \setminus O(t)| \leq |O(t)|$ where $O(t)$ is the set of unique pages in OPT's queue. This easily implies the desired bound. To establish this, they use a slot model in which uniform sized pages arrive only during integer times which allows one to define unique pages. This may appear to be a technicality, however when considering non-uniform sized pages, it is not clear how one defines unique pages since this number varies during the transmission of p as requests accumulate. Our approach avoids this issue in a clean manner by not assuming a slot model or uniform sized pages.

3.3.2 Minimizing the Maximum Delay Factor

In this section we consider minimizing the maximum delay factor in the broadcast setting. We start by showing that no 1-speed online algorithm can be $(n/4)$-competitive for minimizing the maximum delay factor where n is the number of pages. We then prove the following theorem

Theorem 12. *There is an online algorithm that is $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive for minimizing the maximum weighted delay factor in the broadcast*

setting where pages have uniform size. For non-uniform sized pages there is a $(1+\epsilon)$*-speed* $O(1/\epsilon^3)$*-competitive online algorithm.*

In Section 3.3.2 we show a $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive online algorithm for uniform sized pages and uniform weight requests. Finally, we extend our algorithm and analysis to the case of non-uniform page sizes and uniform weight requests to obtain a $(1+\epsilon)$-speed $O(1/\epsilon^3)$-competitive online algorithm in Section 3.3.2. In Section 3.4 we show how to extend these results to the case where requests have non-uniform weights.

Theorem 13. *Every* 1*-speed online deterministic algorithm for broadcast scheduling to minimize the maximum delay factor has a competitive ratio of at least* $n/4$ *where* n *is the number of pages even when pages have uniform sizes.*

Proof. The lower bound instance we construct is inspired by a lower bound given in [26]. Let $\mathcal{A}$ be any online 1-speed algorithm and let n be a multiple of 4. We consider the following adversary. At time 0, the adversary requests pages $1, \ldots, \frac{n}{2}$, all which have a deadline of $\frac{n}{2}$. Between time 1 and $\frac{n}{4}$ the adversary requests whatever page the online algorithm $\mathcal{A}$ broadcasts immediately after that request is broadcast; this new request also has a deadline of $\frac{n}{2}$. It follows that at time $t = \frac{n}{2}$ the online algorithm $\mathcal{A}$ has $\frac{n}{4}$ requests for distinct pages in its queue. However, the adversary can finish all these requests by time $\frac{n}{2}$. Then starting at time $\frac{n}{2}$ the adversary requests $\frac{n}{2}$ new pages, say $\frac{n}{2}+1, \ldots, n$. These new pages are requested, one at each time step, in a cyclic fashion for n^2 cycles. More formally, for $i = 1, \ldots, n/2$, page $\frac{n}{2}+i$ is requested at times $j \cdot (\frac{n}{2}) + i - 1$ for $j = 1, \ldots, n$. Each of these requests has a slack of one which means that their deadline is one unit after their arrival. The adversary can satisfy these requests with delay since it has no queue at any time; thus its maximum delay factor is 1. However, the online algorithm $\mathcal{A}$ has $\frac{n}{4}$ requests in its queue at time $\frac{n}{2}$; each of these has a slack of $\frac{n}{2}$. We now argue that the delay factor of $\mathcal{A}$ is at least $n/4$. If the algorithm satisfies two slack 1 requests for the same page by a single transmission, then its delay factor is $n/2$; this follows since the requests for the same page are $n/2$ time

units apart. Otherwise, the algorithm does not merge any requests for the same page and hence finishes the the last request by time $n/2 + n^2/2 + n/4$. If the last request to be finished is a slack 1 request, then its delay factor is at least $n/4$ since the last slack 1 requests is released at time $n/2 + n^2/2$. If the last request to be finished is one of the requests with slack $n/2$, then its delay factor is at least $(n^2/2)/(n/2) = n$. □

Uniform Sized Pages

In this section we develop an online algorithm, for uniform sized pages, that is competitive given extra speed. Simple examples show that natural generalizations of the algorithm SSF to the broadcast setting fail to be constant competitive with any constant resource augmentation. The reason for this is that an algorithm that focuses on requests with the smallest slack can be made to do an arbitrary amount of "extra" work over the optimal schedule by repeatedly requesting the same page and giving the page small slack. The algorithm will repeatedly broadcast the same page while the adversary waits and satisfies multiple requests for this page with a single transmission. Further, we will show in Section 4.4 that another greedy algorithm modeled after the well-studied algorithm Longest-Wait-First is not constant competitive with any fixed constant resource augmentation.

We begin by developing a variant of SSF that *adaptively* introduces a waiting time for requests. The algorithm uses a single real-valued parameter $c < 1$ to control the waiting period. The algorithm SSF-W (SSF with waiting) is formally defined below. We note that the algorithm is non-preemptive in that a request once scheduled is not preempted. As we mentioned earlier, for uniform sized requests, preemption is not very helpful. At each time, the algorithm computes the delay factor of all requests in the algorithm's queue, and considers requests for scheduling only when their delay factor is comparable to the request with maximum delay factor among the unsatisfied requests. Since our algorithm SSF schedules only the requests that have waited sufficiently long, the adversary cannot

delay those requests to satisfy them with a less number of transmissions. We formalize this intuition in Lemma 14.

Algorithm: SSF-W

- The algorithm is non-preemptive. Let t be a time that the machine is free (either because a request has just finished or there are no requests to process).
- Let $Q(t)$ be the set of alive requests at time t and let $\alpha_t = \max_{J_{p,i} \in Q(t)} \max\{1, \frac{t-r_{p,i}}{S_{p,i}}\}$ be the maximum current delay factor of requests in $Q(t)$.
- Let $Q'(t) = \{J_{p,i} \in Q(t) \mid \frac{t-r_{p,i}}{S_{p,i}} \geq \frac{1}{c}\alpha_t\}$ be the set of requests in $Q(t)$ with current delay factor at least $\frac{1}{c}\alpha_t$.
- Let $J_{p,i}$ be the request in $Q'(t)$ with the smallest slack. Broadcast page p non-preemptively.

We analyze SSF-W when it is given a $(1+\epsilon)$-speed machine. Let $c > 1 + \frac{2}{\epsilon}$ be the constant which parameterizes SSF-W. Let σ be an arbitrary sequence of requests. We let OPT denote some fixed offline optimum schedule and let α^* and $\alpha^{\text{SSF-W}}$ denote the maximum delay factor achieved by OPT and SSF-W, respectively. We will show that $\alpha^{\text{SSF-W}} \leq c^2\alpha^*$. For the sake of contradiction, suppose that SSF-W witnesses a delay factor greater than $c^2\alpha^*$. We consider the *first* time t^* when SSF-W has some request in its queue with delay factor $c^2\alpha^*$. Let the request $J_{q,k}$ be a request which achieves the delay factor $c^2\alpha^*$ at time t^*. Let t_1 be the smallest time less than t^* such that at each time t during the interval $(t_1, t^*]$ if SSF-W is forced to broadcast by request $J_{p,i}$ at time t it is the case that $\frac{t-r_{p,i}}{S_{p,i}} > \alpha^*$ and $S_{p,i} \leq S_{q,k}$. We let $I = [t_1, t^*]$[2].

[2]The algorithm SSF-W was proposed in [32] and was shown to be $(2+\epsilon)$-speed $O(1/\epsilon^2)$-competitive. The improved analysis of the same algorithm that we present here first appeared in [31]. The weaker analysis in [32] was based on defining t_1 (implicitly) to be $r_{q,k} + c(C_{q,k} - r_{q,k})$.

We let $\mathcal{J}_I$ denote the requests which forced SSF-W to broadcast during the interval $[t_1, t^*]$. We now show that any two requests in $\mathcal{J}_I$ cannot be satisfied with a single broadcast by the optimal solution. Intuitively, the most effective way the adversary performs better than SSF-W is to merge requests of the same page into a single broadcast. Here we will show this is not possible for the requests in $\mathcal{J}_I$.

Lemma 14. OPT *cannot merge any two requests in* $\mathcal{J}_I$ *into a single broadcast.*

Proof. Let $J_{x,i}, J_{x,j} \in \mathcal{J}_I$ such that $i < j$. Let t' be the time that SSF-W starts satisfying request $J_{x,i}$. By the definition of I, request $J_{x,i}$ must have delay factor greater than α^* at time $C_{x,i}$. We also know that the request $J_{x,j}$ must arrive after time t', otherwise request $J_{x,j}$ must also be satisfied at time t'. If the optimal solution combines these requests into a single broadcast then the request $J_{x,i}$ must wait until the request $J_{x,j}$ arrives to be satisfied. However, this means that the request $J_{x,i}$ must achieve a delay factor greater than α^* by OPT, a contradiction to the definition of α^*. □

To fully exploit the advantage of speed augmentation, we need to ensure that the length of the interval I is sufficiently long.

Lemma 15. $|I| = |[t_1, t^*]| \geq (c^2 - c)S_{q,k}\alpha^*$.

Proof. The request $J_{q,k}$ has delay factor greater than $c\alpha^*$ at any time during $I' = [t', t^*]$, where $t' = t^* - (c^2 - c)S_{q,k}\alpha^*$. Let $\tau \in I'$. The largest delay factor any request can have at time τ is less than $c^2\alpha^*$ by definition of t^* being the first time SSF-W witnesses delay factor $c^2\alpha^*$. Hence, $\alpha_\tau \leq c^2\alpha^*$. Thus, the request $J_{q,k}$ is in the queue $Q(\tau)$ because $c\alpha^* \geq \frac{1}{c}\alpha_\tau$. Moreover, this means that any request that forced SSF-W to broadcast during I', must have delay factor greater than α^* and since $J_{q,k} \in Q(\tau)$ for any $\tau \in I'$, the requests scheduled during I' must have slack at most $S_{q,k}$. □

We now explain a high level view of how a contradiction is found. From Lemma 14, we know any two requests in $\mathcal{J}_I$ cannot be merged by OPT. Thus if

we show that OPT must finish all these requests during an interval which is not long enough to include all of them, we will have a contradiction. More precisely, we will show that all requests in $\mathcal{J}_I$ must be finished during I_{opt} by OPT, where $I_{opt} = [t_1 - 2S_{q,k}\alpha^* c, t^*]$. It is easy to see that all these requests already have delay factor α^* by time t^*, thus the optimal solution must finish them by time t^*. We first lower bound the arrival times of the requests in $\mathcal{J}_I$.

Lemma 16. *Any request in $\mathcal{J}_I$ must have arrived no earlier than $t_1 - 2S_{q,k}\alpha^* c$.*

Proof. For the sake of contradiction, suppose that some request $J_{p,i} \in \mathcal{J}_I$ arrived at time $t' < t_1 - 2S_{q,k}\alpha^* c$. Recall that $J_{p,i}$ has a slack no bigger than $S_{q,k}$ by the definition of I. Therefore at time $t_1 - S_{q,k}\alpha^* c$, $J_{p,i}$ has a delay factor greater than $c\alpha^*$. Thus any request scheduled during the interval $I' = [t_1 - S_{q,k}\alpha^* c, t_1]$ has a delay factor greater than α^*. We observe that $J_{p,i}$ is in $Q(\tau)$ for $\tau \in I'$; otherwise there must be a request with a delay factor bigger than $c^2\alpha^*$ at time τ and this is a contradiction to the assumption that t^* is the first time that SSF-W witnessed a delay factor of $c^2\alpha^*$. Therefore any request scheduled during I' has a slack no bigger than $S_{p,i}$. Also we know that $S_{p,i} \leq S_{q,k}$. In sum, we showed that any request done during I' had slack no bigger than $S_{q,k}$ and a delay factor greater than α^*, which is a contradiction to the definition of t_1. □

Now we are ready to prove the competitiveness of SSF-W.

Lemma 17. *Suppose c is a constant s.t. $c > 1 + 2/\epsilon$. If* SSF-W *has $(1+\epsilon)$-speed then $\alpha^{\text{SSF-W}} \leq c^2\alpha^*$.*

Proof. For the sake of contradiction, suppose that $\alpha^{\text{SSF-W}} > c^2\alpha^*$. During the interval I, the number of broadcasts which SSF-W transmits is $(1+\epsilon)|I|$. From Lemma 16, all the requests processed during I have arrived no earlier than $t_1 - 2c\alpha^* S_{q,k}$. We know that the optimal solution must process these requests before time t^* because these requests have delay factor greater than α^* by t^*. By Lemma 14 the optimal solution must make a unique broadcast for each of these requests. Thus, the optimal solution must finish all of these requests in $2c\alpha^* S_{q,k} + |I|$

time steps. Thus, it must hold that $(1+\epsilon)|I| \leq 2c\alpha^* S_{q,k} + |I|$. Using Lemma 15, this simplifies to $c \leq 1 + 2/\epsilon$, which is a contradiction to $c > 1 + 2/\epsilon$. □

The previous lemmas proves that SSF-W is a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm for minimizing the maximum delay factor in broadcast scheduling with uniform sized pages when $c = 1 + 3/\epsilon$.

Non-Uniform Sized Pages

Here we extend our ideas to the case where pages can have non-uniform sizes for the objective of minimizing the maximum delay factor. For this problem, in [32] we developed a generalization of SSF-W and showed that it is a $(4+\epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm. Later, we gave a $(2+\epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm using a similar proof technique as used in the uniform page size setting given in the paper [31]. In this chapter we show a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^3})$-competitive online algorithm by improving our analyses based on the technique developed by Bansal, Krishnaswamy and Nagarajan [8] which considers *fractional* schedules.

We elaborate on the model for non-uniform page sizes. Each page p has size l_p, which we assume for simplicity is an integer. Page p consists of uniform sized pieces $(1,p),(2,p),\ldots(l_p,p)$. In a time slot one piece of a unique page can be broadcast by the server. A request $J_{p,i}$ is satisfied if it receives each of the pieces of page p in *sequential* order; in other words, we assume that the clients do not buffer the pieces if they are sent out of order. We assume preemption is allowed, and pieces of different pages can be interspersed. We call this model the integral broadcast setting. We now describe a relaxed notion of a schedule that we call a fractional schedule. In a fractional schedule the pieces of a page p are indistinguishable and the only relevant information are the time slots during which p is transmitted. Now a request $J_{p,i}$ is satisfied once l_p pieces of page p have been broadcast. That is, $J_{p,i}$ is satisfied once the server devotes l_p time slots to page p after $r_{p,i}$. A reduction in [8] gives a scheme that translates an algorithm with 1 speed for the fractional broadcast setting into a $(1+\epsilon')$-speed integer

schedule where the flow time, $C_i - r_i$, of each request increases by a factor of at most $\frac{8}{\epsilon'}$ for any fixed $\epsilon' > 0$. Using this technique, any schedule that is s-speed c-competitive for the fractional setting can be converted online into a schedule that is $s(1+\epsilon')$-speed $O(\frac{c}{\epsilon'})$-competitive for the integral setting. The algorithm used in [8] simulates the fractional schedule. The algorithm gives priorities to pages based on the unsatisfied requests for the page. The priority of a request is based on the flow time of the request in the fractional schedule; a smaller flow time corresponds to higher priority. Generally, the algorithm broadcasts the page with the highest priority unsatisfied request. We now restrict our attention to the fractional model.

We outline the details of modifications to SSF-W. As before, at any time t, the algorithm considers the alive requests $Q(t)$ and a subset $Q'(t)$ of requests that have waited sufficiently long; that is those with current delay factor at least $\frac{1}{c}\alpha_t$ where α_t is the maximum current delay factor for requests in $Q(t)$. Among all requests in $Q'(t)$ the algorithm picks the one with the smallest slack and broadcasts a unit amount of the page for that request; ties are broken arbitrarily. Recall that since the fractional setting is considered, the algorithm only needs to specify the page being broadcast and not the piece of the page. The algorithm may preempt the broadcast of p that is forced by request $J_{p,i}$ if another request $J_{p',j}$ becomes available for scheduling such that $S_{p',j} < S_{p,i}$. A key issue that differentiates the algorithms in [32, 31] from the one here is that those algorithms directly generate an integral schedule; therefore they have to not only specify the page but also the piece of the page. In particular the algorithms in [32, 31] could preempt the transmission of a page p for a request $J_{p,i}$ and transmit p again from the start if a new request $J_{p,i'}$ for p arrives and has much smaller slack than that of $J_{p,i}$. In the sequential model this means that the work done for $J_{p,i}$ is "wasted" and it would be satisfied at the same time as $J_{p,i'}$. In the fractional setting and the algorithm considered here $J_{p,i}$ would continue to be satisfied as if $J_{p,i'}$ did not arrive and prior transmission would not be wasted.

We now analyze the algorithm assuming that it has a $(1+\epsilon)$-speed advantage

over the optimal offline algorithm. As before, let σ be an arbitrary sequence of requests. We let OPT denote some fixed offline optimum schedule and let α^* denote the optimum delay factor. Let $c > 1+\frac{3}{\epsilon}$ be the constant that parameterizes SSF-W. We will show that $\alpha^{\text{SSF-W}} \leq c^2\alpha^*$. For the sake of contradiction, suppose that SSF-W witnesses a delay factor greater than $c^2\alpha^*$. We consider the *first* time t^* when SSF-W has some request in its queue with delay factor $c^2\alpha^*$. Let the request $J_{q,k}$ be a request which achieves the delay factor $c^2\alpha^*$ at time t^*. Let t_1 be the smallest time less than t^* such that at each time t during the interval $(t_1, t^*]$ if SSF-W is forced to broadcast by request $J_{p,i}$ at time t it is the case that $\frac{t-r_{p,i}}{S_{p,i}} > \alpha^*$ and $S_{p,i} \leq S_{q,k}$. Again, let $I = [t_1, t^*]$. Notice that some requests that force SSF-W to broadcast during I could have started being satisfied before t_1. We now show a lemma analogous to Lemma 14. We say that two requests for the same page p are satisfied simultaneously at time t if both requests are unsatisfied prior to t, p is broadcast at time t, and both requests receive l_p units of p after their arrival.

Lemma 18. *Consider any two distinct requests $J_{x,j}$ and $J_{x,i}$ for some page x. If $J_{x,j}$ forces* SSF-W *to broadcast during I before $r_{x,i}$, then* OPT *cannot satisfy $J_{x,j}$ and $J_{x,i}$ simultaneously at any time.*

Proof. Let t' be the time that SSF-W is forced to broadcast page x by $J_{x,j}$ where $t' < r_{x,i}$. By the definition of I, request $J_{x,j}$ must have delay factor greater than α^* at time t'. Hence, if OPT satisfies $J_{x,i}$ and $J_{x,j}$ simultaneously then $J_{x,j}$ will have delay factor strictly larger than α^* in OPT, a contradiction. □

The following Lemma 19, 20 can be proved in the same way Lemma 15, 16 are proved. This is because the algorithm SSF-W is designed to be oblivious to page sizes. Indeed, the key definitions in the proofs including the first time t^* that SSF-W witnesses delay factor $c^2\alpha^*$, the request $J_{q,k}$ that has delay factor $c^2\alpha^*$ at t^*, and $t' = t^* - (c^2 - c)S_{q,k}\alpha^*$ stay the same regardless of whether pages have a uniform size or not. The advantage of thinking about a fractional schedule is that the work conservation argument can be applied once we have Lemma 18.

Lemma 19. $|I| = |[t_1, t^*]| \geq (c^2 - c) S_{q,k} \alpha^*$.

Lemma 20. *Any request which forced* SSF-W *to schedule a page during I must have arrived after time $t_1 - 2S_{q,k}\alpha^* c$.*

Using the previous lemmas we can bound the competitiveness of SSF-W.

Lemma 21. *Suppose c is a constant s.t. $c > 1 + 2/\epsilon$. If* SSF-W *has $(1+\epsilon)$-speed then $\alpha^{\text{SSF-W}} \leq c^2 \alpha^*$.*

Proof. For the sake of contradiction, suppose that $\alpha^{\text{SSF-W}} > c^2 \alpha^*$. During the interval I, the number of broadcasts which SSF-W transmits is $(1+\epsilon)|I|$. From Lemma 20, all the requests that forced SSF-W to broadcast during I have arrived no earlier than $t_1 - 2c\alpha^* S_{q,k}$. We know that the optimal solution must process these requests before time t^* because these requests have delay factor greater than α^* by t^*. By Lemma 18 the optimal solution cannot simultaneously satisfy two requests $J_{x,i}$ and $J_{x,j}$ that forced SSF-W to broadcast during I if $r_{x,i}$ is later than when $J_{x,i}$ forced SSF-W to broadcast. This implies the optimal solution must broadcast at least $(1+\epsilon)|I|$ units in $2c\alpha^* S_{q,k} + |I|$ time steps. Thus, it must hold that $(1+\epsilon)|I| \leq 2c\alpha^* S_{q,k} + |I|$. Using Lemma 19, this simplifies to $c \leq 1 + 2/\epsilon$, which is a contradiction to $c > 1 + 2/\epsilon$. □

By setting $c = 1 + 3/\epsilon$ we have the following theorem.

Theorem 22. *The algorithm* SSF-W *is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive for minimizing the maximum delay factor in a fractional schedule when pages have non-uniform sizes.*

Using the reduction of [8] previously discussed, we have shown the second part of Theorem 12.

3.4 Minimizing the Maximum Weighted Flow Time and Weighted Delay Factor

We show the connection of our analysis of SSF and SSF-W to the problem of minimizing the maximum *weighted* flow time. In the standard setting for the problem of minimizing the maximum weighted flow time, requests do not have deadlines, but have weights. Each request J_i has a positive weight w_i. The goal is to minimize $\max_i w_i(C_i - r_i)$. Similar to the algorithm SSF, we give the following algorithm called BWF (Biggest-Weight-First). The algorithm BWF always schedules the request with the largest weight.

Algorithm: BWF

- Let $Q(t)$ be the set of alive requests at t.
- Let J_i be the request with the largest weight among requests in $Q(t)$, ties broken by arrival time.
- Preempt the current request and schedule J_i if it is not being processed.

Similarly we can think of the problem of minimizing the maximum weighted delay factor. Here a request J_i has a deadline d_i and a weight w_i. The objective now is to minimize $\max_i w_i \max\{1, \frac{C_i - r_i}{S_i}\}$. Let the modified weight of a request J_i be defined as $w'_i = w_i/S_i$. The algorithm BWF above when implemented with modified weights is the algorithm SRF (Smallest Ratio First). It is not difficult to adapt the analysis for SSF in Section 3.2 to show that BWF is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive for the problem of minimizing the maximum weighted flow time, and that SRF is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive for the problem of minimizing the maximum weighted delay factor.

Now consider the problem of minimizing the maximum weighted response time in broadcast scheduling when pages have uniform sizes. A request $J_{p,i}$ has a weight

$w_{p,i}$. The goal is to minimize the maximum weighted flow time $\max_{p,i} w_{p,i}(C_{p,i} - r_{p,i})$. We extend the algorithm BWF to get an algorithm called BWF-W for Biggest-Wait-First with Waiting. The algorithm is parameterized by a constant $c > 1$. At any time t before broadcasting a page, BWF-W determines the largest weighted wait time of any request which has yet to be satisfied. Let this value be ρ_t. The algorithm then chooses to broadcast a page corresponding to the request with largest weight amongst the requests whose current weighted wait time at time t is larger than $\frac{1}{c}\rho_t$.

Algorithm: BWF-W

- The algorithm is non-preemptive. Let t be a time that the machine is free (either because a request has just finished or there are no requests to process).
- Let $Q(t)$ be the set of alive requests at time t and let $\rho_t = \max_{J_{p,i} \in Q(t)} w_{p,i}(t - r_{p,i})$.
- Let $Q'(t) = \{J_{p,i} \in Q(t) \mid w_{p,i}(t - r_{p,i}) \geq \frac{1}{c}\rho_t\}$.
- Let $J_{p,i}$ be the request in $Q'(t)$ with the largest weight. Broadcast page p non-preemptively.

Although minimizing the maximum delay factor and minimizing the maximum weighted flow time are very similar metrics, the problems are not equivalent. It may also be of interest to minimize the maximum *weighted* delay factor. In this setting each request has a deadline and a weight. The goal is to minimize $\max_{p,i} w_{p,i} \max\{1, \frac{C_{p,i} - r_{p,i}}{S_{p,i}}\}$. For this setting we can simply alter BWF-W where we use modified weights for requests: $w'_{p,i}$ for request $J_{p,i}$ is defined as $w_{p,i}/S_{p,i}$. We call the resulting algorithm SRF-W (Smallest-Ratio-First with Waiting).

For the problems of minimizing the maximum weighted flow time and weighted delay factor, the upper bounds shown for SSF-W in this chapter also hold for

BWF and SRF-W, respectively. The analysis of BWF and SRF-W is very similar to that of SSF-W. To illustrate how the proofs extend, we prove that SRF-W is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive for minimizing the maximum weighted delay factor in broadcast scheduling where all pages are uniform sized and there is a single machine. This is the most general problem discussed as it is a generalization of weighted flow time and the delay factor. We analyze SRF-W when it is given a $(1+\epsilon)$-speed machine. Let $c > 1 + \frac{2}{\epsilon}$ be the constant which parameterizes SRF-W. Let σ be an arbitrary sequence of requests. We let OPT denote some fixed offline optimum schedule and let α^* and $\alpha^{\text{SRF-W}}$ denote the maximum weighted delay factor achieved by OPT and SRF-W, respectively. We will show that $\alpha^{\text{SRF-W}} \leq c^2\alpha^*$. For the sake of contradiction, suppose that SRF-W witnesses a weighted delay factor greater than $c^2\alpha^*$. We consider the *first* time t^* when SRF-W has some request in its queue with weighted delay factor $c^2\alpha^*$. Let the request $J_{q,k}$ be a request which achieves the weighed delay factor $c^2\alpha^*$ at time t^*. Let t_1 be the smallest time less than t^* such that at each time t during the interval $(t_1, t^*]$ if SRF-W is forced to broadcast by request $J_{p,i}$ at time t it is the case that $\frac{w_{p,i}(C_{p,i}-r_{p,i})}{S_{p,i}} > \alpha^*$ and $\frac{S_{p,i}}{w_{p,i}} \leq \frac{S_{q,k}}{w_{q,k}}$. Throughout this section we let $I = [t_1, t^*]$.

We let $\mathcal{J}_I$ denote the requests which forced SRF-W to schedule broadcasts during the interval $[t_1, t^*]$. We now show that any two request in $\mathcal{J}_I$ cannot be satisfied with a single broadcast by the optimal solution.

Lemma 23. OPT *cannot merge any two requests in* $\mathcal{J}_I$ *into a single broadcast.*

Proof. Let $J_{x,i}, J_{x,j} \in \mathcal{J}_I$ such that $i < j$. Let t' be the time that SRF-W satisfies request $J_{x,i}$. By the definition of I, request $J_{x,i}$ must have weighted delay factor greater than α^* at this time. We also know that the request $J_{x,j}$ must arrive after time t', otherwise request $J_{x,j}$ must also be satisfied at time t'. If the optimal solution combines these requests into a single broadcast then the request $J_{x,i}$ must wait until the request $J_{x,j}$ arrives to be satisfied. However, this means that the request $J_{x,i}$ must achieve a weighted delay factor greater than α^* by OPT, a

contradiction. □

As in the analysis of SSF-W we show that interval I is sufficiently long.

Lemma 24. $|I| = |[t_1, t^*]| \geq (c^2 - c)\frac{S_{q,k}}{w_{q,k}}\alpha^*$.

Proof. The request $J_{q,k}$ has weighted delay factor $c\alpha^*$ at time $t' = t^* - (c^2 - c)\frac{S_{q,k}}{w_{q,k}}\alpha^*$. The largest weighted delay factor any request can have at time t' is less than $c^2\alpha^*$ by definition of t^* being the first time SRF-W witnesses weighted delay factor $c^2\alpha^*$. Hence, $\alpha_{t'} \leq c^2\alpha^*$. Thus, the request $J_{q,k}$ is in $Q(t')$ because $c\alpha^* \geq \frac{1}{c}\alpha_{t'}$. Moreover, this means that any request that forced SRF-W to broadcast during $[t', t^*]$, must have weighted delay factor greater than α^* and since $J_{q,k} \in Q(t')$, the requests scheduled during $[t', t^*]$ must have a ratio of slack over weight of at most $\frac{S_{q,k}}{w_{q,k}}$. □

Lemma 25. *Any request in $\mathcal{J}_I$ must have arrived after time $t_1 - 2\frac{S_{q,k}}{w_{q,k}}\alpha^* c$.*

Proof. For the sake of contradiction, suppose that some request $J_{p,i} \in \mathcal{J}_I$ arrived at time $t' < t_1 - 2\frac{S_{q,k}}{w_{q,k}}\alpha^* c$. Recall that $\frac{S_{p,i}}{w_{p,i}} \leq \frac{S_{q,k}}{w_{q,k}}$ by the definition of I. Therefore at time $t_1 - \frac{S_{q,k}}{w_{q,k}}\alpha^* c$, $J_{p,i}$ has a weighted delay factor greater than $c\alpha^*$. Thus any request scheduled during the interval $I' = [t_1 - \frac{S_{q,k}}{w_{q,k}}\alpha^* c, t_1]$ has a weighted delay factor greater than α^*. We observe that $J_{p,i}$ is in $Q(\tau)$ for $\tau \in I'$; otherwise there must be a request with weighted delay factor bigger than $c^2\alpha^*$ at time τ and this is a contradiction to the assumption that t^* is the first time that SRF-W witnessed a weighted delay factor of $c^2\alpha^*$. Therefore any request scheduled during I' has a slack over weight no bigger than $\frac{S_{p,i}}{w_{p,i}}$. Also we know that $\frac{S_{p,i}}{w_{p,i}} \leq \frac{S_{q,k}}{w_{q,k}}$. In sum, we showed that any request done during I' had slack over weight no bigger than $\frac{S_{q,k}}{w_{q,k}}$ and a delay factor greater than α^*, which is a contradiction to the definition of t_1. □

Now we are ready to prove the competitiveness of SRF-W.

Lemma 26. *Suppose c is a constant s.t. $c > 1 + 2/\epsilon$. If* SRF-W *has $(1+\epsilon)$-speed then $\alpha^{\text{SRF-W}} \leq c^2\alpha^*$.*

Proof. For the sake of contradiction, suppose that $\alpha^{\text{SRF-W}} > c^2\alpha^*$. During the interval I, the number of broadcasts which SRF-W transmits is $(1+\epsilon)|I|$. From Lemma 25, all the requests processed during I have arrived no earlier than $t_1 - 2c\alpha^*\frac{S_{q,k}}{w_{q,k}}$. We know that the optimal solution must process these requests before time t^* because these requests have weighted delay factor greater than α^* by t^*. By Lemma 23 the optimal solution must make a unique broadcast for each of these requests. Thus, the optimal solution must finish all of these requests in $2c\alpha^*\frac{S_{q,k}}{w_{q,k}} + |I|$ time steps. Thus it must hold that $(1+\epsilon)|I| \leq 2c\alpha^*\frac{S_{q,k}}{w_{q,k}} + |I|$. Using Lemma 24, this simplifies to $c \leq 1+2/\epsilon$, which is a contradiction to $c > 1+2/\epsilon$. □

3.5 Conclusions

We considered online scheduling to minimize maximum (weighted) flow time and to minimize maximum (weighted) delay factor. Delay factor and the weighted flow time metrics have not been previously considered. We developed scalable algorithms for these metrics in both the standard and broadcast scheduling models. Our algorithms demonstrate an interesting trade off on whether to prioritize requests with larger weight or those that have waited longer in the system. Understanding this trade off has led to the first online scalable algorithm for minimizing average flow time in broadcast scheduling [48] which has been an open problem for several years.

We close with the following open problems. Our algorithm for the maximum delay factor with uniform sized pages uses a parameter that explicitly depends on the speed given to the algorithm. Is there an algorithm that is scalable without needing this information? FIFO is 2-competitive for minimizing maximum flow time in broadcast scheduling. In the offline setting can the 2-approximation implied by FIFO be improved? For the more general problem of minimizing maximum delay factor, no non-trivial offline approximation is known that does not use resource augmentation.

Chapter 4

Longest Wait First and its Variants in the Broadcast Model

4.1 Introduction

In this chapter we focus on the LWF algorithm in the broadcast setting of unit-sized pages. See chapter 2 for motivation and related work. In addition to being a natural greedy policy, LWF has been shown to outperform other natural policies [2]; moreover, related variants are known to be work well in certain stochastic settings. It is, therefore, of interest to better understand its performance. Our results on LWF in this chapter are for unit-size pages. We make the following contributions.

- We give a simple and intuitive analysis of LWF that already improves the speed bound in [39]; the analysis shows that LWF is 5-speed $O(1)$-competitive for average flow time. Here we present this similar analysis. In [30] we show using a more complex analysis that LWF is $(3.4 + \epsilon)$-speed $O(1/\epsilon^3)$-competitive.

- We show that a natural generalization of LWF that we call LF is $O(k)$-speed $O(k)$-competitive for minimizing the ℓ_k norm of flow time — these bounds extend to average delay factor and ℓ_k norms of delay factor. These are the first non-trivial results for ℓ_k norms in broadcast scheduling for $k > 1$.

Our results show that LWF-like algorithms have reasonable theoretical performance even for these more difficult metrics. We derive these additional results in a unified fashion via a general framework that is made possible by our analysis for LWF. We go on to show the following,

- For any constants $s, c > 1$, LF is not c-competitive with s-speed for minimizing maximum delay factor (or weighted response time) in standard scheduling of uniform sized requests.

This lower bound suggests that LF may require a speed that increases with k to obtain $O(1)$-competitiveness for ℓ_k norms.

Our analysis of LWF borrows several key ideas from [39], however, we make some crucial simplifications. We outline the main differences in Section 4.1.1 where we give a brief overview of our approach. The analysis techniques from the work presented in this chapter helped to lead to the first scalable algorithm for average flow time in the broadcast setting. As mentioned, this scalable algorithm and its analysis are presented in the next chapter. This work also shows the first algorithm with non-trivial worst case performance guarantees for the ℓ_k-norms of flow time. Later in this thesis, a scalable algorithm is presented for the ℓ_k-norms of flow time.

In this chapter we assume, for simplicity, the discrete time model. In this model, at each integer time t, the following things happen exactly in the following order; the scheduler make a decision of which page p to broadcast; the page p is broadcast and all outstanding requests of page p are immediately satisfied, thus having finish time t; new requests arrive. Note that new pages which arrive at t are not satisfied by the broadcasting at the time t. It is important to keep it in mind that all these things happen only at integer times. See [39] for more discussion on discrete time versus continuous time models.

4.1.1 Overview of the Analysis of LWF

We give a high level overview of our analysis of LWF. Let OPT denote some fixed optimal 1-speed offline solution; we overload notation and use OPT also to denote the value of the optimal schedule. Recall that for simplicity of analysis, we assume the discrete-time model in which requests arrive at integer times. For the same reason we analyze LWF with an integer speed $s > 1$. We can assume

that LWF is never idle. Thus, in each time step LWF broadcasts s pages and the optimal solution broadcasts 1 page. We also assume that requests arrive at integer times. At time t, a request is in the set $U(t)$ if it is unsatisfied by the scheduler at time t. In the broadcast setting LWF with speed s is defined as the following.

Algorithm: LWF_s

- At any integer time t, broadcast the s pages with the largest waiting times, where the waiting time of page p is $\sum_{J_{p,i} \in U(t)} (t - r_{p,i})$.

Our analysis of LWF is inspired by that in [39]. Here we summarize our approach and indicate the main differences from the analysis in [39]. Given the schedule of LWF_s on a request sequence σ, the requests are partitioned into two disjoint sets S (self-chargeable requests) and N (non-self-chargeable requests). Let the total flow time accumulated by LWF_s for requests in S and N be denoted by LWF_s^S and LWF_s^N respectively. Likewise, let OPT^S and OPT^N be the flow-time OPT accumulates for requests in S and N, respectively. S is the set of requests whose flow-time is comparable to their flow-time in OPT. Hence one immediately obtains that $\text{LWF}_s^S \leq \rho \text{OPT}^S$ for some constant ρ. For requests in N, instead of charging them only to the optimal solution, these requests are charged to the total flow time accumulated by LWF *and* OPT. It will be shown that $\text{LWF}_s^N \leq \delta \text{LWF}_s + \rho \text{OPT}^N$ for some $\delta < 1$; this is crux of the proof. It follows that $\text{LWF}_s = \text{LWF}_s^S + \text{LWF}_s^N \leq \rho \text{OPT}^S + \rho \text{OPT}^N + \delta \text{LWF} \leq \rho \text{OPT} + \delta \text{LWF}$. This shows that $\text{LWF}_s \leq \frac{\rho}{1-\delta} \text{OPT}$, which will complete our analysis. Perhaps the key idea in [39] is the idea of charging LWF_s^N to LWF_s with a $\delta < 1$; as shown in [53], no algorithm for any constant speed can be locally competitive with respect to all adversaries and hence previous approaches in the non-broadcast scheduling context that establish local competitiveness with respect to OPT cannot work.

In [39], the authors do not charge LWF_s^N directly to LWF_s. Instead, they further split N into two types and do a much more involved analysis to bound the flow-time of the type 2 requests via the flow-time of type 1 requests. Moreover,

they first transform the given instance to canonical instance in a complex way and prove the correctness of the transformation. Our simple proof shows that these complex arguments can be done away with. We also improve the speed bounds and generalize the proof to other objective functions.

4.1.2 Preliminaries

To show that $\mathrm{LWF}_s^N \leq \delta \mathrm{LWF}_s + \rho \mathrm{OPT}^N$, we will map the requests in N to other requests scheduled by LWF_s which have comparable flow time. An issue that can occur when using a charging scheme is that one has to be careful not to overcharge. In this setting, this means for a single request $J_{p,i}$ we must bound of the number of requests in N which are charged to $J_{p,i}$. To overcome the overcharging issue, we will appeal to a generalization of Hall's theorem. Here we will have a bipartite graph $G = (X \cup Y)$ where the vertices in X will correspond to requests in N. The vertices in Y will correspond to all requests scheduled by LWF_s. A vertex $u \in X$ will be adjacent to a vertex $v \in Y$ if u and v have comparable flow time and v was satisfied while u was in our queue and unsatisfied; that is, u can be charged to v. We then use a simple generalization of Hall's theorem, which we call *Fractional Hall's Theorem*. Here a vertex of $u \in X$ is matched to a vertex of $v \in Y$ with weight $\gamma_{u,v}$ where $\gamma_{u,v}$ is not necessarily an integer. Note that a vertex can be matched to multiple vertices.

Definition 27 (c-covering)**.** *Let $G = (X \cup Y, E)$ be a bipartite graph whose two parts are X and Y, and let $\ell : E \to [0,1]$ be an edge-weight function. We say that ℓ is a c-covering if for each $u \in X$, $\sum_{(u,v)\in E} \gamma_{u,v} = 1$ and for each $v \in Y$, $\sum_{(u,v)\in E} \gamma_{u,v} \leq c$.*

The following lemma follows easily from either Hall's Theorem or the Max-Flow Min-Cut Theorem.

Lemma 28 (Fractional Hall's theorem)**.** *Let $G = (V = X \cup Y, E)$ be a bipartite graph whose two parts are X and Y, respectively. For a subset S of X, let $N_G(S) =$*

$\{v \in Y | uv \in E,\ u \in S\}$, *be the neighborhood of* S. *For every* $S \subseteq X$, *if* $|N_G(S)| \geq \frac{1}{c}|S|$, *then there exists a* c*-covering for* X.

Throughout this chapter we will discuss time intervals and unless explicitly mentioned we will assume that they are closed intervals with integer end points. When considering some contiguous time interval $I = [s, t]$ we will say that $|I| = t - s + 1$ is the length of interval I; in other words it is the number of integers in I. For simplicity, we abuse this notation; when X is a set of closed intervals, we let $|X|$ denote the number of distinct integers in some interval of X. Note that $|X|$ also can be seen as the sum of the lengths of maximal contiguous sub-intervals if X is composed of non-overlapping intervals.

To be able to apply Lemma 28, we show another lemma which will be used throughout this chapter. Lemma 29 says that the union of some fraction of time intervals is comparable to that of the whole time interval.

Lemma 29. *Let* $X = \{[s_1, t_1], \ldots, [s_k, t_k]\}$ *be a finite set of closed intervals and let* $X' = \{[s'_1, t_1], \ldots, [s'_k, t_k]\}$ *be an associated set of intervals such that for* $1 \leq i \leq k$, $s'_i \in [s_i, t_i]$ *and* $|[s'_i, t_i]| \geq \lambda |[s_i, t_i]|$. *Then* $|X'| \geq \lambda |X|$.

Proof. Let I be the union of all intervals in X. I' is similarly defined for X'. We prove the lemma when I' is a contiguous interval; otherwise we can simply sum over all maximal intervals in I'. WLOG, we can set $I = [s_1, t']$ and $I' = [s', t']$. This is because I must start with one interval in X, say $[s_1, t_1]$ and both I and I' must have the same ending point t' by construction. Since $s \leq s'_1$, it is enough to show that $\frac{t - s'_1 + 1}{t - s_1 + 1} \geq \lambda$ and it follows from the given condition that $|[s'_1, t_1]| \geq \lambda |[s_1, t_1]|$, (i.e. $t_1 - s'_1 + 1 \geq \lambda(t_1 - s_1 + 1)$) and $t \geq t_1$. □

4.2 Minimizing Average Flow Time

We focus our attention on minimizing average flow time. A fair amount of notation is needed to clearly illustrate our ideas. Following [39], for each page, we will partition time into intervals via *events*. Events for page p are defined by LWF_s's

broadcasts of page p. When LWF_s broadcasts page p a new event occurs. An event x for page p will be defined as $E_{p,x} = \langle b_{p,x}, e_{p,x} \rangle$ where $b_{p,x}$ is the beginning of the event and $e_{p,x}$ is the end. Here LWF_s broadcast page p at time $b_{p,x}$ and this is the xth broadcast of page p. Then LWF_s broadcast page p at time $e_{p,x}$ and this is the $(x+1)$st broadcast of page p. This starts a new event $E_{p,x+1}$. Therefore, the algorithm LWF_s does not broadcast p on the time interval $[b_{p,x}+1, e_{p,x}-1]$. Thus, it can be seen that for page p, $e_{p,x-1} = b_{p,x}$. It is important to note that the optimal offline solution may broadcast page p multiple (or zero) times during an event for page p. See Figure 5.2.

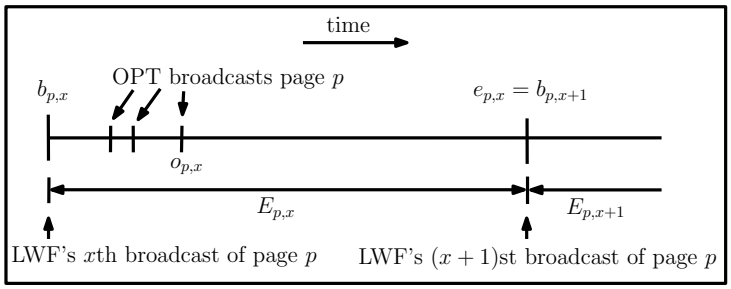


Figure 4.1: Events for page p.

For each event $E_{p,x}$ we let $\mathcal{J}_{p,x} = \{(p,i) \mid r_{p,i} \in [b_{p,x}, e_{p,x}-1]\}$ denote the set of requests for p that arrive in the interval $[b_{p,x}, e_{p,x}-1]$ and are satisfied by LWF_s at $e_{p,x}$. We let $F_{p,x}$ denote the flow-time in LWF_s of all requests in $\mathcal{J}_{p,x}$. Similarly we define $F^*_{p,x}$ to be flow time in OPT for all requests in $\mathcal{J}_{p,x}$. Note that OPT may or may not satisfy requests in $\mathcal{J}_{p,x}$ during the interval $[b_{p,x}, e_{p,x}]$.

An event $E_{p,x}$ is said to be self-chargeable and in the set S if $F_{p,x} \leq F^*_{p,x}$ or $e_{p,x} - b_{p,x} < \rho$, where $\rho > 1$ is a constant which will be fixed later. Otherwise the event is non-self-chargeable and is in the set N. Implicitly we are classifying the requests as self-chargeable or non-self-chargeable, however it is easier to work with events rather than individual requests. As the names suggest, self-chargeable events can be easily charged to the flow-time of an optimal schedule. To help analyze the flow-time for non-chargeable events, we set up additional notation and further refine the requests in N.

Consider a non-self-chargeable event $E_{p,x}$. Note that since this event is non-

self-chargeable, the optimal solution must broadcast page p during the interval $[b_{p,x}+1, e_{p,x}-1]$; otherwise, $F_{p,x} \leq F^*_{p,x}$ and the event is self-chargeable. Let $o_{p,x}$ be the last broadcast of page p by the optimal solution during the interval $[b_{p,x}+1, e_{p,x}-1]$. We define $o'_{p,x}$ for a non-self-chargeable event $E_{p,x}$ as $\min\{o_{p,x}, e_{p,x}-\rho\}$. This ensures that the interval $[o'_{p,x}, e_{p,x}]$ is sufficiently long; this is for technical reasons and the reader should think of $o'_{p,x}$ as essentially the same as $o_{p,x}$.

Let $\mathrm{LWF}^S_s = \sum_{p,x:E_{p,x}\in S} F_{p,x}$ and $\mathrm{LWF}^N_s = \sum_{p,x:E_{p,x}\in N} F_{p,x}$ denote the the total flow time for self-chargeable and non self-chargeable events respectively. Similarly, let $\mathrm{OPT}^S = \sum_{p,x:E_{p,x}\in S} F^*_{p,x}$ and $\mathrm{OPT}^N = \sum_{p,x:E_{p,x}\in N} F^*_{p,x}$. For a non-chargeable event $E_{p,x}$ we divide $\mathcal{J}_{p,x}$ into early requests and late requests depending on whether the request arrives before $o'_{p,x}$ or not. Letting $F^e_{p,x}$ and $F^l_{p,x}$ denote the flow-time of early and late requests respectively, we have $F_{p,x} = F^e_{p,x} + F^l_{p,x}$. Let $\mathrm{LWF}^{N^e}_s$ and $\mathrm{LWF}^{N^l}_s$ denote the total flow time of early and late requests of non-self-chargeable events for LWF's schedule, respectively.

The following two lemmas follow easily from the definitions.

Lemma 30. $\mathrm{LWF}^S_s \leq \rho\mathrm{OPT}^S$.

Lemma 31. $\mathrm{LWF}^{N^l}_s \leq \rho\mathrm{OPT}^N$.

Thus the main task is to bound $\mathrm{LWF}^{N^e}_s$. For a non-chargeable event $E_{p,x}$ we try to charge $F^e_{p,x}$ to events ending in the interval $[o'_{p,x}, e_{p,x}-1]$. The lemma below quantifies the relationship between $F^e_{p,x}$ and the flow-time of events ending in this interval.

Lemma 32. *For any* $0 \leq \lambda \leq 1$, *if* $e_{q,y} \in [\lceil o'_{p,x} + \lambda(e_{p,x} - o'_{p,x})\rceil, e_{p,x}-1]$ *then* $F_{q,y} \geq \lambda F^e_{p,x}$.

Proof. Let $F_{p,x}(t)$ be the total waiting time accumulated by LWF for page p on the time interval $[b_{p,x}, t]$. We divide $F_{p,x}(t)$ into two parts $F^e_{p,x}(t)$ and $F^l_{p,x}(t)$, which are the flow time due to early requests and to late requests, respectively. Note that $F_{p,x}(t) = F^e_{p,x}(t) + F^l_{p,x}(t)$. The early requests arrived before time $o'_{p,x}$, thus, for any $t' \geq \lceil o'_{p,x} + \lambda(e_{p,x} - o'_{p,x})\rceil$, $F^e_{p,x}(t') \geq \lambda F^e_{p,x}(e_{p,x}) = \lambda F^e_{p,x}$.

Since LWF_s chose to transmit q at $e_{q,y}$ when p was available to be transmitted, it must be the case that $F_{q,y} \geq F_{p,x}(e_{q,y}) \geq F^e_{p,x}(e_{q,y})$. Combining this with the fact that $F^e_{p,x}(e_{q,y}) \geq \lambda F^e_{p,x}$, the lemma follows.

□

With the above setup in place, we now prove that LWF_s is $O(1)$ competitive for $s = 5$ via a clean and simple proof. This proof can be extended to non-integer speeds with better bounds on the speed. In particular, it can be shown that $\text{LWF}_{3.4+\epsilon}$ is $O(1/\epsilon^3)$-competitive.

4.2.1 Analysis of 5-Speed

This section will be devoted to proving the following main lemma that bounds the flow-time of early requests of non self-chargeable events.

Lemma 33. *For $\rho \geq 1$, $\text{LWF}_5^{N^e} \leq \frac{4\rho}{5(\rho-1)}\text{LWF}_5$.*

Assuming the lemma, LWF_5 is $O(1)$-competitive, using the argument outlined earlier in Section 4.1.1.

Theorem 34. $\text{LWF}_5 \leq 90\text{OPT}$.

Proof. By combining Lemma 30, 31 and 33, we have that $\text{LWF}_5 = \text{LWF}_5^S + \text{LWF}_5^{N^l} + \text{LWF}_5^{N^e} \leq \rho\text{OPT}^S + \rho\text{OPT}^N + \frac{4\rho}{5(\rho-1)}\text{LWF}_5$. Setting $\rho = 10$ completes the proof. □

We now prove Lemma 33. In the analysis, we assume that LWF broadcasts 5 pages at each time; otherwise we can apply the same argument to maximal subintervals when LWF is fully busy, respectively. Let $E_{p,x} \in N$. We define two intervals $I_{p,x} = [o'_{p,x}, e_{p,x} - 1]$ and $I'_{p,x} = [o'_{p,x} + \lceil (e_{p,x} - o'_{p,x})/2 \rceil, e_{p,x} - 1]$. Since $\rho \leq e_{p,x} - o'_{p,x}$, it follows that $|I'_{p,x}| \geq \frac{\rho-1}{2\rho}|I_{p,x}|$. We wish to charge $F^e_{p,x}$ to events (could be in S or N) in the interval $I'_{p,x}$. By Lemma 32, each event $E_{q,y}$ that finishes in $I'_{p,x}$ satisfies the property that $F_{q,y} \geq F^e_{p,x}/2$. Moreover, there are $5(\lfloor e_{p,x} - o'_{p,x})/2 \rfloor$ such events to charge to since LWF_5 transmits 5 pages in each

time slot. Thus, locally for $E_{p,x}$ there are enough events to charge to if ρ is a sufficiently large constant. However, an event $E_{q,y}$ with $e_{q,y} \in I'_{p,x}$ may also be charged by many other events if we follow this simple local charging scheme. To overcome this overcharging, we resort to a global charging scheme by setting up a bipartite graph G and invoking the fractional Hall's theorem (see Lemma 28) on this graph.

The bipartite graph $G = (X \cup Y, E)$ is defined as follows. There is exactly one vertex $u_{p,x} \in X$ for each non-self-chargeable event $E_{p,x} \in N$ and there is exactly one vertex $v_{q,y} \in Y$ for each event $E_{q,y} \in A$, where A is the set of all events. Consider two vertices $u_{p,x} \in X$ and $v_{q,y} \in Y$. There is an edge $u_{p,x}v_{q,y} \in E$ if and only if $e_{q,y} \in I'_{p,x}$. By Lemma 32, if there is an edge between $u_{p,x} \in X$ and $v_{q,y} \in Y$ then $F_{q,y} \geq F^e_{p,x}/2$.

The goal is now to show that G has a $\frac{2\rho}{5(\rho-1)}$-covering. Consider any non-empty set $Z \subseteq X$ and a vertex $u_{p,x} \in Z$. Recall that the interval $I_{p,x}$ contains at least one broadcast by OPT of page p. Let $\mathcal{I} = \bigcup_{u_{p,x} \in Z} I_{p,x}$ be the union of the time intervals corresponding to events in Z. Similarly, define $\mathcal{I}' = \bigcup_{u_{p,x} \in Z} I'_{p,x}$.

We claim that $|Z| \leq |\mathcal{I}|$. This is because the optimal solution has 1-speed and it has to do a separate broadcast for each event in Z during $\mathcal{I}$. Now consider the neighborhood of Z, $N_G(Z)$. We note that $|N_G(Z)| = 5|\mathcal{I}'|$ since LWF_5 broadcasts 5 pages for each time slot in $|\mathcal{I}'|$ and each such broadcast is adjacent to an event in Z from the definition of G. From Lemma 29, $|\mathcal{I}'| \geq \frac{\rho-1}{2\rho}|\mathcal{I}|$ as we had already observed that $|I'_{p,x}| \geq \frac{\rho-1}{2\rho}|I_{p,x}|$ for each $E_{p,x} \in N$. Thus we conclude that $|N_G(Z)| = 5|\mathcal{I}'| \geq 5\frac{\rho-1}{2\rho}|\mathcal{I}| \geq 5\frac{\rho-1}{2\rho}|Z|$. Since this holds for $\forall Z \subseteq X$, by Lemma 28, there must exist a $\frac{2\rho}{5(\rho-1)}$-covering. Let ℓ be such a covering. Finally, we prove that the covering implies the desired bound on $\text{LWF}_5^{N^e}$.

$$\begin{aligned}
\mathrm{LWF}_5^{N^e} &= \sum_{u_{p,x}\in X} F^e_{p,x} \text{ [By Definition]} \\
&= \sum_{u_{p,x}v_{q,y}\in E} \gamma_{u_{p,x},v_{q,y}} F^e_{p,x} \text{ [By Def. 27, i.e. for } \forall u_{p,x}\in X,\ \textstyle\sum_{v_{q,y}\in Y} \gamma_{u_{p,x},v_{q,y}}=1] \\
&\leq \sum_{u_{p,x}v_{q,y}\in E} \gamma_{u_{p,x},v_{q,y}} 2F_{q,y} \text{ [By Lemma 32]} \\
&\leq \frac{4\rho}{5(\rho-1)} \sum_{v_{q,y}\in Y} F_{q,y} \text{ [Change order of } \textstyle\sum \text{ and } \ell \text{ is a } \frac{2\rho}{5(\rho-1)}\text{-covering]} \\
&\leq \frac{4\rho}{5(\rho-1)} \mathrm{LWF}_5. \text{ [Since } Y \text{ includes all events]}
\end{aligned}$$

This finishes the proof of Lemma 33.

Remark 2. *If non-integer speeds are allowed then the analysis in this subsection can be extended to show that* LWF *is* $4+\epsilon$*-speed* $O(1+1/\epsilon^2)$*-competitive.*

4.3 Generalization to Delay-Factor and ℓ_k Norms

In this section, our proof techniques are extended to show that a generalization of LWF is $O(1)$-speed $O(1)$-competitive for minimizing the average delay-factor and minimizing the ℓ_k-norm of the delay-factor. Recall that flow-time can be subsumed as a special case of delay-factor. Thus, these results will also apply to ℓ_k norms of flow-time. Instead of focusing on specific objective functions, we develop a general framework and derive results for delay-factor and ℓ_k norms as special cases. First, we set up some notation. We assume that for each request $J_{p,i}$ there is a non-decreasing function $m_{p,i}(t)$ that gives the cost/penalty of that $J_{p,i}$ accumulates if it has *waited* for a time of t units after its arrival. Thus the total cost/penalty incurred for a schedule that finishes $J_{p,i}$ at $C_{p,i}$ is $m_{p,i}(C_{p,i}-r_{p,i})$. For flow-time $m_{p,i}(t)=t$ while for delay-factor it is $\max(1, \frac{t-r_{p,i}}{d_{p,i}-r_{p,i}})$. For ℓ_k norms of delay-factor we set $m_{p,i}(t)=\max(1, \frac{t-r_{p,i}}{d_{p,i}-r_{p,i}})^k$. Note that the ℓ_k norm of delay-factor for a given sequence of requests is $\sqrt[k]{\sum_{p,i} m_{p,i}(C_{p,i}-r_{p,i})}$ but we can ignore

the outer k'th root by focusing on the inner sum.

A natural generalization of LWF to more general metrics is described below; we refer to this (greedy) algorithm as LF for Longest First. We in fact describe LF_s which is given s speed over the adversary.

Algorithm: LF_s

- At any integer time t, broadcast the s pages with the largest m-waiting times where the m-waiting time of page p at t is $\sum_{J_{p,i} \in U(t)} m_{p,i}(t - r_{p,i})$.

Remark 3. *The algorithm and analysis do not assume that the functions $m_{p,i}$ are "uniform" over requests. In principle each request $J_{p,i}$ could have a different penalty function.*

In order to analyze LF, we need a lower bound on the "growth" rate of the functions $m_{p,i}()$. In particular we assume that there is a function $h : [0,1] \rightarrow \mathbb{R}^+$ such that $m_{p,i}(\lambda t) \geq h(\lambda) m_{p,i}(t)$ for all $\lambda \in [0,1]$. It is not to difficult to see that for flow-time and delay-factor we can choose $h(\lambda) = \lambda$, and for ℓ_k norms of flow-time and delay-factor, we can set $h(\lambda) = \lambda^k$. We also define a function $m : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ as $m(x) = \max_{(p,i)} m_{p,i}(x)$. The rest of the analysis depends only on h and m.

In the following subsection we outline a generalization of the analysis from Section 4.2.1 that applies to LF_s; the analysis bounds various quantities in terms of the functions $h()$ and $m()$. In Section 4.3.2, we derive the results for minimizing delay-factor and ℓ_k norms of delay-factor.

4.3.1 Analysis of LF

To bound the competitiveness of LF_s, we use the same techniques we used for bounding the competitiveness of LWF_s. Events are again defined in the same fashion; $E_{p,x}$ is the event defined by the x'th transmission of p by LF_s. We again partition events into self-chargeable and non self-chargeable events

and charge self-chargeable events to the optimal value and charge non-self-chargeable events to $\delta \mathrm{LF}_s + m(\rho)\mathrm{OPT}^N$ for some $\delta < 1$. For an event $E_{p,x}$, let $M_{p,x}(t) = \sum_{J_{p,i} \in \mathcal{J}_{p,x}} m_{p,i}(t - r_{p,i})$ denote the total m-cost of all requests for p that arrive in $[b_{p,x}, e_{p,x} - 1]$ that are satisfied at $e_{p,x}$. We let $M^*_{p,x}(t)$ be the m-cost of the same set of requests for the optimal solution. An event $E_{p,x}$ is self-chargeable if $M_{p,x} \leq m(\rho)M^*_{p,x}$ or $e_{p,x} - b_{p,x} \leq \rho$ for some constant ρ to be optimized later. The remaining events are non self-chargeable. Again, requests for non-self-chargeable events are divided into early requests and late requests based on whether they arrive before $o'_{p,x}$ or not where $o'_{p,x} = \min\{o_{p,x}, e_{p,x} - \rho\}$. Let $M^e_{p,x}$ and $M^l_{p,x}$ be the flow time accumulated for early and late requests of a non-self-chargeable event $E_{p,x}$, respectively. The values of LF^N_s, $\mathrm{LF}^{N^l}_s$, $\mathrm{LF}^{N^e}_s$, and LF^S_s are defined in the same way as LWF^N_s, $\mathrm{LWF}^{N^l}_s$, $\mathrm{LWF}^{N^e}_s$, and LWF^S_s. Likewise for OPT. The following two lemmas are analogues of Lemmas 30 and 31 and follow from definitions.

Lemma 35. $\mathrm{LF}^S_s \leq m(\rho)\mathrm{OPT}^S$.

Lemma 36. $\mathrm{LF}^{N^l}_s \leq m(\rho)\mathrm{OPT}^N$.

We now show a generalization of Lemma 32 that states that any event $E_{q,y}$ such that $e_{q,y}$ is close to $e_{p,x}$ has m-waiting time comparable to the m-waiting time of early requests of $E_{p,x}$.

Lemma 37. *Suppose $E_{p,x}$ and $E_{q,y}$ are two events such that $e_{q,y} \in [\lceil o'_{p,x} + \lambda(e_{p,x} - o'_{p,x})\rceil, e_{p,x} - 1]$, $M_{q,y} \geq h(\lambda)M^e_{p,x}$.*

Proof. Consider an early request $J_{p,i}$ in $\mathcal{J}_{p,x}$ and let $t \in [\lceil o'_{p,x} + \lambda(e_{p,x} - o'_{p,x})\rceil, e_{p,x} - 1]$. Since $r_{p,i} \leq o'_{p,x}$, it follows that $t \geq \lambda(e_{p,x} - r_{p,i}) + r_{p,i}$. Hence, $m_{p,i}(t - r_{p,i}) \geq h(\lambda)m_{p,i}(e_{p,x} - r_{p,i})$. Summing over all early requests, it follows that $M^e_{p,x}(t) \geq h(\lambda)M^e_{p,x}$. Since LF_s chose to transmit q at $t = e_{q,y}$ instead of p, it follows that $M_{q,y} \geq M_{p,x}(e_{q,y}) \geq M^e_{p,x}(e_{q,y}) \geq h(\lambda)M^e_{p,x}$. □

As in Section 4.2.1, the key ingredient of the analysis is to bound the waiting time of early requests. We state the analogue of Lemma 33 below. Observe that

we have an additional parameter β. In Lemma 33 we hard wire β to be $1/2$ to simplify the exposition. In the more general setting, the parameter β needs to be tuned based on h.

Lemma 38. *For any $0 < \beta < 1$, $\mathrm{LF}_s^{N^e} \leq \frac{\rho}{sh(\beta)(\rho(1-\beta)-1)}\mathrm{LF}_s$, where h is some scaling function for m.*

The proof of the above lemma follows essentially the same lines as that of Lemma 33. The idea is to charge $M^e_{p,x}$ to events in the interval $[o'_{p,x} + \lceil\beta(e_{p,x} - o'_{p,x})\rceil, e_{p,x} - 1]$. Using Lemma 37, each event in this interval is within a factor of $h(\lambda)$ of $M^e_{p,x}$. The length of this interval is at least $\frac{\rho(1-\beta)-1}{\rho}$ times the length of the interval $[o'_{p,x}, e_{p,x} - 1]$. To avoid overcharging we again resort to the global scheme using fractional Hall's theorem after we setup the bipartite graph. We can then prove the existence of a $\frac{\rho}{s(\rho(1-\beta)-1)}$-covering and since each event can pay to within a factor of $h(\beta)$, the lemma follows.

Putting the above lemmas together we derive the following theorem.

Theorem 39. *Let $\beta \in (0,1)$ and $\rho > 1$ be given constants. If s is an integer such that $\frac{\rho}{sh(\beta)(\rho(1-\beta)-1)} \leq \delta < 1$, then algorithm LF_s is s-speed $\frac{m(\rho)}{1-\delta}$-competitive.*

4.3.2 Results for Delay-Factor and ℓ_k Norms

We apply Theorem 39 with appropriate choice of parameters to show that LF_s is $O(1)$-competitive with $O(1)$ speed.

For minimizing average delay-factor we have $h(\lambda) = \lambda$ and $m(x) \leq x$. For this reason, average delay-factor behaves essentially the same as average flow-time and we can carry over the results from flow-time.

Theorem 40. *The algorithm* LF *is* 5*-speed $O(1)$ competitive for minimizing the average delay-factor. For non-integer speeds it is $4+\epsilon$-speed $O(1/\epsilon^2)$-competitive.*

For ℓ_k norms of delay-factor we have $h(\lambda) = \lambda^k$ and $m(x) \leq x^k$. By choosing $\beta = \frac{k}{k+1}$, $\rho = 90(k+1)$ and $s = 3(k+1)$ in Theorem 39, we can show that the

algorithm LF is $3(k+1)$-speed $O(\rho^k)$-competitive for minimizing $\sum_{p,i} m_{p,i}(C_{p,i} - r_{p,i})$. Thus for minimizing the L^k-norm delay factor, we obtain $\sqrt[k]{O(\rho^k)} = O(\rho)$ competitiveness, which shows the following.

Theorem 41. *For $k \geq 1$, the algorithm* LF *is $O(k)$-speed $O(k)$-competitive for minimizing ℓ_k-norm of delay-factor.*

4.4 Lower Bound on LF

In this section, we show a lower bound of LF. We note that LF for the maximum delay factor problem is similar to the algorithm SSF-W considered in the previous chapter. To show the lower bound we will consider the algorithm LF in the *standard* scheduling setting when all requests have uniform sizes. Note that we will drop the notation from the broadcast setting and consider the notation for the standard setting. Recall that the non-uniform requests setting can be reduced to the uniform request setting when preemption is permissible.

Notice that LF is the same as SSF-W when $c = 1$. However, we are able to show a negative result on this algorithm for minimizing the maximum delay factor. This demonstrates the importance of the trade-off between scheduling a request with smallest slack and scheduling the request with a large delay factor that SSF-W makes. The rest of this section will be devoted to showing the following theorem.

Theorem 42. *For any constant $s > 1$,* LF *is not constant competitive with s-speed for minimizing the maximum delay factor (or weighted flow time) in the standard scheduling setting when requests have uniform sizes.*

Since LF processes the request J_i such that $\frac{t-r_i}{S_i}$ is maximized, it would be helpful to formally define the quantity. Let us define the wait ratio of J_i at time $t \geq r_i$ as $\frac{t-r_i}{S_i}$; recall that r_i and S_i are the arrival time and slack size of J_i respectively. Note that J_i's wait ratio at time C_i is the same as its delay factor if J_i has delay factor no smaller than 1. Further note that J_i's delay

factor is equal to $\max\{1, \frac{C_i - r_i}{S_i}\}$. The algorithm LF schedules the request with the largest wait ratio at each time. LF can be seen as a natural generalization of FIFO. This is because FIFO schedules the request with largest wait time at each time. Recall that SSF-W makes requests to wait to be scheduled to help merge potential requests in a single broadcast. The algorithm LF behaves similarly since it implicitly delays each request until it is the request with the largest wait ratio, potentially merging many requests into a single broadcast. Hence, this algorithm is a natural candidate for the problem of minimizing the maximum delay factor and it does not need any parameters as the algorithm SSF-W does. We show however that this algorithm does not have a constant competitive ratio with any constant speed.

We construct the following adversarial instance σ for any integral speed-up $s > 1$ and any integer $c \geq 2$; the assumption of s and c being integral can be easily removed by multiplying the parameters in the instance by a sufficiently large constant. For this problem instance we will show that LF has wait ratio at least c, while OPT has wait ratio at most 1. In the instance σ there is a sequence of groups of requests, $\mathcal{J}_i$ for $0 \leq i \leq k$, where k is an integer to be fixed later. We now explain the requests in each group. For simplicity of notation and readability, we will allow requests to arrive at negative times. We can shift all of the arrival times later to make the arrival times positive. All requests have unit sizes. All requests in each group $\mathcal{J}_i$ have the same arrival time $A_i = -(sc)^{k-i+1} - \sum_{j=0}^{k-1-i}(sc)^j$ and have the same slack size $S_i = \frac{s(sc)^{k-i}}{(1-1/sc)^{k-i}}$. For notational simplicity, we override the notation S_i to refer to the slack size of any request in $\mathcal{J}_i$, rather than to refer to the slack size of an individual request J_i. There are $s(sc)^{k+1}$ requests in group $\mathcal{J}_0$ and $s(sc)^{k-i}$ requests in group $\mathcal{J}_i$ for $1 \leq i \leq k$.

We now explain how LF and OPT behave for the instance σ. Sometimes, we will use $\mathcal{J}_i$ to refer to requests in $\mathcal{J}_i$ rather than explicitly saying "requests in $\mathcal{J}_i$", since all requests in the same group are indistinguishable to the scheduler. For the first group $\mathcal{J}_0$, LF keeps processing $\mathcal{J}_0$ upon its arrival until completing it. On the other hand, we let OPT procrastinate $\mathcal{J}_0$ until OPT finishes all requests

in $\mathcal{J}_1$ to $\mathcal{J}_k$. This does not hurt OPT, since the slack size of the requests in $\mathcal{J}_0$ is large relative to other requests. For each group $\mathcal{J}_i$ for $1 \leq i \leq k$, OPT will start $\mathcal{J}_i$ upon its arrival and complete each request in $\mathcal{J}_i$ without interruption. To the contrary, for each $1 \leq i \leq k$, LF will not begin scheduling $\mathcal{J}_i$ until finishing all requests in $\mathcal{J}_{i-1}$. In this way substantial delay is accumulated before LF processes $\mathcal{J}_k$ and such a delay is critical for LF, since the slack of $\mathcal{J}_k$ is small. We refer the reader to Figure 4.2.

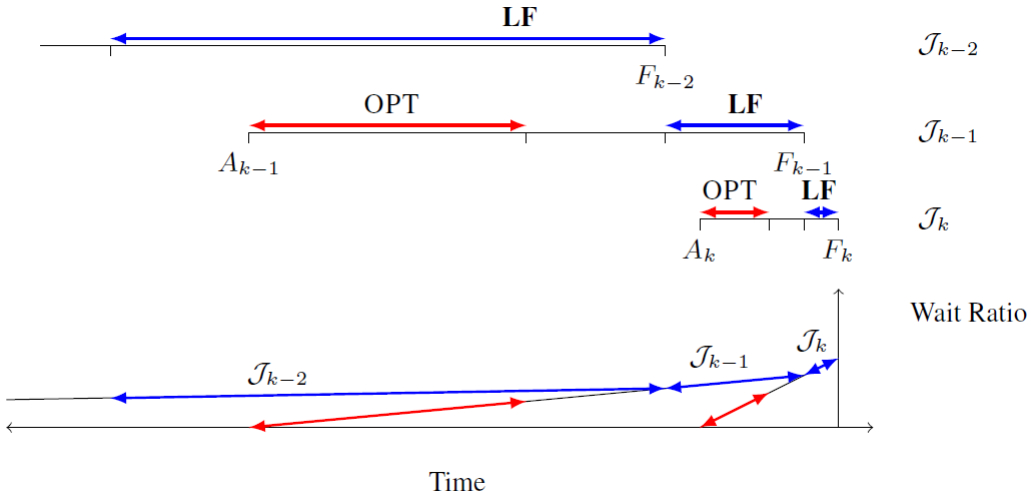


Figure 4.2: Comparison of scheduling of group $\mathcal{J}_k$, $\mathcal{J}_{k-1}$, and $\mathcal{J}_{k-2}$ by LF and OPT.

s We now formally prove that LF has the maximum wait ratio c, while OPT has wait ratio at most 1 for the given problem instance σ. Let $F_i = A_i + (sc)^{k-i+1} = -\sum_{j=0}^{k-1-i}(sc)^j, 0 \leq i \leq k$. Let $r_i(t)$ denote the wait ratio of any request in $\mathcal{J}_i$ at time t. We fix k to be an integer such that $3sc(1-\frac{1}{sc})^k c \leq 1$.

Lemma 43. LF, *given speed s, completely schedules $\mathcal{J}_0$ during $[A_0, F_0]$ and $\mathcal{J}_i$ during $[F_{i-1}, F_i]$, $1 \leq i \leq k$. Further, the maximum wait ratio of any request in $\mathcal{J}_k$ under* LF*'s schedule is c.*

Proof. Observe that the length of the time interval $[A_0, F_0]$ is exactly the amount of time LF with speed s needs to completely process $\mathcal{J}_0$, since $s|[A_0, F_0]| = s(sc)^{k+1} = |\mathcal{J}_0|$. Similarly we observe that the length of the time interval $[F_{i-1}, F_i], 1 \leq i \leq k$ is exactly the amount of time LF with speed s requires to completely process $\mathcal{J}_i$, since $s|[F_{i-1}, F_i]| = s(sc)^{k-i} = |\mathcal{J}_i|$.

First we show that $\mathcal{J}_0$ is finished during $[A_0, F_0]$ by LF. Note that before time F_0, no request in $\mathcal{J}_j$, $j \geq 2$ arrives, since $F_0 = -\sum_{j=0}^{k-1}(sc)^j \leq -(sc)^{k-1} - \sum_{j=0}^{k-3}(sc)^j = A_2$ and all requests in $\mathcal{J}_j$, $j \geq 2$ arrive no earlier than time A_2. We will show that $\mathcal{J}_0$ has the same wait ratio as $\mathcal{J}_1$ at time F_0. Then since $\mathcal{J}_0$ has a slack greater than $\mathcal{J}_1$, at any time t during $[A_0, F_0]$, $r_0(t) > r_1(t)$ and hence LF will work on $\mathcal{J}_0$ over $\mathcal{J}_1$. Indeed, the wait ratio of $\mathcal{J}_0$ at time F_0 is $r_0(F_0) = \frac{F_0 - A_0}{S_0} = (sc)^{k+1} \Big/ \frac{s(sc)^k}{(1-1/sc)^k} = c(1-1/sc)^k$, which is equal to the wait ratio of $\mathcal{J}_1$ at time F_0, $r_1(F_0) = \frac{F_0 - A_1}{S_1} = \Big((sc)^k - (sc)^{k-1}\Big) \Big/ \frac{s(sc)^{k-1}}{(1-1/sc)^{k-1}} = c(1-1/sc)^k$.

To complete the proof, we show that $\mathcal{J}_i$, $i \geq 1$ is finished during $[F_{i-1}, F_i]$ by LF. This proof is very similar to the above. Note that no request in $\mathcal{J}_j$, $j \geq i+2$ arrives before time F_i, since $F_i = -\sum_{j=0}^{k-1-i}(sc)^j \leq -(sc)^{k-i-1} - \sum_{j=0}^{k-3-i}(sc)^j = A_{i+2}$ and all requests in $\mathcal{J}_j$, $j \leq i+2$ arrive no earlier than time A_{i+2}. We will show that $\mathcal{J}_i$ has the same wait ratio as $\mathcal{J}_{i+1}$ at time F_i. Then since the slack of $\mathcal{J}_{i+1}$ is smaller than $\mathcal{J}_i$, at any time t during $[F_{i-1}, F_i]$, $\mathcal{J}_i$ will have wait ratio no smaller than $\mathcal{J}_{i+1}$ and hence LF will work on $\mathcal{J}_i$ over $\mathcal{J}_{i+1}$. Indeed, the wait ratio of $\mathcal{J}_i$ at time F_i is $r_i(F_i) = \frac{F_i - A_i}{S_i} = (sc)^{k-i+1} \Big/ \frac{s(sc)^{k-i}}{(1-1/sc)^{k-i}} = c(1-1/sc)^{k-i}$, which is equal to the current delay factor of $\mathcal{J}_{i+1}$ at time F_i, $r_{i+1}(F_i) = \frac{F_i - A_{i+1}}{S_{i+1}} = \Big((sc)^{k-i} - (sc)^{k-i-1}\Big) \Big/ \frac{s(sc)^{k-1-i}}{(1-1/sc)^{k-i-1}} = c(1-1/sc)^{k-i}$. Hence LF has wait ratio at least c for a certain request in $\mathcal{J}_k$. □

In the following lemma, we show that there exists a feasible scheduling by OPT that has wait ratio at most 1, which together with Lemma 43 will complete the proof of Theorem 42.

Lemma 44. *Consider a schedule which processes all requests in $\mathcal{J}_0$ during $[F_k, F_k + |\mathcal{J}_0|]$ and all requests in $\mathcal{J}_i$ during $[A_i, A_i + |\mathcal{J}_i|]$ for $1 \leq i \leq k$ with speed 1. This schedule is feasible and, moreover, the maximum wait ratio of any request under this schedule is at most one.*

Proof. We first observe that the time intervals $[F_k, F_k + |\mathcal{J}_0|]$ and $[A_i, A_i + |\mathcal{J}_i|]$ for $1 \leq i \leq k$ do not overlap, since for $i \geq 1$, $A_{i+1} - (A_i + |\mathcal{J}_i|) = (sc)^{k+1-i} + (sc)^{k-1-i} - s(sc)^{k-i} - (sc)^{k-i} \geq (sc)^{k-i}(sc - s - 1) > 0$, and $F_k - (A_k + |\mathcal{J}_k|) = sc - s > 0$.

Further, all requests in $\mathcal{J}_0$ can be completed during $[F_k, F_k + |\mathcal{J}_0|]$ by a scheduler with speed 1. Likewise, this shows that all requests in $\mathcal{J}_i$ can be completed during $[A_i, A_i + |\mathcal{J}_i|]$ by a scheduler with speed 1. Hence this is a feasible schedule for a 1 speed algorithm.

It now remains to upper bound the maximum wait ratio of any request under the suggested schedule. Consider any request in $\mathcal{J}_i$, $i \geq 1$. The maximum wait ratio of $\mathcal{J}_i$ under the schedule is $\frac{A_i+|\mathcal{J}_i|-A_i}{S_i} = s(sc)^{k-i} \Big/ \frac{s(sc)^{k-i}}{(1-1/sc)^{k-i}} = (1-1/sc)^{k-i} < 1$. The maximum wait ratio of any request in $\mathcal{J}_0$ is $r_0(F_k + |\mathcal{J}_0|) = \frac{F_k+|\mathcal{J}_0|-A_0}{S_0} = \left((s+1)(sc)^{k+1} + \sum_{j=0}^{k-1}(sc)^j\right) \Big/ \frac{s(sc)^k}{(1-1/sc)^k} \leq 3s(sc)^{k+1} \Big/ \frac{s(sc)^k}{(1-1/sc)^k} \leq 3sc(1-1/sc)^k \leq 1$. The last inequality holds since k was chosen to satisfy the inequality. □

4.5 Conclusion

We gave a simpler analysis of LWF for minimizing average flow-time in broadcast scheduling. This not only helps improve the speed bound but also results in extending the algorithm and analysis to more general objective functions such a delay-factor and ℓ_k norms of delay-factor. Our analysis of LWF has lead to the development of the first scalable algorithm for average flow time in the broadcast model, which is shown in the next chapter.

We also showed LF is not $O(1)$-competitive with any speed for ℓ_∞-norm of delay factor, which is equivalent to minimizing the maximum delay factor. Thus, we believe the speed requirement for LF to obtain $O(1)$-competitiveness needs to grow with k for ℓ_k-norms of delay factor. It would be interesting to formally prove this. After this work, a scalable algorithm was given for the ℓ_k-norms of flow time in the broadcast model. This is presented later in this thesis.

Chapter 5

A Scalable Algorithm for Minimizing Average Flow Time in the Broadcast Model

5.1 Introduction

In the previous chapter we considered the algorithm LWF and showed that it is a 5-speed $O(1)$-competitive algorithm for average flow time in the broadcast setting. This chapter is focused on introducing a scalable algorithm for average flow time in the broadcast setting. Recall that there are strong lower bounds on the competitive ratio of any algorithm for the average flow time problem. Thus this is essentially the best result that one can hope for. See chapter 2 for motivation and related work.

We give the first online scalable algorithm LA-W. We prove that LA-W is $(1+\epsilon)$-speed $O(1/\epsilon^{11})$-competitive for any $0 < \epsilon \leq 1$. This is the first algorithm shown to be scalable in the broadcast setting and this answered a central open problem in the area. Our algorithm LA-W is similar to LWF in that it prioritizes pages with large flow time, however LA-W also gives preference to requests which have arrived recently. Favoring requests which have arrived recently has been shown to be useful in [40]. The algorithm LA-W focuses on *pages* which have requests that arrived recently. This is fundamentally different from the algorithm given in [40], which focuses on requests that arrived recently without considering the page they are requesting. Unfortunately, in the broadcast setting it is difficult to categorize which pages have requests that arrived recently, since the arrival of requests can be scattered over time. To counter this, we develop a novel and robust way to compare the arrival time of requests between two different pages.

Overview of the Algorithm: Let $F_p(t)$ be the total waiting time of unsatisfied requests for page p at time t and let $F_{max}(t) = \max_p F_p(t)$. LWF schedules a page p such that $F_p(t) = F_{max}(t)$. Notice that LWF schedules the page without considering the number of outstanding requests for the page. Due to this, LWF may broadcast a page with a relatively small number of unsatisfied requests which have been waiting to be scheduled for a long period of time. However, a page with a small number of requests does not accumulate flow time quickly. In some cases, pages which have a large number of unsatisfied requests should be broadcasted since these requests will rapidly accumulate flow time. Using this insight, [39] was able to show a lower bound of 1.618 on the speed LWF required to be $O(1)$-competitive.

Our algorithm LA-W keeps the main spirit of LWF by always broadcasting pages with flow time comparable to $F_{max}(t)$ at each time t. However, amongst the pages with flow time comparable to $F_{max}(t)$, LA-W prioritizes pages with requests which have arrived recently. By prioritizing recent requests, we avoid the potentially negative behavior of LWF. This is because a page with requests that arrived recently must have a large number of outstanding requests to have flow time similar to F_{max}. As mentioned, we develop a new way to compare the arrive time of requests for two different pages. Using this technique, we will be able to break up time into intervals and show when requests arrive on these intervals, thus allowing us to determine how LA-W and the optimal solution must behave on these intervals.

The algorithm LA-W broadcasts pages with unsatisfied requests that arrived recently to potentially find pages which have a large number of outstanding requests. The reader may wonder why we chose pages in this manner when we could simply broadcast the page with many outstanding requests. In fact, we have considered an algorithm which schedules the page with the largest number of outstanding requests amongst the pages with flow time comparable to $F_{max}(t)$. For this algorithm, we have established that it is scalable for the problem of minimizing the *maximum weighted* flow time in broadcast scheduling [31], see chapter

3. Further, we have preliminary evidence that this algorithm is $O(1)$ competitive for average flow time when given more than 2 speed. We however were unable to determine its performance for average flow time when given less than 2 speed.

5.2 Time Model and Algorithm

We assume that all requests have unit processing time and without loss of generality this is 1. In our time model we assume that requests arrive only at non-negative integer times. Any scheduling algorithm A with speed $s \geq 1$ schedules a page every $1/s$ time-steps starting from time 0. When A broadcasts a page p at time t, all alive (unsatisfied) requests for page p which arrived *strictly* earlier than t are *immediately* satisfied by the broadcast. If $J_{p,i}$ is a request satisfied by a broadcast, it has flow time $t - r_{p,i}$. Note that under the schedule produced by the optimal solution with 1-speed, every request has flow time at least 1. On the other hand, A with speed $s > 1$ may finish some requests within a delay less than one. Though it would seem fair to force A to schedule requests after at least one time step, we do not assume this because our analysis will hold in either case and this assumption improves the readability of the analysis.

Before introducing our algorithm, we state notation that will be used throughout the chapter. For any time interval starting at b and ending at e, we let $|I| = e - b$. For a set of requests R, we will let $F(R)$ be the flow time accumulated for the requests in R by our algorithm. For a page p we will let $F_p(t)$ be the total flow time accumulated at time t for unsatisfied requests for page p. We will let $F(R, t)$ be the total flow time accumulated by our algorithm for requests in the set R at time t. Note that if some requests in R arrive after time t then these requests do not contribute to the value of $F(R, t)$. We let $F^*(R)$ denote the total flow time OPT accumulates for a set of requests R.

We now introduce our algorithm, denoted by LA-W for Latest Arrival time with Waiting. We assume that LA-W is given $s = 1 + \epsilon$ speed where $0 < \epsilon \leq 1$ is a fixed constant. Our algorithm is parameterized by constants $c > 1$ and $\beta < 1$

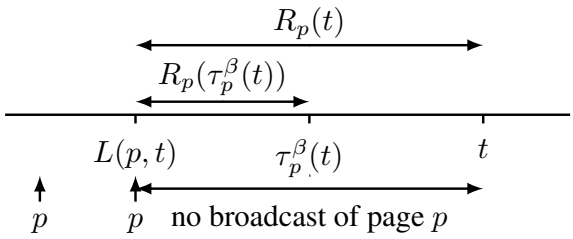


Figure 5.1: $R_p(t)$ denotes the alive requests of page p at time t, i.e. the requests of page p which arrived during $[L(p,t),t]$. Likewise, $R_p(\tau_p^\beta(t))$ denotes the requests which arrived during $[L(p,t),\tau_p^\beta(t)]$.

depending on ϵ, later we will define $\beta = \left(\frac{\epsilon}{1000}\right)^4$ and $c = \left(\frac{10000}{\epsilon^3}\right)$. For each page p and time t, let $R_p(t)$ denote the set of alive requests for page p at time t. Let $L(p,t)$ be the last time before time t that our algorithm broadcasted page p. If there is no such time then $L(p,t) = 0$. Note that $R_p(t)$ is equivalent to the set of the requests for page p which arrived during $[L(p,t),t]$. For a page p and time t let $\tau_p^\beta(t) = \texttt{argmin}_{L(p,t)\le t'\le t}(F(R_p(t'),t) \ge (1-\beta)F_p(t))$. In other words, $\tau_p^\beta(t)$ denotes the *earliest* time t' no later than time t and no earlier than time $L(p,t)$ such that the requests in $R_p(t')$ have total flow time at least $(1-\beta)F_p(t)$ at time t. By this definition, if $R_{[L(p,t),\tau_p^\beta(t)]}$ is the set of requests for page p that arrive on the interval $[L(p,t),\tau_p^\beta(t)]$ and $R_{[L(p,t),\tau_p^\beta(t))}$ is the set of requests for page p that arrive on the interval $[L(p,t),\tau_p^\beta(t))$ then $F(R_{[L(p,t),\tau_p^\beta(t)]},t) \ge (1-\beta)F_p(t)$ and $F(R_{[L(p,t),\tau_p^\beta(t))},t) < (1-\beta)F_p(t)$. See Figure 5.1.

Algorithm: LA-W

Let t be a time where our algorithm is not broadcasting a page.

Let $F_{max}(t) = \max_p F_p(t)$.

Broadcast one page according to Rule 2 every $\lfloor \frac{10}{\epsilon} \rfloor$ broadcasts, and broadcast one page according to Rule 1 otherwise.

`Rule 1`: broadcast the page $p = \texttt{argmax}_{p' \in Q(t)} \tau_{p'}^{\beta}(t)$,
where $Q(t) = \{q \mid F_q(t) \geq \frac{1}{c} F_{max}(t)\}$ breaking ties arbitrarily.

`Rule 2`: broadcast a page p where $F_p(t) = F_{max}(t)$ breaking ties arbitrarily.

Our algorithm LA-W broadcasts pages mainly according to `Rule 1` while occasionally broadcasting a page according to `Rule 2`. The second rule uses LWF's scheduling policy which broadcasts a page with the highest flow time. The first rule chooses a page p with the latest time $\tau_p^{\beta}(t)$ among the pages with flow time close to $F_{max}(t)$. The value of $\tau_p^{\beta}(t)$ can be interpreted as the latest arrival time of any unsatisfied request for page p after discounting requests that arrived recently that have small flow time. Since the arrival of requests for the same page p can be scattered over time, we will use $\tau_p^{\beta}(t)$ as the representative arrival time of those requests. Notice that if all requests for page p arrive at time t' then $\tau_p^{\beta}(t) = t'$ for any $0 < \beta \leq 1$. We remark that we do not know if `Rule 2` is needed for LA-W to be $(1+\epsilon)$-speed $O(1)$-competitive. `Rule 2` will play a crucial role in our analysis, but we do not have a proof that `Rule 1` alone performs poorly.

5.3 Analysis

Let σ be a fixed sequence of requests. OPT denotes a fixed offline optimal solution. We assume LA-W$_{1+\epsilon}$ is always busy scheduling pages for the sequence

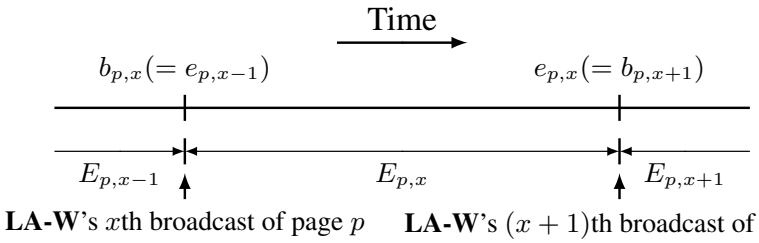


Figure 5.2: Events for page p.

σ. Otherwise, our arguments can be applied to each maximal time interval where LA-W$_{1+\epsilon}$ is busy. Following the lead of [39, 30], time is partitioned into *events* for each page p. Events for page p are defined by LA-W$_{1+\epsilon}$'s broadcasts of page p. Each time LA-W$_{1+\epsilon}$ broadcasts a page, an event begins and an event ends. An event $E_{p,x} = \langle b_{p,x}, e_{p,x} \rangle$ begins at time $b_{p,x}$ and ends at time $e_{p,x}$. Here, LA-W$_{1+\epsilon}$ broadcasts page p at time $b_{p,x}$ and at time $e_{p,x}$. These are the xth and $(x+1)$st broadcasts of page p by LA-W$_{1+\epsilon}$. The $(x+1)$st broadcast of page p starts a new event $E_{p,x+1}$ and $e_{p,x} = b_{p,x+1}$. On the time interval $(b_{p,x}, e_{p,x})$ LA-W$_{1+\epsilon}$ does not broadcast page p. The optimal solution can broadcast page p zero or more times during an event $E_{p,x}$. See Figure 5.2.

For an event $E_{p,x}$, let $R_{p,x}$ denote the set of requests satisfied by the $(x+1)$st broadcast of page p. Notice that all requests in $R_{p,x}$ arrive during $E_{p,x}$, formally during $[b_{p,x}, e_{p,x})$. Let $F_{p,x} = F(R_{p,x})$ be the total flow time LA-W$_{1+\epsilon}$ accumulates for requests in $R_{p,x}$. Likewise let $F^*_{p,x} = F^*(R_{p,x})$ be the flow time OPT accumulates for requests in $R_{p,x}$. We refer to $F_{p,x}$ as the flow time of $E_{p,x}$. Similarly to requests, for a set $\mathcal{E}$ of events we let $F(\mathcal{E}) = \sum_{E_{p,x} \in \mathcal{E}} F_{p,x}$.

Our goal is to show that $\sum_p \sum_x F_{p,x} \leq O(1)\text{OPT}$. We start by partitioning events into two groups. An event $E_{p,x}$ is called *self-chargeable* if $F_{p,x} \leq \gamma F^*_{p,x}$ where $\gamma \geq 1$ is a constant that will be fixed as $\gamma = \frac{10000}{\epsilon^2 \beta}$ later. Let $\mathcal{S}$ be the set of all self-chargeable events. The other events are called *non-self-chargeable* and are in the set $\mathcal{N}$. By definition of self-chargeable events, we can easily bound $F(\mathcal{S})$ by OPT.

Lemma 45. $F(\mathcal{S}) \leq \gamma \text{OPT}$.

Proof. $F(\mathcal{S}) = \sum_{E_{p,x} \in \mathcal{S}} F_{p,x} \leq \sum_{E_{p,x} \in \mathcal{S}} \gamma F^*_{p,x} \leq \gamma \text{OPT}$. □

We now concentrate on non-self-chargeable events. Notice that for a non-self-chargeable event $E_{p,x}$, the optimal solution must broadcast page p during $E_{p,x}$, formally during $(b_{p,x}, e_{p,x})$. Otherwise, $F^*_{p,x} \geq F_{p,x}$ and the event is self-chargeable. We further partition non-self-chargeable events into two classes. Consider a non-self-chargeable event $E_{p,x}$. Let α and k be constants such that $\alpha < 1$, $k > 1$ and $\beta k < 1$. We will fix $\alpha = \frac{\epsilon}{100}$ and $k = \frac{10}{\epsilon}(\lceil 1000c \rceil + 2) + 1$ later. $E_{p,x}$ is in the set $\mathcal{N}_1$ if for some $\beta \leq \rho \leq \beta k$ it is the case that at least $\lceil \alpha s(e_{p,x} - \tau^\rho_p(e_{p,x})) \rceil$ self-chargeable events end on the interval $[\tau^\rho_p(e_{p,x}), e_{p,x})$. Notice that the time $\tau^\rho_p(e_{p,x})$ exists because $\rho < 1$. A non-self-chargeable event not in $\mathcal{N}_1$ is in $\mathcal{N}_2$. The sets $\mathcal{N}_1$ and $\mathcal{N}_2$ are similar to how [30] partitions non-self-chargeable events.

The events in $\mathcal{N}_1$ can easily be bounded by OPT. We do this by bounding $F(\mathcal{N}_1)$ by the flow time of the self chargeable events ending during the events in $\mathcal{N}_1$. Knowing that $F(\mathcal{S}) \leq \gamma \text{OPT}$ we will be able to bound $F(\mathcal{N}_1)$ by OPT. We will formally show $F(\mathcal{N}_1) \leq O(\frac{1}{\epsilon^{11}})\text{OPT}$ later in Lemma 51.

The most interesting events are those which are in $\mathcal{N}_2$. Since each event $E_{p,x}$ in $\mathcal{N}_2$ has a relatively small number of self-chargeable events ending during $E_{p,x}$, we cannot directly bound $F(\mathcal{N}_2)$ by OPT. Instead, we will show that the total flow time of events in $\mathcal{N}_2$ accounts for only a fraction of LA-W$_{1+\epsilon}$'s total flow time, i.e. $F(\mathcal{N}_2) \leq \delta \text{LA-W}_{1+\epsilon}$ for some constant $\delta < 1$ which is independent of ϵ. In [31] and [39] speed greater than 3.4 was needed to bound $F(\mathcal{N}_2)$. Our goal is to ensure $\delta < 1$ with only $(1+\epsilon)$ speed. Showing this will complete our analysis as follows. Using this, Lemma 45 and Lemma 51, we have that $\text{LA-W}_{1+\epsilon} = F(\mathcal{S}) + F(\mathcal{N}) = F(\mathcal{S}) + F(\mathcal{N}_1) + F(\mathcal{N}_2) \leq \gamma \text{OPT} + O(\frac{1}{\epsilon^{11}})\text{OPT} + \delta \text{LA-W}_{1+\epsilon}$, which simplifies to $\text{LA-W}_{1+\epsilon} \leq \frac{\gamma + O(\frac{1}{\epsilon^{11}})}{1-\delta}\text{OPT}$. This will imply the following theorem.

Theorem 46. *For $0 < \epsilon \leq 1$, the algorithm* LA-W *is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^{11}})$-competitive for minimizing average flow time in broadcast scheduling with unit sized pages.*

Before continuing, we show some properties of events in $\mathcal{N}$. Say that we set $\gamma \geq \frac{1}{\beta}$. Then it is not hard to show that OPT must broadcast page p during $I = [\tau_p^\beta(e_{p,x}), e_{p,x})$ for any non-self-chargeable event $E_{p,x}$. Indeed, the requests for page p that arrive during the interval I have total flow time at least $\beta F_{p,x}$ in LA-W$_{1+\epsilon}$'s schedule by definition of τ^β. If OPT does not broadcast page p during I this implies that these requests also have total flow time $\beta F_{p,x}$ in OPT's schedule. However, then $F^*_{p,x} \geq \beta F_{p,x} \geq \frac{1}{\gamma} F_{p,x}$, contradicting the fact that $E_{p,x}$ is non-sef-chargeable.

Lemma 47. *Suppose that $\gamma \geq \frac{1}{\beta}$. Then, for any non-self-chargeable event $E_{p,x}$, the optimal solution must broadcast page p during the interval $[\tau_p^\beta(e_{p,x}), e_{p,x})$.*

Proof. For the sake of contradiction assume the lemma is false. The event $E_{p,x}$ is non-self-chargeable therefore the optimal solution must broadcast page p at some time during $(b_{p,x}, \tau_p^\beta(e_{p,x}))$. Let t be the latest broadcasting time of page p by the optimal solution during $(b_{p,x}, \tau_p^\beta(e_{p,x}))$. Let $S_{[b_{p,x},t]}$ and $S_{(t,e_{p,x})}$ be the set of requests for page p which arrive during $[b_{p,x}, t]$ and $(t, e_{p,x})$, respectively. We know that $F(S_{[b_{p,x},t]}) < (1-\beta)F_{p,x}$ by definition of $\tau_p^\beta(e_{p,x})$ and $t < \tau_p^\beta(e_{p,x})$. Thus $F(S_{(t,e_{p,x})}) = F(R_{p,x} \setminus S_{[b_{p,x},t]}) > \beta F_{p,x}$. Since the optimal solution does not broadcast page p during $(t, e_{p,x})$, it follows that $F^*_{p,x} \geq F^*(S_{(t,e_{p,x})}) > \beta F_{p,x} \geq \frac{1}{\gamma} F_{p,x}$, which is a contradiction to $E_{p,x}$ being a non-self-chargeable event. □

Now say that we set $\gamma \geq \frac{10000}{\beta\epsilon^2}$. Using similar ideas as in Lemma 47, we will be able to show that $|[\tau_p^\beta(e_{p,x}), e_{p,x})| \geq \frac{10000}{\epsilon^2}$. This will be used to ensure that the intervals considered in our remaining arguments are sufficiently long.

Lemma 48. *Suppose $\gamma \geq \frac{10000}{\epsilon^2\beta}$. Then, for any non-self-chargeable event $E_{p,x}$, $|[\tau_p^\beta(e_{p,x}), e_{p,x}]| \geq \frac{10000}{\epsilon^2}$.*

Proof. For the sake of contradiction, assume that there exists a non-self-chargeable event $E_{p,x}$ such that $|[\tau_p^\beta(e_{p,x}), e_{p,x}]| < \frac{10000}{\epsilon^2}$. Let S be the set of requests for page p which arrive on the interval $[\tau_p^\beta(e_{p,x}), e_{p,x})$. By definition of $\tau_p^\beta(e_{p,x})$ it must be the case that $F(S) > \beta F_{p,x}$. We now want to bound the number of requests in S.

Since each request in S can accumulate flow time at most $|[\tau_p^\beta(e_{p,x}), e_{p,x}]| < \frac{10000}{\epsilon^2}$, we have that $F(S) < |S|\frac{10000}{\epsilon^2}$, thus $\beta F_{p,x} < |S|\frac{10000}{\epsilon^2}$. Hence we have that $|S| > \frac{\epsilon^2}{10000}\beta F_{p,x}$. The optimal solution must accumulate at least $|S|$ flow time for the requests in S, therefore $F^*_{p,x} \geq |S| > \frac{\epsilon^2}{10000}\beta F_{p,x} \geq \frac{1}{\gamma}F_{p,x}$. This is a contradiction to $E_{p,x}$ being non-self-chargeable. □

To bound $F(\mathcal{N}_1)$ and $F(\mathcal{N}_2)$, we need the following two lemmas. For any event $E_{p,x}$, the first lemma will be used to bound the flow time accumulated for page p at different times during $E_{p,x}$. This will help us to compare the flow time of $E_{p,x}$ to the flow time of events ending during $E_{p,x}$. The proof of this lemma follows easily by definition of flow time.

Lemma 49. *For any event $E_{p,x}$, let $R' \subseteq R_{p,x}$. Let t be such that $b_{p,x} \leq t < e_{p,x}$. Suppose that all requests in R' arrive no later than time t. Then for any $0 \leq \eta < 1$, $F(R', t + \eta(e_{p,x} - t)) \geq \eta F(R')$. Further, if $F(R') \geq \upsilon F_{p,x}$, then $F(R', t + \eta(e_{p,x} - t)) \geq \eta\upsilon F_{p,x}$.*

Proof. $F(R', t+\eta(e_{p,x}-t)) = \sum_{J_{p,i}\in R'}(t+\eta(e_{p,x}-t)-r_{p,i}) = \sum_{J_{p,i}\in R'}((1-\eta)t+\eta e_{p,x}-a_{p,i}) \geq \sum_{J_{p,i}\in R'}((1-\eta)r_{p,i}+\eta e_{p,x}-r_{p,i}) = \eta\sum_{J_{p,i}\in R'}(e_{p,x}-r_{p,i}) = \eta F(R')$. The inequality holds, since any request $J_{p,i}$ in R' arrives no later than time t. □

The next lemma gives a global charging scheme built on Hall's theorem, which is a generalization of the techniques used in [39, 30].This lemma shows how to charge the flow time of some events to the total flow time LA-W$_{1+\epsilon}$ accumulates. The proof is technical and we defer its proof to Section 5.3.4.

Lemma 50. *Let $\mathcal{A}$ be a set of events. Let $\mu, \kappa > 0$ be some constants. Let $\lambda \geq 1$ be an integer. For each event $E_{p,x} \in \mathcal{A}$, suppose there exists an interval $I_{p,x}$ and a set of events $\mathcal{B}_{p,x}$ such that*

The optimal solution broadcasts page p at least λ times during the interval $I_{p,x}$. Further, $I_{p,x}$ is disjoint with $I_{p,x'}$ for any $E_{p,x'} \in \mathcal{A}$ s.t. $x' \neq x$.

$|\mathcal{B}_{p,x}| \geq \mu|I_{p,x}|$ *and* $E_{q,y} \in \mathcal{B}_{p,x}$ *only if* $e_{q,y} \in I_{p,x}$ and $F_{q,y} \geq \kappa F_{p,x}$. *Let* $\mathcal{B} = \bigcup_{(p,x):E_{p,x}\in\mathcal{A}} \mathcal{B}_{p,x}$ *and* $d = \min_{E_{p,x}\in\mathcal{A}} |I_{p,x}|$. *Then,* $F(\mathcal{A}) \leq (\frac{2}{\lambda\kappa\mu})(\frac{d+1}{d})F(\mathcal{B}) \leq (\frac{2}{\lambda\kappa\mu})(\frac{d+1}{d})\text{LA-W}_{1+\epsilon}$.

This lemma can be interpreted as follows. For a set of events $\mathcal{A} \subseteq \mathcal{N}_2$, we charge the flow time of each event $E_{p,x} \in \mathcal{A}$ to some events ending during $I_{p,x}$. In our analysis, $I_{p,x}$ will always be a subinterval of $E_{p,x}$; thus for any fixed page p, $\{I_{p,x} \mid E_{p,x} \in \mathcal{A}\}$ are disjoint. If the following conditions hold for each event $E_{p,x} \in \mathcal{A}$, then $F(\mathcal{A}) \ll$ [1]$\text{LA-W}_{1+\epsilon}$. (1) There are at least λ broadcasts by OPT of page p during $I_{p,x}$. (2) We can find a sufficiently large fraction of events ending during $I_{p,x}$, denoted by μ, such that each of these events have flow time at least $\kappa F_{p,x}$. (3) $I_{p,x}$ is sufficiently long for all $E_{p,x} \in \mathcal{A}$. The bound we get on $F(\mathcal{A})$ improves by either finding many broadcasts of page p by OPT during $I_{p,x}$ or by finding sufficiently many events with very large flow time ending during $I_{p,x}$.

Using the global charging scheme given in Lemma 50, we can bound the flow time of events in $\mathcal{N}_1$ by OPT. This is not too difficult and follows easily by combining the analysis given in [30], the definition of τ and Lemma 50. The proof is similar to that given in [30].

Lemma 51. $F(\mathcal{N}_1) \leq O(\frac{1}{\epsilon^{11}})\text{OPT}$.

Proof. We apply Lemma 50 using the notation given in the lemma. Let $\mathcal{A}$ be the set of all $\mathcal{N}_1$ events. Consider any event $E_{p,x} \in \mathcal{A}$. Let $I_{p,x} = [\tau_p^\rho(e_{p,x}), e_{p,x})$ for some fixed $\beta \leq \rho \leq \beta(\frac{10}{\epsilon}(\lceil 1000c \rceil + 2) + 1)$ such that at least $\lceil \alpha s(e_{p,x} - \tau^\rho(e_{p,x})) \rceil$ self-chargeable events end on $I_{p,x}$. Note that ρ exists by definition of $\mathcal{N}_1$ events. By Lemma 47, the optimal solution must broadcast page p during $I_{p,x}$. Due to this we set $\lambda = 1$. Since $|I_{p,x}| \geq \frac{10000}{\epsilon^2}$ by Lemma 48, we have $d = \min_{E_{p,x}\in\mathcal{A}} |I_{p,x}| \geq \frac{10000}{\epsilon^2}$.

Let $\mathcal{B}_{p,x}$ be the self-chargeable events ending during $I'_{p,x} = [\tau_p^\rho(e_{p,x}) + \frac{\alpha}{2}(e_{p,x} - \tau_p^\rho(e_{p,x})), e_{p,x})$. Note that there are at most $\lceil \frac{\alpha s}{2}|I_{p,x}| \rceil$ events ending during $I_{p,x} \setminus I'_{p,x}$, because the algorithm broadcasts a page every $\frac{1}{s}$ time steps during $I_{p,x} \setminus I'_{p,x}$.

[1] $A \ll B$ should read as $A < \xi B$ for some constant $\xi < 1$.

Therefore there exist at least $\lceil \alpha s |I_{p,x}| \rceil - \lceil \frac{\alpha s}{2} |I_{p,x}| \rceil \geq \lfloor \frac{\alpha s}{2} |I_{p,x}| \rfloor \geq \frac{\alpha s}{4} |I_{p,x}|$ self-chargeable events ending during $I'_{p,x}$. Hence, $|\mathcal{B}_{p,x}| \geq \frac{\alpha s}{4} |I_{p,x}|$ and we can set $\mu = \frac{\alpha s}{4}$.

Let $E_{q,y} \in \mathcal{B}_{p,x}$. By Lemma 49 and the definition of $\tau_p^{\rho}(e_{p,x})$ we know that at anytime $t \in I'_{p,x}$ it is the case that $F_{p,x}(t) \geq \frac{\alpha}{2}(1-\rho)F_{p,x}$. Since our algorithm chose to broadcast page q at time $e_{p,x} \in I'_{p,x}$ over page p, we have $F_{q,y} \geq \frac{\alpha}{2c}(1-\rho)F_{p,x}$. Therefore we can set $\kappa = \frac{\alpha}{2c}(1-\rho)$.

In sum, by Lemma 50,

$$F(\mathcal{N}_1) \leq \frac{2}{\lambda \kappa \mu} \frac{d+1}{d} F(\mathcal{S}) = (\frac{16c}{\alpha^2 s})(\frac{1}{1-\rho}) \frac{d+1}{d} (\gamma \mathrm{OPT}) = O(\frac{1}{\epsilon^{11}})\mathrm{OPT}.$$

□

We now focus on bounding the flow time of events in $\mathcal{N}_2$. To exploit Lemma 50, $\mathcal{N}_2$ is partitioned into three disjoint sets $\mathbb{T}_1$, $\mathbb{T}_2$ and $\mathbb{T}_3$. To discuss the high level interpretation of the sets $\mathbb{T}_1$, $\mathbb{T}_2$ and $\mathbb{T}_3$ we fix an event $E_{p,x} \in \mathcal{N}_2$ and page p and drop the subscript p, x. For the event E we will consider different subintervals of E defined by τ. Let $I^i = [\tau^{\beta(\frac{10}{\epsilon} i+1)}(e), e)$ for $i \in \mathbb{N}$. Notice that I^i is a subinterval of I^{i+1} for all i. We will concentrate on the intervals I^i for different values of i. Concentrating on these intervals will allow us to break up the event E so that we can better understand when the requests for page p arrived during E and how the optimal solution and LA-W$_{1+\epsilon}$ behaved during E.

The event E will be in the set $\mathbb{T}_1$ if for some i it is the case that page p is not in the queue Q for a sufficiently large number of broadcasts by LA-W$_{1+\epsilon}$ during I^i. By definition of Q, if p is not in $Q(t)$ then there exists another page q such that $F_q(t) > cF_p(t)$. `Rule 2` of LA-W broadcasts a page with the highest flow time every $\lfloor \frac{10}{\epsilon} \rfloor$ broadcasts. Using this, we will be able to find sufficiently many events ending during E with flow time much larger than the flow time of event E. Then Lemma 50 can be used to show that $F(\mathbb{T}_1) \ll$ LA-W$_{1+\epsilon}$. Intuitively, the requests in $\mathbb{T}_1$ cannot account for most of LA-W$_{1+\epsilon}$'s flow time since there exists other events with flow time much larger than those in $\mathbb{T}_1$.

If the event E is not in the set $\mathbb{T}_1$ and if the length of I^{i+1} is sufficiently longer than the length of I^i for many different values of i then the event E will be in the set $\mathbb{T}_2$. For such an event E, the requests for page p that arrive during E will be grouped according to when they arrived. We will show that each of these groups contributes to a substantial amount of event E's flow time. Knowing that E is non-self-chargeable, we will show that OPT must perform a unique broadcast of page p for each of these groups during E. This allows us to show that $F(\mathbb{T}_2) \ll \text{LA-W}_{1+\epsilon}$ using Lemma 50. Intuitively, since the optimal solution has to perform a lot of broadcasts for each event in $\mathbb{T}_2$, there cannot be many events in $\mathbb{T}_2$. Therefore the events in $\mathbb{T}_2$ do not account for a large portion of LA-$\text{W}_{1+\epsilon}$'s flow time.

Finally $\mathbb{T}_3$ will consist of all events in $\mathcal{N}_2$ that are not in $\mathbb{T}_1$ or $\mathbb{T}_2$. Using the definitions of $\mathbb{T}_1$, $\mathbb{T}_2$ and τ we will be able to show that no events can be in $\mathbb{T}_3$ and this will complete our analysis. Showing that $\mathbb{T}_3 = \emptyset$ is the most difficult part of the analysis and this is where `Rule 1` and resource augmentation plays a crucial role. We now formally define the sets $\mathbb{T}_1$, $\mathbb{T}_2$ and $\mathbb{T}_3$. For simplicity of notation, let $\tau_{p,x}^{\beta,i} = \tau_p^{\beta(\frac{10}{\epsilon}i+1)}(e_{p,x})$. A $\mathcal{N}_2$ event $E_{p,x}$ is in

- $\mathbb{T}_1$ if and only if for some $0 \leq i \leq \lceil 1000c \rceil + 2$ the page p is not in Q for at least $\lceil \frac{\epsilon s}{10} |[\tau_{p,x}^{\beta,i}, e_{p,x})| \rceil$ broadcasts by our algorithm on the interval $[\tau_{p,x}^{\beta,i}, e_{p,x})$.
- $\mathbb{T}_2$ if and only if $E_{p,x} \notin \mathbb{T}_1$ and for all $0 \leq i \leq \lceil 1000c \rceil$, $\tau_{p,x}^{\beta,i} - \tau_{p,x}^{\beta,i+1} \geq \frac{\epsilon}{10}(e_{p,x} - \tau_{p,x}^{\beta,i})$
- $\mathbb{T}_3$ otherwise.

We note that if β and c are chosen such that $\beta(\frac{10}{\epsilon}(\lceil 1000c \rceil + 2) + 1) < 1$, then the time $\tau_{p,x}^{\beta,i}$ must exist for all $0 \leq i \leq \lceil 1000c \rceil + 2$. The rest of the chapter is organized as follows. In Section 5.3.1 we will show that $F(\mathbb{T}_1) \ll \text{LA-W}_{1+\epsilon}$. Then in Section 5.3.2 we will show that $F(\mathbb{T}_2) \ll \text{LA-W}_{1+\epsilon}$. Finally we will

show that $\mathbb{T}_3 = \emptyset$ in Section 5.3.3. Before continuing, we fix our constants, so that our arguments can be verified. As already mentioned, we let $\beta = (\frac{\epsilon}{1000})^4$, $c = \frac{10000}{\epsilon^3}$, $\gamma = \frac{10000}{\epsilon^2\beta}$, $\alpha = \frac{\epsilon}{100}$ and $k = \frac{10}{\epsilon}(\lceil 1000c \rceil + 2) + 1$. Note that $\tau_{p,x}^{\beta,\lceil 1000c \rceil + 2} = \tau_p^{\beta k}(e_{p,x})$ for any page p by definition of k and $\tau_{p,x}^{\beta,i}$. Recall that our algorithm is parameterized by β and c. Here we have chosen c and β so that the analysis is readable and easy to verify and not to optimize the analysis.

5.3.1 Bounding $\mathbb{T}_1$ events.

In this section we bound $F(\mathbb{T}_1)$. By definition of $\mathbb{T}_1$, for each event $E_{p,x} \in \mathbb{T}_1$ the page p is not in Q for at least $\lceil \frac{\epsilon s}{10} |[t_{p,x}, e_{p,x})| \rceil$ broadcasts by LA-W$_{1+\epsilon}$ on the interval $[t_{p,x}, e_{p,x})$ where $t_{p,x} = \tau_{p,x}^{\beta,i}$ for some fixed $0 \leq i \leq \lceil 1000c \rceil + 2$. Recall that our goal is to show that there are many events ending during $E_{p,x}$ with flow time much larger than $F_{p,x}$. After finding these events, we will charge $F_{p,x}$ to these events. We begin by actually finding such events in the next lemma.

Lemma 52. *For an event $E_{p,x} \in \mathbb{T}_1$ there exist at least $(\frac{\epsilon^2 s}{205})|[t_{p,x}, e_{p,x})|$ events ending on the interval $[t_{p,x}, e_{p,x})$ with flow time at least $\frac{c\epsilon}{20}(1 - \beta k)F_{p,x}$.*

Proof. Let $S_{[b_{p,x}, t_{p,x}]}$ be the requests for page p which arrive during $[b_{p,x}, t_{p,x}]$. By the definition of $t_{p,x}$ and τ, we have $F(S_{[b_{p,x}, t_{p,x}]}) \geq (1 - \beta(\frac{10}{\epsilon}i + 1))F_{p,x} \geq (1 - \beta k)F_{p,x}$. Let $I = [t_{p,x} + \frac{\epsilon}{20}(e_{p,x} - t_{p,x}), e_{p,x})$. For any time $t \in I$, by Lemma 49,

$$F(S_{[b_{p,x}, t_{p,x}]}, t) \geq \frac{\epsilon}{20}(1 - \beta k)F_{p,x}. \tag{5.1}$$

By definition of $\mathbb{T}_1$, there are at least $\lceil \frac{\epsilon s}{10}(e_{p,x} - t_{p,x}) \rceil$ broadcasts by our algorithm on the interval $[t_{p,x}, e_{p,x})$ where page p is not in Q. At most $\lceil \frac{\epsilon s}{20}(e_{p,x} - t_{p,x}) \rceil$ of these broadcasts end on the interval $[t_{p,x}, t_{p,x} + \frac{\epsilon}{20}(e_{p,x} - t_{p,x}))$. Therefore, there are at least $\lceil \frac{\epsilon s}{10}(e_{p,x} - t_{p,x}) \rceil - \lceil \frac{\epsilon s}{20}(e_{p,x} - t_{p,x}) \rceil \geq \lfloor \frac{\epsilon s}{20}(e_{p,x} - t_{p,x}) \rfloor$ broadcasts by our algorithm on the interval I where page p is not in Q when these broadcasts were scheduled.

Now consider a time $t \in I$ where page p is not in $Q(t)$. By definition of Q, at time t there must exist some page q such that $F_q(t) \geq cF_p(t)$. Our algorithm schedules the page with the largest flow time every $\lfloor \frac{10}{\epsilon} \rfloor$ broadcasts according to `Rule 2`. Therefore, within $\lfloor \frac{10}{\epsilon} \rfloor$ broadcasts some page q where $F_q(t) \geq cF_p(t)$ is broadcasted by the algorithm. Using this and (5.1), there exists an event $E_{q,y}$ with flow time at least $F_{q,y} > \frac{c\epsilon}{20}(1-\beta k)F_{p,x}$ such that $e_{q,y} \in [t, t + \frac{1}{s}\lfloor \frac{10}{\epsilon} \rfloor)$. Using Lemma 48 to ensure the interval $[t_{p,x}, e_{p,x})$ is sufficiently long, we conclude that there exist at least $\lfloor (\lfloor \frac{\epsilon s}{20}(e_{p,x} - t_{p,x}) \rfloor / \lfloor \frac{10}{\epsilon} \rfloor) \rfloor \geq (\frac{\epsilon^2 s}{205})|[t_{p,x}, e_{p,x})|$ events ending during I with flow time at least $\frac{c\epsilon}{20}(1-\beta k)F_{p,x}$. □

We can now easily bound $F(\mathbb{T}_1)$ by LA-W$_{1+\epsilon}$ using lemmas 50, 52, 47 and 48.

Lemma 53. $F(\mathbb{T}_1) < \frac{83}{100}$LA-W$_{1+\epsilon}$.

Proof. We apply Lemma 50 using the notation given in the lemma. Consider any $E_{p,x} \in \mathbb{T}_1$. Let $I_{p,x} = [t_{p,x}, e_{p,x})$. We know that the optimal solution must broadcast page p at least once on the interval $[t_{p,x}, e_{p,x})$ by Lemma 47, since $[\tau_p^\beta(e_{p,x}), e_{p,x})$ is a subinterval of $[t_{p,x}, e_{p,x})$. So we can set $\lambda = 1$. By Lemma 52 we have that for any event $E_{p,x} \in \mathbb{T}_1$ there exist at least $\frac{\epsilon^2 s}{205}|[t_{p,x}, e_{p,x})|$ events ending on the interval $[t_{p,x}, e_{p,x})$ of flow time at least $\frac{c\epsilon}{20}(1-\beta k)F_{p,x}$. If we let the set $\mathcal{B}_{p,x}$ consist of these events, we can set $\mu = \frac{\epsilon^2 s}{205}$ and $\kappa = \frac{c\epsilon}{20}(1-\beta k)$. Using Lemma 48 we know that $|I_{p,x}| \geq 10000/\epsilon^2$ and therefore $d = \min_{E_{p,x} \in \mathcal{A}} |I_{p,x}| \geq 10000/\epsilon^2$. Thus we have

$$F(\mathbb{T}_1) \leq \frac{2}{\kappa\mu\lambda}\frac{d+1}{d}\text{LA-W}_{1+\epsilon} = (\frac{41}{50(1+\epsilon)})(\frac{1}{1-\beta k})\frac{d+1}{d}\text{LA-W}_{1+\epsilon} < \frac{83}{100}\text{LA-W}_{1+\epsilon}.$$

□

5.3.2 Bounding $\mathbb{T}_2$ events.

In this section, we bound $F(\mathbb{T}_2)$. Recall that our goal is to show that for any event $E_{p,x} \in \mathbb{T}_2$, the optimal solution must broadcast page p many times during $E_{p,x}$. To

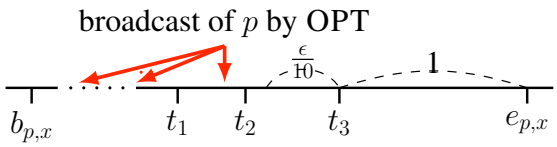


Figure 5.3: For any event $E_{p,x}$ in $\mathbb{T}_2$, OPT must broadcast page p during $[t_1, t_3)$.

find these broadcasts by the optimal solution, we break up each event $E_{p,x} \in \mathbb{T}_2$ into the time intervals $[\tau_{p,x}^{\beta,i+2}, \tau_{p,x}^{\beta,i})$. By definition of τ, we know that the requests for page p that arrive during $[\tau_{p,x}^{\beta,i+2}, \tau_{p,x}^{\beta,i+1}]$ account for a substantial portion of the flow time of event $E_{p,x}$. Knowing this and that the length of $[\tau_{p,x}^{\beta,i+1}, \tau_{p,x}^{\beta,i})$ is sufficiently long by definition of events in $\mathbb{T}_2$, we will be able to show that the optimal solution must broadcast page p during $[\tau_{p,x}^{\beta,i+2}, \tau_{p,x}^{\beta,i})$. Otherwise, these requests wait for a sufficiently long time to be scheduled by OPT and, therefore, OPT must accumulate flow time at least $\frac{1}{\gamma} F_{p,x}$ for these requests. This contradicts the fact that events in $\mathbb{T}_2$ are non-self-chargeable.

Lemma 54. *Let $E_{p,x}$ be an event in $\mathbb{T}_2$. For any integer i s.t. $0 \leq i \leq \lceil 1000c \rceil$, the optimal solution must broadcast page p during the interval $[\tau_{p,x}^{\beta,i+2}, \tau_{p,x}^{\beta,i})$.*

Proof. For any fixed integer i such that $0 \leq i \leq \lceil 1000c \rceil$, let $t_1 = \tau_{p,x}^{\beta,i+2}$, $t_2 = \tau_{p,x}^{\beta,i+1}$, and $t_3 = \tau_{p,x}^{\beta,i}$. Note that $t_3 - t_2 \geq \frac{\epsilon}{10}(e_{p,x} - t_3)$ and $t_1 < t_2 < t_3$, since $E_{p,x} \in \mathbb{T}_2$. See Figure 5.3. Let $S_{[t_1,e_{p,x})}$, $S_{(t_2,e_{p,x})}$ and $S_{[t_1,t_2]}$ be the set of requests for page p which arrive on the intervals $[t_1, e_{p,x})$, $(t_2, e_{p,x})$ and $[t_1, t_2]$, respectively. By definition of t_1 and t_2, we have that $F(S_{[t_1,e_{p,x})}) > \beta(\frac{10}{\epsilon}(i+2)+1)F_{p,x}$ and $F(S_{(t_2,e_{p,x})}) \leq \beta(\frac{10}{\epsilon}(i+1)+1)F_{p,x}$. Thus we have,

$$F(S_{[t_1,t_2]}) = F(S_{[t_1,e_{p,x})}) - F(S_{(t_2,e_{p,x})}) > \frac{10}{\epsilon}\beta F_{p,x}. \tag{5.2}$$

With the fact $t_3 - t_2 \geq \frac{\epsilon}{10}(e_{p,x} - t_3)$, the fact that the requests in $S_{[t_1,t_2]}$ arrive by time t_2, and (5.2), by applying Lemma 49 we have

$$F(S_{[t_1,t_2]}, t_3) \geq (\frac{\epsilon}{10})(\frac{10}{\epsilon})\beta F_{p,x} = \beta F_{p,x}. \tag{5.3}$$

For the sake of contradiction, suppose that the optimal solution does not broadcast page p on the interval $[t_1, t_3)$. Then

$$F^*_{p,x} \geq F(S_{[t_1,t_2]}, t_3) \geq \beta F_{p,x} \geq \frac{1}{\gamma} F_{p,x}. \tag{5.4}$$

This is a contradiction to $E_{p,x}$ being non-self-chargeable. □

Corollary 55. *For each event $E_{p,x} \in \mathbb{T}_2$, the optimal solution broadcasts page p at least $\lceil 500c \rceil$ times during the interval $[\tau^{\beta k}_{p,x}, e_{p,x})$.*

At this point, we have shown the most interesting property of events in $\mathbb{T}_2$ and we are almost ready to bound $F(\mathbb{T}_2)$. Before bounding $F(\mathbb{T}_2)$, we first find events to charge to. For each event $E_{p,x} \in \mathbb{T}_2$, we want to charge $F_{p,x}$ to some events ending during $[\tau^{\beta k}_{p,x}, e_{p,x})$ because we know OPT broadcasts page p many times during this interval. Knowing that LA-W always broadcasts the page with flow time close to the highest flow time, we can easily find events ending during $[\tau^{\beta k}_{p,x}, e_{p,x})$ with sufficiently large flow time.

Lemma 56. *Consider any event $E_{p,x} \in \mathbb{T}_2$. Let $I_{p,x} = [\tau^{\beta k}_{p,x}, e_{p,x})$. There exist at least $\frac{49}{100}(1+\epsilon)|I_{p,x}|$ events ending during $I_{p,x}$ with flow time at least $\frac{1}{2c}(1-\beta k)F_{p,x}$.*

Proof. Let $I'_{p,x} = [\tau^{\beta k}_{p,x} + \frac{1}{2}(e_{p,x} - \tau^{\beta k}_{p,x}), e_{p,x})$. Note that there are at least $\lfloor (1+\epsilon)\frac{1}{2}|I_{p,x}| \rfloor \geq (1+\epsilon)\frac{49}{100}|I_{p,x}|$ events ending during $I'_{p,x}$; the inequality is due to Lemma 48 to ensure $|I_{p,x}|$ is sufficiently long. Let $E_{q,y}$ be an event such that $e_{q,y} \in I'_{p,x}$. We now show that $F_{q,y} \geq \frac{1}{2c}(1-\beta k)F_{p,x}$. By Lemma 49 and the definition of $\tau^{\beta k}_{p,x}$ we have $F_p(e_{q,y}) \geq \frac{1}{2}(1-\beta k)F_{p,x}$. Since our algorithm chose page q over page p at time t, according to either `Rule 1` or `Rule 2`, $F_{q,y} \geq \frac{1}{c}F_p(e_{q,y})$. Hence we conclude that $F_{q,y} \geq \frac{1}{2c}(1-\beta k)F_{p,x}$. □

Finally we bound the flow time of $\mathbb{T}_2$ events by charging an event $E_{p,x} \in \mathbb{T}_2$ to the events we found in Lemma 56. Notice that the events we are charging to can have flow time less that $F_{p,x}$, but we counter this by finding many broadcasts of page p by OPT during $E_{p,x}$.

Lemma 57. *For* $0 < \epsilon \leq 1$, $F(\mathbb{T}_2) < \frac{2}{100}\text{LA-W}_{1+\epsilon}$.

Proof. We apply Lemma 50. Let $E_{p,x} \in \mathbb{T}_2$ and $I_{p,x} = [\tau_{p,x}^{\beta k}, e_{p,x})$. By Corollary 55 we can set $\lambda = 500c$. By letting $\mathcal{B}_{p,x}$ be the set of events found for $E_{p,x}$ in Lemma 56, we can set $\kappa = \frac{1}{2c}(1 - \beta k)$ and $\mu = \frac{49}{100}(1 + \epsilon)$. Using Lemma 48 we know that $|I_{p,x}| \geq 10000/\epsilon^2$ and therefore $d = \min_{E_{p,x} \in \mathcal{A}} |I_{p,x}| \geq 10000/\epsilon^2$. The desired result follows by simple calculation. □

5.3.3 There are no events in $\mathbb{T}_3$.

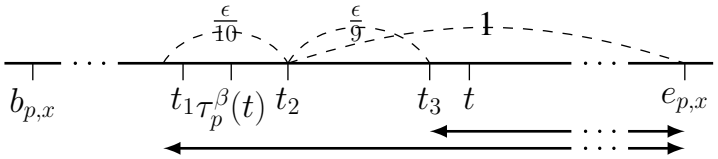


Figure 5.4: For an event $E_{p,x}$ in $\mathbb{T}_3$, during $[t_1, e_{p,x})$ OPT must make a unique broadcast for most events which end during $[t_3, e_{p,x})$.

In this section we show $\mathbb{T}_3 = \emptyset$. For the sake of contradiction suppose that $\mathbb{T}_3$ is non-empty. Fix an event $E_{p,x} \in \mathbb{T}_3$. For some *fixed* $0 \leq i \leq \lceil 1000c \rceil$ we have that $\tau_{p,x}^{\beta,i} - \tau_{p,x}^{\beta,i+1} < \frac{\epsilon}{10}(e_{p,x} - \tau_{p,x}^{\beta,i})$ because $E_{p,x} \notin \mathbb{T}_2$. Let $t_1 = \tau_{p,x}^{\beta,i+1}$ and $t_2 = \tau_{p,x}^{\beta,i}$. Let $t_3 = t_2 + \frac{\epsilon}{9}(e_{p,x} - t_2)$. Let $\mathcal{E}$ be all the non-self-chargeable events ending during $[t_3, e_{p,x})$ which were scheduled by `Rule 1` when page p was in Q. Our goal is to show that OPT must make a unique broadcast for each event in $\mathcal{E}$ on the interval $[t_1, e_{p,x})$. Then it will be shown that $|\mathcal{E}| > |[t_1, e_{p,x})| + 1$ by showing $|\mathcal{E}| \simeq (1 + \epsilon)|[t_3, e_{p,x})| > |[t_1, e_{p,x})| + 1$. Since OPT has 1 speed, this will show that OPT cannot complete these broadcasts on the interval $[t_1, e_{p,x})$. This contradiction will imply that $\mathbb{T}_3 = \emptyset$. See Figure 5.4.

Recall that by Lemma 47, for any $E_{q,y} \in \mathcal{E}$, the optimal solution must broadcast page q on the interval $[\tau_q^\beta(e_{q,y}), e_{q,y})$ because $E_{q,y}$ is non-self-chargeable. Further, note that such broadcasts are unique to $E_{q,y}$, i.e. not contained in $E_{q,y'}$ for any $y' \neq y$ because $E_{q,y'}$ and $E_{q,y}$ are disjoint by definition. For any $E_{q,y} \in \mathcal{E}$, if

we show that $\tau_q^\beta(e_{q,y}) \in [t_1, e_{p,x})$ then we will know that OPT performs these broadcasts on $[t_1, e_{p,x})$. This is where `Rule 1` will play a crucial role in our analysis. We will first show that $\tau_p^\beta(t) \geq t_1$ for all times $t \in [t_3, e_{p,x})$. By definition, if page q was scheduled by `Rule 1` and page p was in $Q(t)$ then $\tau_p^\beta(t) \leq \tau_q^\beta(t)$. Hence, for any $E_{q,y} \in \mathcal{E}$ we will have that $t_1 \leq \tau_p^\beta(e_{q,y}) \leq \tau_q^\beta(e_{q,y})$ and OPT broadcasts page q on $[t_1, e_{p,x})$.

Lemma 58. *For the event $E_{p,x} \in \mathbb{T}_3$, at any time $t \in [t_3, e_{p,x})$, $\tau_p^\beta(t) \geq t_1$.*

Proof. For the sake of contradiction assume that $\tau_p^\beta(t) < t_1$. Let $t' = \tau_p^\beta(t)$. Note that $t' < t_1 \leq t_2 < t < e_{p,x}$. Let $S_{[t_1,e_{p,x})}$, $S_{(t_2,e_{p,x})}$ and $S_{[t_1,t_2]}$ be the set of requests which arrive for page p on the intervals $[t_1, e_{p,x})$, $(t_2, e_{p,x})$, and $[t_1, t_2]$, respectively. By definition of t_1 and t_2, we have $F(S_{[t_1,e_{p,x})}) > \beta(\frac{10}{\epsilon}(i+1)+1)F_{p,x}$ and $F(S_{(t_2,e_{p,x})}) \leq \beta(\frac{10}{\epsilon}i+1)F_{p,x}$. Hence,

$$F(S_{[t_1,t_2]}) = F(S_{[t_1,e_{p,x})}) - F(S_{(t_2,e_{p,x})}) > \frac{10}{\epsilon}\beta F_{p,x}. \tag{5.5}$$

By the definition of $t' = \tau_p^\beta(t)$, we have $F(S_{(t',t_2]}, t) \leq F(S_{(t',t]}, t) \leq \beta F_p(t) \leq \beta F_{p,x}$. Since $t \geq t_2 + \frac{\epsilon}{9}(e_{p,x} - t_2)$, by Lemma 49, $\frac{\epsilon}{9}F(S_{(t',t_2]}, e_{p,x}) \leq F(S_{(t',t_2]}, t)$. Thus we have,

$$F(S_{(t',t_2]}) = F(S_{(t',t_2]}, e_{p,x}) \leq \frac{9}{\epsilon}F(S_{(t',t_2]}, t) \leq \frac{9}{\epsilon}\beta F_{p,x}. \tag{5.6}$$

Knowing that $F(S_{(t',t_2]}) \geq F(S_{[t_1,t_2]})$, this is a contradiction to (5.5). □

Finally we are ready to show that $\mathbb{T}_3 = \emptyset$. This lemma follows by using the previous lemma and counting the number of broadcasts the optimal solution must do on the interval $[t_1, e_{p,x})$. It is in the next lemma that we rely strongly on resource augmentation.

Lemma 59. *It must be the case that $\mathbb{T}_3 = \emptyset$.*

Proof. Recall that $\mathcal{E}$ is the set of all the non-self-chargeable events ending during $[t_3, e_{p,x})$ which were scheduled by `Rule 1` when page p was in Q. We first show

$|\mathcal{E}| > s(1 - \frac{34}{100}\epsilon)(e_{p,x} - t_2)$ by a simple counting argument. We know that at least $\lfloor s(1 - \frac{\epsilon}{9})(e_{p,x} - t_2) \rfloor$ events end during $[t_3, e_{p,x})$ by definition of t_3 and t_2. Among these events we know that at most $\alpha s(e_{p,x} - t_2)$ events are self-chargeable, since $E_{p,x} \in \mathcal{N}_2$; at most $\lceil s(e_{p,x} - t_2) \rceil / \lfloor \frac{10}{\epsilon} \rfloor + 1 \leq \epsilon s \frac{101}{900}(e_{p,x} - t_2)$ broadcasts are scheduled by `Rule 2`, since our algorithm performs according to `Rule 2` every $\lfloor \frac{10}{\epsilon} \rfloor$ broadcasts (the inequality is due to Lemma 48); and at most $\frac{\epsilon s}{10}(e_{p,x} - t_2)$ events were scheduled when p is not in the queue Q, since $E_{p,x} \notin \mathbb{T}_1$. By subtracting these numbers from the number of events ending during $I'_{p,x}$ and knowing that $(e_{p,x} - t_2) \geq \frac{10000}{\epsilon^2}$ by Lemma 48, we have

$$(1 - \frac{34}{100}\epsilon)(1 + \epsilon)(e_{p,x} - t_2) \leq |\mathcal{E}|. \tag{5.7}$$

Knowing that $t_2 - t_1 < \frac{\epsilon}{10}(e_{p,x} - t_2)$, we have

$$|[t_1, e_{p,x}]| < (1 + \frac{\epsilon}{10})(e_{p,x} - t_2). \tag{5.8}$$

As discussed previously, Lemma 58 implies that OPT must make a unique broadcast for each event in $\mathcal{E}$ during $[t_1, e_{p,x})$. Since the optimal solution has 1 speed, with Lemma 48, it must be the case that

$$|\mathcal{E}| \leq |[t_1, e_{p,x})| + 1 \leq (1 + \frac{\epsilon}{10000})|[t_1, e_{p,x})|. \tag{5.9}$$

By combining (5.7), (5.8), and (5.9), we have that $(1 - \frac{34}{100}\epsilon)(1 + \epsilon) < (1 + \frac{\epsilon}{10})(1 + \frac{\epsilon}{10000})$. For any $0 < \epsilon \leq 1$ this is not true, so we obtain a contradiction. □

By lemmas 53, 57 and 59 we have that $F(\mathcal{N}_2) \leq \frac{85}{100}\text{LA-W}_{1+\epsilon}$. The proof of Theorem 46 follows easily by combining this and lemmas 45 and 51. To complete the analysis it only remains to prove Lemma 50. This is done in the next section.

5.3.4 Proof of Lemma 50

Here we prove Lemma 50. The proof of this lemma relies on a generalization of Hall's theorem from chapter 4. This generalization was shown in Lemma 28. This lemma was implicitly used in [39], and later formalized in [30]. Now we are ready to prove Lemma 50.

Proof of [Lemma 50] We start by creating a bipartite graph $G = (X \cup Y, E)$. There is one vertex $u_{p,x} \in X$ for each event $E_{p,x} \in \mathcal{A}$ and there is a vertex $v_{q,y} \in Y$ for each event in $E_{q,y} \in \mathcal{B}$. Let $u_{p,x} \in X$ and $v_{q,y} \in Y$. There is an edge connecting $u_{p,x}$ and $v_{q,y}$ if and only if $E_{q,y} \in \mathcal{B}_{p,x}$. For any set $Z \subseteq X$, let $\mathcal{I}(Z)$ be the set of intervals corresponding to events in Z, i.e. $\mathcal{I}(Z) = \{\ I_{p,x} \mid u_{p,x} \in Z\}$. We let $\bigcup \mathcal{I}(Z)$ denote the union of intervals in $\mathcal{I}(Z)$. We denote the sum of length of maximal subintervals in $\bigcup \mathcal{I}(Z)$ by $|\bigcup \mathcal{I}(Z)|$. We will now show that G has a $((\frac{2}{\lambda\mu})(\frac{d+1}{d}))$-covering for X.

Consider any fixed set $Z \subseteq X$. We know the optimal solution must perform λ unique broadcasts for each event in Z during $\mathcal{I}(Z)$ and these broadcasts can only occur at integral times by definition of OPT. We observe that each maximal interval I' in $\bigcup \mathcal{I}(Z)$ contains at most $|I'| + 1 = \frac{|I'|+1}{|I'|}|I'| \leq \frac{d+1}{d}|I'|$ integers. Thus it follows that $\bigcup \mathcal{I}(Z)$ contains at most $\frac{d+1}{d}|\bigcup \mathcal{I}(Z)|$ integers. Therefore we have

$$\lambda|Z| \leq \frac{d+1}{d}|\bigcup \mathcal{I}(Z)|. \tag{5.10}$$

From now on, for simplicity, we assume that $\bigcup \mathcal{I}(Z)$ is one continuous interval; otherwise our argument can be applied to each maximal subinterval in $\bigcup \mathcal{I}(Z)$. Let $\mathcal{I}' \subseteq \mathcal{I}(Z)$ be such that for any two intervals $I_{p,x}, I_{q,y} \in \mathcal{I}'$ it is the case that $I_{p,x}$ is not completely contained in $I_{q,y}$, and also $\bigcup \mathcal{I}' = \bigcup \mathcal{I}(Z)$. By definition,

$$|\bigcup \mathcal{I}'| = |\bigcup \mathcal{I}(Z)|. \tag{5.11}$$

We order all intervals in $\mathcal{I}'$ in the increasing order of starting points. We pick intervals from $\mathcal{I}'$ one by one and label them by the order they are picked; the i_{th}

selected interval is denoted by I_i. Starting with I_1, we pick I_{i+1} so that I_{i+1} the least overlaps with I_i; here we will say I_{i+1} overlaps with I_i even when I_{i+1} starts exactly where I_i ends. Let $\mathcal{I}'_{odd}$ and $\mathcal{I}'_{even}$ be the odd indexed and even indexed intervals, respectively. WLOG, assume that $|\bigcup \mathcal{I}'_{odd}| \geq |\bigcup \mathcal{I}'_{even}|$. Since $\bigcup \mathcal{I}'_{odd}$ and $\bigcup \mathcal{I}'_{even}$ is a partition of $\bigcup \mathcal{I}'$, we know that $|\bigcup \mathcal{I}'_{odd}| + |\bigcup \mathcal{I}'_{even}| \geq |\bigcup \mathcal{I}'|$. Thus we have

$$|\bigcup \mathcal{I}'_{odd}| \geq \frac{1}{2}|\bigcup \mathcal{I}'|. \tag{5.12}$$

Let $N_G(Z)$ be the neighborhood of Z. We now show that $|N_G(Z)| \geq \mu|\bigcup \mathcal{I}'_{odd}|$. Note that $u_{p,x}$, corresponding to $I_{p,x}$ in $\mathcal{I}'_{odd}$, has at least $\mu|I_{p,x}|$ neighbors. Also note that all intervals in $\mathcal{I}'_{odd}$ are disjoint by construction of $\mathcal{I}'_{odd}$. Hence, by summing up all neighbors of vertices corresponding to intervals in $\mathcal{I}'_{odd}$, we have

$$|N_G(Z)| \geq \mu|\bigcup \mathcal{I}'_{odd}|. \tag{5.13}$$

From (5.10), (5.11), (5.12) and (5.13), We have $|N_G(Z)| \geq (\frac{\lambda\mu}{2})(\frac{d}{d+1})|Z|$ and G has a $((\frac{2}{\lambda\mu})(\frac{d+1}{d}))$-covering using Lemma 28. Let ℓ be such a covering.

$$\begin{aligned}
F(\mathcal{A}) &= \sum_{u_{p,x}\in X} F_{p,x} \\
&= \sum_{u_{p,x}v_{q,y}\in E} \ell_{u_{p,x}v_{q,y}} F_{p,x} \quad \text{[By definition of the covering]} \\
&\leq \sum_{u_{p,x}v_{q,y}\in E} \ell_{u_{p,x}v_{q,y}} \frac{F_{q,y}}{\kappa} \quad \text{[By } F_{q,y} \geq \kappa F_{p,x}\text{]} \\
&\leq (\frac{2}{\kappa\lambda\mu})(\frac{d+1}{d}) \sum_{v_{q,y}\in Y} F_{q,y} \\
&\qquad \text{[Change order of the summation and } \ell \text{ is a } ((\tfrac{2}{\lambda\mu})(\tfrac{d+1}{d}))\text{-covering]} \\
&= (\frac{2}{\kappa\lambda\mu})(\frac{d+1}{d})F(\mathcal{B}) \\
&\qquad \text{[Since } Y \text{ is the set of vertices corresponding to events in } \mathcal{B}\text{]} \\
&\leq (\frac{2}{\kappa\lambda\mu})(\frac{d+1}{d})\text{LA-W}_{1+\epsilon} \quad \text{[Since } \mathcal{B} \text{ is a subset of all events]}
\end{aligned}$$

□

5.4 Conclusion

In this chapter we have given the first $(1+\epsilon)$-speed $O(1)$-competitive algorithm for the objective of minimizing the total flow time in broadcast scheduling with unit sized pages. Bansal et al. gave another scalable algorithm with better competitive ratio which works for varying sized pages [8].

It is also important to note that the algorithm LA-W is parameterized by ϵ and so is the algorithm given in [8]. It would be interesting to show a $(1+\epsilon)$-speed $O(1)$-competitive algorithm which scales with ϵ without knowledge of ϵ.

Chapter 6

A Scalable Algorithm for Minimizing the ℓ_k-norms of Flow Time in the Broadcast Model

6.1 Introduction

In chapter 4, it was shown that LWF is 5-speed $O(1)$-competitive for average flow time in the broadcast setting. Then this result was extended to the ℓ_k norms of flow time to show that a variant of LWF is $O(k)$-speed $O(k)$-competitive. In this chapter, we focus on reducing the speed needed for an algorithm to be $O(1)$ competitive for the ℓ_k norms of flow time in the broadcast model. See chapter 2 for motivation and related work. In this chapter we consider the algorithm of [46], which is a generalization of the scalable algorithm for average flow time introduced by [8]. We show that this algorithm is $(1+\epsilon)$-speed $O(1)$-competitive algorithm for the ℓ_k norm of flow time. Given the strong lower bounds on the problem, the algorithm is essentially the best positive result that can be shown in the worst case setting up to a constant factor in the competitive ratio for fixed ϵ. Specifically, we show the following theorem.

Theorem 60. *There is an algorithm that is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^4})$-competitive for minimizing the ℓ_k-norms of flow time in the broadcast for arbitrary $1 \leq k < \infty$ and $0 < \epsilon < 1$.*

Techniques: To show that an algorithm is $O(1)$ competitive in scheduling theory, it suffices to show that at any time the increase in the algorithm's objective function is within a constant of the increase in the optimal solution's objective function. This is called a local argument. For instance, this was used to show that SETF is a scalable algorithm for the ℓ_k norms of flow time on a single machine

in the standard setting. However, this property does not hold in some scheduling problems. It was shown in [53] that no algorithm can be locally competitive with the optimal solution for the ℓ_1 norm in broadcast scheduling. This was due to the fact that an adversary may do a considerably less amount of work as compared to the algorithm to satisfy all the requests.

To avoid a local argument we use a potential function analysis. This has recently become popular in scheduling theory [36, 40, 46, 24]. See [50] for a tutorial of this technique. In this work we introduce an interesting potential function. Our potential function takes insights from [40, 46, 24, 8]. The potential function of [46] is most closely related to our potential function.

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

6.2 Algorithm and Analysis

For reference we now describe the broadcast model assumed in this chapter. We note that we consider non-uniform size pages throughout this chapter. Page p has size l_p. Over time requests arrive for specific pages. Each request is for one page and there can be multiple requests for the same page. The arrival time of a request $J_{p,i}$ is $r_{p,i}$. Each page p is divided into unit pieces $(1, p), (2, p), \ldots (l_p, p)$. One unit piece can be broadcasted by the server in a unit time slot. A request $J_{p,i}$ for page p is satisfied if it receives each of the integer pieces of page p in *sequential* order; here some pieces of other pages can be transmitted between the pieces of page p. The time a request is satisfied or completed by our algorithm is denoted as $C_{p,i}$. The flow time of a request $J_{p,i}$ is $(C_{p,i} - r_{p,i})$. The goal of the scheduler

is to minimize the ℓ_k norm of flow time $\sqrt[k]{\sum_{p,i}(C_{p,i} - J_{p,i})^k}$.

6.2.1 Fractional ℓ_k norm Flow Time

To bound the ℓ_k norm of flow time of a schedule, we will focus on bounding the total kth power flow time of a schedule, $\sum_{p,i}(C_{p,i} - r_{p,i})^k$. Here we have dropped the outer kth root. To do this, we focus on bounding the kth power flow time of a *fractional* schedule. In a fractional schedule the server is allowed to broadcast an infinitesimal amount of data for more than one page in a single time slot. Let $y_p(t)$ denote the rate at which page p is broadcasted at time t. In the fractional model, the finish time of a request is different than in the integral model. The finish time of request $J_{p,i}$ for page p is now defined to be the first time t that $\int_{r_{p,i}}^{t} y_p(t)\mathrm{dt} = l_p$. Notice that in this setting a request need not receive the unit pieces of page p sequentially. Bansal et al. [8] showed a reduction from the integral broadcast setting to the fractional broadcast setting. This reduction implies that a s-speed c-competitive algorithm for the fraction broadcast setting can be converted into an algorithm that is $s(1+\epsilon')$-speed $O(\frac{c}{\epsilon'})$ for the integral broadcast setting where $\epsilon > 0$. See [46] for details. Due to this reduction, we will focus on the fractional setting.

6.2.2 Algorithm

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

weights sum up to be exactly $\beta w(t)$ at each time for simplicity of analysis.

We denote the algorithm as BWLAPS, that is, the broadcast version of the algorithm Latest-Arrival-Processor-Sharing (LAPS) introduced in [40]. The algorithm BWLAPS devotes its processing power to the requests in $N'_a(t)$. A request $J_{p,i} \in N'_a(t)$ is processed at a rate of $x_{p,i}(t) = s\frac{w_{p,i}(t)}{\beta w(t)}$. Notice that the total speed used is at most s. By processing a request $J_{p,i}$ for page p, we mean that the page p is broadcasted. More formally, let $S_p(t)$ be the set of requests for page p in $N_a(t)$. Then page p is broadcasted at a rate of $y_p(t) = \sum_{J_{p,i} \in S_p(t)} x_{p,i}(t)$.

6.2.3 Potential Function

Let $y^*_p(t)\mathrm{dt}$ be the rate at which that OPT broadcasts page p at time t. Let $\mathtt{Opt}(t_1, t_2, p) = \int_{t_1}^{t_2} y^*_p(t)\mathrm{dt}$. For a request $J_{p,i}$ let $\mathtt{On}(t_1, t_2, J_{p,i}) = \int_{t_1}^{t_2} x_{p,i}(t)\mathrm{dt}$. We define $z_{p,i}(t)$ to be $\frac{\mathtt{On}(t,\infty,J_{p,i})\cdot\mathtt{Opt}(r_{p,i},t,p)}{l_p}$. Our potential function is now defined as,

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

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

6.2.4 Continuous Change in $\Phi(t)$

It remains to consider the change in Φ during a time interval $[t, t + \mathrm{dt}]$ when no jobs arrive or are completed. Let $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) = \sum_{J_{p,i} \in N_a(t)} k(t - r_{p,i})^{k-1}\mathrm{dt}$ and let $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) = \sum_{J_{p,i} \in N_o(t)} k(t - r_{p,i})^{k-1}\mathrm{dt}$. The values of $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t)$ and $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$ are the increase rate of the kth power flow time of BWLAPS' schedule

and OPT's, respectively. Our goal is to show that $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq \frac{1}{\beta}(\frac{2}{\epsilon})^{k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$ at all times t.

Before we proceed, we introduce some simple lemmas and propositions. The following lemma was shown in [8]; for completeness, the proof is given here.

Lemma 61. *For any request $J_{p,i} \in N_a(t)$ it is the case that $\sum_{J_{q,j} \in N_a(t), r_{q,j} \geq r_{p,j}} z_j(t) \leq (t - r_{p,i})$.*

Proof. Let $S_p(t)$ denote the set of requests in $N_a(t)$ that are for page p. First notice that $\sum_{J_{p,j} \in S_p(t)} \mathtt{On}(t, \infty, J_{p,j}) \leq l_p$ for all times t and pages p. Indeed, each request $J_{p,j} \in S_p(t)$ is satisfied once page p has been broadcasted by an amount of l_p. By definition of $\mathtt{On}$ the claim follows. For any fixed request $J_{p,i} \in N_a(t)$ we have that,

$$\begin{aligned}
& \sum_{J_{q,j} \in N_a(t), r_{q,j} \geq r_{p,i}} z_{q,j}(t) \\
= & \sum_q \sum_{J_{q,j} \in S_q(t), r_{q,j} \geq r_{p,i}} \frac{\mathtt{On}(t, \infty, J_{q,j}) \cdot \mathtt{Opt}(r_{q,j}, t, q)}{l_q} \\
\leq & \sum_q \sum_{J_{q,j} \in S_q(t), r_{q,j} \geq r_{p,i}} \frac{\mathtt{On}(t, \infty, J_{q,j}) \cdot \mathtt{Opt}(r_{p,i}, t, q)}{l_q} \\
= & \sum_q \mathtt{Opt}(r_{p,i}, t, q) \Big[\frac{1}{l_q} \sum_{J_{qj} \in S_q(t), r_{q,j} \geq r_{p,i}} \mathtt{On}(t, \infty, J_{q,j}) \Big] \\
\leq & \sum_q \mathtt{Opt}(r_{p,i}, t, q) \quad [\text{Since } \textstyle\sum_{J_{q,j} \in S_q(t)} \mathtt{On}(t, \infty, J_{q,j}) \leq l_q] \\
\leq & (t - r_{q,i}) \quad [\text{Since OPT has 1-speed}]
\end{aligned}$$

□

The following proposition easily follows from the above lemma.

Proposition 62.

$$\sum_{J_{p,i} \in N_a(t)} W_{p,i}(t) \leq \sum_{J_{p,i} \in N_a(t)} (1 + \frac{1}{\epsilon})^{k-1} w_{p,i}(t) \leq (\frac{2}{\epsilon})^{k-1} \frac{\mathrm{d}}{\mathrm{dt}} \mathrm{BWLAPS}(t).$$

We address each of the possible changes in $\Phi(t)$. First it is easy to see that the change in $\Phi(t)$ due to time is $\sum_{J_{p,i} \in N_a(t)} W_{p,i}(t)$. We now address the change in $\Phi(t)$ due to OPT's processing. Let page γ be the page OPT broadcasts at time t and let $S_\gamma(t)$ be the requests in $N_a(t)$ for page γ. First we observe for any page p that $\sum_{J_{p,j} \in S_p(t), r_{p,j} \geq r_{p,i}} \texttt{On}(t, \infty, J_{p,j}) \leq l_p$ due to the fact that the algorithm needs to broadcast page p for at most a l_p amount of time to satisfy each request for page p in $N_a(t)$. Also, we have that $\Delta\texttt{Opt}(t', t, \gamma) = \mathrm{dt}$ for any time $t' \leq t$ because the optimal solution has only 1 speed. Using these two facts, for any request $J_{p,i} \in N_a(t)$ we can bound the change in $\sum_{J_{q,j} \in N_a(t), r_{q,j} \geq r_{p,i}} z_{q,j}(t)$ to be at most $\sum_{J_{\gamma,j} \in S_\gamma(t), r_{\gamma,j} \geq r_{p,i}} \frac{\texttt{On}(t,\infty,J_{\gamma,j})}{l_\gamma} \cdot \Delta\texttt{Opt}(r_{\gamma,j}, t, \gamma) \leq \mathrm{dt}$. This allows us to upper bound the change in $\Phi(t)$ due to OPT's processing to be $\frac{1}{\epsilon} \sum_{J_{p,i} \in N_a(t)} W_{p,i}(t)$.

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 cases.

Case (a): $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) \leq \frac{1}{\beta\epsilon}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$. In this case we can charge $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t)$ and $\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t)$ directly to the optimal solution. Indeed, by Proposition 62 and simple algebra, $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq \sum_{J_{p,i} \in N_a(t)} w_{p,i}(t) + (1 + \frac{1}{\epsilon}) \sum_{J_{p,i} \in N_a(t)} W_{p,i}(t) \leq 2(\frac{2}{\epsilon})^k \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) \leq \frac{1}{\beta}(\frac{2}{\epsilon})^{k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$

Case (b): $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t) > \frac{1}{\beta\epsilon}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$. For this case the decrease in $\Phi(t)$ due to our algorithm's processing will play a crucial role to offset other increases. The total change (rate) due to our algorithm's processing is at most,

$$\frac{1}{\epsilon}\Big(\sum_{J_{p,i} \in N'_a(t) \setminus N_o(t)} \frac{\mathrm{d}}{\mathrm{dt}} z_{p,i}(t)\Big) \sum_{J_{p,i} \in N_a(t) \setminus N'_a(t)} W_{p,i}(t) \tag{6.1}$$

Note that for any $r_{p,i} \in N'_a(t) \setminus N_o(t)$, it is the case that $\frac{\mathrm{d}}{\mathrm{dt}} z_{p,i}(t) \leq \frac{\mathrm{d}}{\mathrm{dt}}\texttt{On}(t, \infty, J_{p,i}) = -\frac{w_{p,i}(t)}{\beta w(t)}$ by definition of BWLAPS. Using the assumption that

$\frac{1}{\beta\epsilon}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) < \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{BWLAPS}(t)$, we observe that

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

By simple algebra and Proposition 62 we have that

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

Thus we obtain $(6.1) \leq -\frac{s(1-\epsilon)}{\epsilon}(1-\beta(1+\frac{1}{\epsilon})^{k-1}) \sum_{J_{p,i}\in N_a(t)} W_{p,i}(t)$.

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

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

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

By taking the outer kth root in the objective function and scaling ϵ and β we

have the following theorem.

Theorem 63. *The algorithm* BWLAPS *is* $(1+\epsilon)$*-speed* $O(\frac{1}{\epsilon^3})$*-competitive for the* ℓ_k *of flow time in the fractional broadcast setting where* $k \geq 1$ *and* $0 < \epsilon < 1$.

Using the reduction from the fractional broadcast setting to the integral setting, we have Theorem 60.

6.3 Conclusions

In this chapter we have shown a scalable algorithm for the ℓ_k norms of flow time in broadcast scheduling for all $k \geq 1$. It is important to note that our algorithm depends on the speed ϵ. That is β depends on ϵ in BWLAPS. It would be interesting to show that a scalable algorithm must depend on ϵ to be $O(1)$-competitive or to give a scalable algorithm that does not explicitly depend on ϵ.

Chapter 7

Minimizing General Cost Functions

7.1 Introduction

When a client submits a job to a system, the client would like their job completed as quickly as possible. In other words, the client desires the sever to minimize the flow time of their job. The flow time F_j of job J_j is defined as $C_j - r_j$, where C_j is the time when the job J_j completes. When there are multiple unsatisfied jobs, the server is required to make a scheduling decision of which job or jobs to prioritize. The order the jobs are completed depends on a global scheduling objective. For example, minimizing the total flow time.

In this chapter, we study a single machine scheduling problem that generalizes many natural scheduling objectives. For our problem, we allow each job to have a positive real weight/importance w_j. For a job J_j with flow time F_j, a cost of $w_j g(F_j)$ is incurred for the job. The only restriction on the cost function $g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is that it is nondecreasing, so that it is never cheaper to finish a job later. The cost of a schedule is $\sum_j w_j g(F_j)$. We assume that preemption is allowed without any penalty. This framework includes several scheduling problems:

Weighted Flow Time: When $g(x) = x$, the objective becomes the total weighted flow time [16]. The total stretch is a special case of the total weighted flow time where $w_j = 1/l_j$ [20].

Weighted Flow Time Squared: If $g(x) = x^2$ then the scheduling objective is the sum of weighted squares of the flows of the jobs [9].

Weighted Tardiness with Equal Spans: Assume that there is a deadline d_j

for each job J_j that is equal to the release time of j plus a fixed span d. If $g(t) = 0$ for t not greater than the deadline d_j, and $g(t) = w_j(t - d_j)$ for t greater than the deadline $r_j + d$, then the objective is weighted tardiness.

Weighted Exponential Flow: If $g(x) = a^x$, for some real value $a > 1$, then the scheduling objective is the sum of the exponentials of the flow, which has been suggested as an appropriate objective for scheduling problems related to air traffic control, and quality control in assembly lines [12, 14].

For the latter two objectives, no non-trivial results were previously known in the online setting. Note that our general problem formulation encompasses settings where the penalty for delay may be discontinuous, as is the penalty for late filing of taxes or late payment of parking fines. To our best knowledge, minimizing a discontinuous cost function has not been been previously studied in non-stochastic online scheduling.

Our main result, given in Section 7.2, is:

- The scheduling algorithm Highest Density First (HDF) is $(2 + \epsilon)$-speed $O(1/\epsilon)$-competitive for any cost function g that is non-dcreasing and constant $0 < \epsilon < 1$.

The density of a job J_j is $d_j = \frac{w_j}{l_j}$, the ratio of the weight of the job over the size of the job. The algorithm HDF always processes the job of highest density. Note that HDF is $(2+\epsilon)$-speed $O(1)$-competitive *simultaneously* for all cost functions g. Indeed, this implies that HDF performs reasonably for highly disparate scheduling objectives such as average flow time and exponential flows. In practice it is often not clear what the scheduling objective should be. For competing objectives, tailoring an algorithm for one can come at the cost of not optimizing the other. Our analysis shows that no single objective needs to be chosen. As long as the objective falls into the very general framework we consider, HDF will optimize the objective.

In Section 7.3 we also show that it is not possible to significantly improve upon HDF, or our analysis, along several axes:

- If each job J_j has a distinct cost function g_j then there is no $O(1)$-speed $O(1)$-competitive algorithm for the objective $\sum_j g_j(F_j)$. Thus it is necessary that the cost functions for the jobs be uniform. Our lower bound instance is similar to and inspired by an instance given in Theorem 6.1 of [33].

- For any fixed $\epsilon > 0$ there is no online algorithm that is $(2 - \epsilon)$-speed $O(1)$-competitive and oblivious of the cost function. Hence HDF is essentially the optimal oblivious algorithm.

- No scalable algorithm exists for all non-decreasing functions g. In other words, while there may be a non-oblivious algorithm that is $O(1)$-competitive with less than a factor of two speed augmentation, some non-trivial speed augmentation is necessary.

All of these lower bounds hold even in the case where all jobs have unit weights. Hence, the intrinsic difficulty of the problem is unaffected by weights/priorities. All of these lower bounds hold even for randomized algorithms. Hence, randomization does not seem to be particularly useful to the online algorithm. In contrast, we show that in some special cases, scalable algorithms are achievable:

- In Section 7.4 we show that the algorithm First-In-First-Out (FIFO) is scalable when jobs have unit sizes and weights.

- In Section 7.5 we show that a variation of the algorithm Weighted Late Arrival Processor Sharing (WLAPS) is scalable when the cost function g is concave, continuous and twice-differentiable; hence $g''(F) \leq 0$ for all $F \geq 0$. When g is concave, the longer a job waits to be satisfied, the less urgent it is to complete the job. This objective can be viewed as making a few clients really happy rather than making all clients moderately happy. Although all of the scheduling literature that we are aware focuses on convex cost

functions, there are undoubtedly some applications where a concave costs function better models the scheduler's objectives. The algorithm WLAPS is oblivious to the cost function g as well as nonclairvoyant. A nonclairvoyant algorithm is oblivious to the sizes of the jobs.

7.1.1 Related Results

The online scheduling results that are probably most closely related to the results here are the results in [9], which considers the special case of our problem where the cost function is polynomial. The results in [9] are similar in spirit to the results here. They show that well-known priority scheduling algorithms have the best possible performance. In particular, [9] showed that HDF is $(1+\epsilon)$-speed $O(1/\epsilon^k)$-competitive where k is the degree of the polynomial and $0 < \epsilon < 1$. [9] also showed similar results for the scheduling algorithms Shortest Job First and Shortest Remaining Processing Time where jobs are of equal weight/importance. Notice that these results depend on the degree of the polynomial. Our work shows that HDF is $O(1)$-competitive independent of the rate of growth of the objective function when given $2+\epsilon$ resource augmentation for a fixed $0 < \epsilon < 1$. [9] also showed that any online algorithm is $n^{\Omega(1)}$-competitive without resource augmentation. The analyses of HDF in [9] essentially showed that at all times, and for all ages A, there must be $\Omega(1)$ times as many jobs of age $\Omega(A)$ in the optimal (or an arbitrary) schedule as there are in HDF's schedule. If the cost function g is arbitrary, such a statement is not sufficient to establish $O(1)$-competitiveness. In particular, if the cost function $g(F)$ grows exponentially quickly depending on F or has discontinuities, the previous analysis does not imply HDF has bounded competitiveness. We show the stronger statement that there are $\Omega(1)$ times as many jobs in the optimal schedule that are of age at least A. This necessitates that our proof is quite different than the one in [9].

It is well known that Shortest Remaining Processing Time is optimal for total flow time, when all jobs are of equal weight/importance and when $g(x) = x$.

HDF was first shown to be scalable for weighted flow, when $g(x) = x$, in [16]. The nonclairvoyant algorithm Shortest Elapse Time First is scalable for total flow time [52]. The algorithm LAPS that round robins among recently arriving jobs is also nonclairvoyant and scalable for total flow time [40]. The nonclairvoyant algorithm WLAPS, a natural extension of LAPS was shown to be scalable for weighted flow time [8], and later for weighted squares of flow time [37].

Recently, Bansal and Pruhs considered the offline version of this problem, where each job J_j has a individual cost function $g_j(x)$ [10]. The main result in [10] is a polynomial-time $O(\log \log nP)$-approximation algorithm, where P is the ratio of the size of the largest job to the size of the smallest job. This result is without speed augmentation. Obtaining a better approximation ratio, even in the special case of uniform linear cost functions, that is when $g(x) = x$, is a well known open problem. Thus it is fair to say that the problem that considers general cost functions is very challenging even in the offline setting.

7.1.2 Basic Definitions and Notation

Before our analysis, we formally define some notation. Let n denote the total number of jobs. Jobs are indexed as $J_1, J_2, ..., J_n$. Job J_i arrives at time r_i having weight/importancew_i and initial work/size l_i. For a certain schedule A, let C_i^A be the completion time of J_i under the schedule A. Let $F_i^A = C_i^A - r_i$ denote the flow time of job J_i . The cost function $g : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ is a non-decreasing function that takes a flow time and gives the cost for the flow time. That is, it incurs cost $g(F_i^A)$ for the unweighted objective and $w_i g(F_i^A)$ for the weighted objective. If the schedule is clear in context, the notation A may be omitted. Similarly, we let C_i^* and F_i^* denote the completion time and flow time of job J_i by a fixed optimal schedule. We will let $A(t)$ denote the set of unsatisfied jobs in the schedule at time t by the online algorithm A we consider. Likewise, $O(t)$ denotes the analogous set for a fixed optimal solution OPT. We will overload notation and allow A and OPT to denote the algorithms A and OPT as well as their final objective. We

will use $p_i^A(t)$ and $p_i^O(t)$ to denote the remaining work at time t for job J_i in the A's schedule and OPT's schedule, respectively. Throughout the chapter, for an interval I, we let $|I|$ denote the length of the interval I. For two intervals I and $I' \subseteq I$ we will let $I \setminus I'$ denote I with the subinterval I' removed.

7.2 Analysis of HDF

We show that Highest Density First (HDF) is $(2+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive, for any fixed $\epsilon > 0$, for the objective of $\sum_{i\in[n]} w_i g(F_i)$. We first appeal to the result in [16] that if HDF is s-speed c-competitive when jobs are unit sized then HDF is $(1+\epsilon)s$-speed $\left(\frac{1+\epsilon}{\epsilon} \cdot c\right)$-competitive when jobs have varying sizes. Although in [16], this reduction is stated only for the objective of weighted flow, it can be easily extended to our general cost objective.

Lemma 64. *[16] If* HDF *is s-speed c-competitive for minimizing $\sum_{i\in[n]} w_i g(F_i)$ when all jobs have unit size and arbitrary weights then* HDF *is $(1+\epsilon)s$-speed $\left(\frac{1+\epsilon}{\epsilon} \cdot c\right)$-competitive for the same objective when jobs have varying sizes and arbitrary weights where $\epsilon > 0$ is a constant.*

To prove Lemma 64, we first show the following lemma.

Lemma 65. *Given an online algorithm that is s-speed c-competitive for minimizing $\sum_{i\in[n]} w_i g(F_i)$ when all jobs have unit size and arbitrary weights, then there is an online algorithm that is $(1+\epsilon)s$-speed $\left(\frac{1+\epsilon}{\epsilon} \cdot c\right)$-competitive for the same objective when jobs have varying sizes and arbitrary weights where $\epsilon > 0$ is any constant.*

Proof. Let A' denote an algorithm that is s-speed c-competitive for minimizing $\sum_{i\in[n]} w_i g(F_i)$ when all jobs have unit size and arbitrary weights. Let $\epsilon > 0$ be a constant. Consider any sequence σ of n jobs with varying sizes and varying weights. From this instance, we construct a new instance σ' of unit sizes and varying weight jobs. Here we let Δ denote the unit size and it is assumed that

Δ is sufficiently small such that l_i/Δ and $\frac{\epsilon l_i}{(1+\epsilon)\Delta}$ are integers for all job J_i. For each job J_i of size l_i and weight w_i, replace this job with a set $\mathcal{U}_i$ of unit sized jobs. There are $\frac{l_i}{\Delta}$ unit sized jobs in $\mathcal{U}_i$; notice that this implies that the total size of the jobs in $\mathcal{U}_i$ is l_i. Each job in $\mathcal{U}_i$ has weight $\frac{\Delta w_i}{l_i}$. Each job in $\mathcal{U}_i$ arrives at time r_i, the same time when J_i arrived in σ. This complete the description of the instance σ'.

Let OPT denote the optimal solution for the sequence σ and OPT$'$ denote the optimal solution for the sequence σ'. Note that

$$\text{OPT}' \leq \text{OPT}. \tag{7.1}$$

This is because the most obvious schedule for σ' corresponding to OPT has cost no greater than OPT. By assumption of A', we know that with s speed, the cost of A' on σ' is at most cOPT$'$. Let $\mathcal{U}_i(t)$ denote the jobs in $\mathcal{U}_i$ that have been released but are unsatisfied by time t in A''s schedule. Let β_i denote the first time that $|\mathcal{U}_i(\beta_i)| = \frac{\epsilon l_i}{(1+\epsilon)\Delta}$; recall that $|\mathcal{U}_i(r_i)| = \frac{l_i}{\Delta}$. Knowing that each of the jobs in $\mathcal{U}_i(\beta_i)$ are completed after time β_i in A''s schedule and that $g()$ is non-decreasing, we have

$$\sum_{i\in[n]} |\mathcal{U}_i(\beta_i)| \frac{\Delta w_i}{l_i} g(\beta_i) = \sum_{i\in[n]} \frac{\epsilon w_i}{1+\epsilon} g(\beta_i) \leq A' \tag{7.2}$$

Now consider constructing an algorithm A for the sequence σ based on A'. Whenever the algorithm A' schedules a job in $\mathcal{U}_i$ then the algorithm A processes job J_i at a $(1+\epsilon)$ faster rate of speed (unless J_i is completed). We assume that at any time A has at most one unit sized job $\mathcal{U}_i$ that has been partially proceeded. The algorithm A will complete the job J_i at time β_i. This is because A' completed $\frac{l_i}{\Delta} - \frac{\epsilon l_i}{(1+\epsilon)\Delta} = \frac{l_i}{(1+\epsilon)\Delta}$ jobs in $\mathcal{U}_i$ before β_i. This required A' spending at least $\frac{\Delta}{s} \cdot \frac{l_i}{(1+\epsilon)\Delta} = \frac{l_i}{(1+\epsilon)s}$ time units on jobs in $\mathcal{U}_i$ since A' has s speed and it takes $\frac{\Delta}{s}$ time units for A' to complete a unit sized job. By definition of A, the algorithm A with $(1+\epsilon)s$-speed did at least $\frac{l_i}{(1+\epsilon)s} \cdot (1+\epsilon)s = l_i$ volume of work

for jobs in $\mathcal{U}_i$ by time β_i. Hence A completed each job J_i completed by time β_i. Knowing this and by (7.1) and (7.2), we have

$$\begin{aligned} A &= \sum_{i\in[n]} w_i g(\beta_i) = \frac{1+\epsilon}{\epsilon} \sum_{i\in[n]} \frac{\epsilon w_i}{1+\epsilon} g(\beta_i) \\ &\leq \frac{1+\epsilon}{\epsilon} A' \leq \frac{1+\epsilon}{\epsilon} c\mathrm{OPT}' \qquad \text{[By definition of } A\text{]} \\ &\leq \frac{1+\epsilon}{\epsilon} c\mathrm{OPT} \end{aligned}$$

Knowing that A processes jobs at most $(1+\epsilon)$ faster than A', we have that A is $(1+\epsilon)s$-speed $\frac{(1+\epsilon)}{\epsilon}c$-competitive for σ.

□

We now prove Lemma 64.

Proof of [Lemma 64] Consider any sequence σ of jobs where jobs have varying sizes and weights. To prove this lemma consider the conversion of σ to σ' in Lemma 65 and consider setting the algorithm A' to HDF. Let A denote the algorithm which is generated from HDF in the proof of Lemma 65. To prove the lemma we prove a stronger statement by induction on time. We will show that at any time t HDF on σ has worked on every job at least as much as A on σ. Here HDF and A are both given the same speed.

We prove this by induction on time t. When $t = 0$ the claim clearly holds. Now consider any time $t > 0$ and assume HDF has worked on every job at least as much as A every time before t. Now consider time t. If A does not schedule a job at time t, then the claim follows. Hence, we can assume A schedules some job J_i at time t. Notice that in the proof of Lemma 65 when generating a set of unit sized jobs $\mathcal{U}_i$ from J_i the density of the unit sized jobs in $\mathcal{U}_i$ is the same as the density of job J_i. Knowing that HDF has worked at least as much as A on every job and the definition of HDF, this implies that if J_i is unsatisfied in HDF's schedule at time t then HDF will schedule job J_i. Otherwise J_i is finished

in HDF's schedule at time t. In either case, after time t HDF scheduled each job at least as much as A on every job. Knowing that HDF worked at least as much as A on every job at all times, Lemma 65 gives the claim. □

This reduction slices each job J_i into unit sized jobs of the same density whose total size is l_i. Reducing from an integral objective to a fractional objective has become standard, e.g. [16, 9, 24]. Therefore it is sufficient to show that HDF is 2-speed $O(1)$-competitive for unit-sized jobs. Thus we will make our analysis assuming that all jobs have unit size, which can be set to 1 without loss of generality by scaling the instance. We assume without loss of generality that weights are no smaller than one. For the sake of analysis, we partition into jobs into classes $\mathcal{S}_l, l \geq 0$ depending on their weight: $\mathcal{S}_l := \{J_i \mid 2^l \leq w_i < 2^{l+1}\}$. We let $\mathcal{S}_{\geq l} := \bigcup_{l' \geq l} \mathcal{S}_{l'}$. Consider any input sequence σ where all jobs have unit size. We consider the algorithm HDF with 2 speed-up. Note that HDF always schedules the job with the largest weight when jobs have unit size. We assume that HDF breaks ties in favor of the job that arrived the earliest. To prove the competitiveness of HDF on the sequence σ, we will recast our problem into a network flow where a feasible maximum flow maps flow times of the jobs in the algorithm's schedule and those in the optimal solution's schedule. The weight of each job in the algorithm's schedule will be charged to jobs in the optimal solution's schedule that have flow time at least as large. Moreover, the total weight of the algorithm's jobs mapped to a single job J_i in the optimal solution's schedule will be bounded by $O(w_i)$. Once this is established, the competitiveness of HDF follows.

Formally, the network flow graph $G = (V = \{s\} \cup X \cup Y\{t\}, E)$ is constructed as follows. We refer the reader to Figure 7.1. There are source and sink vertices s and t respectively. There are two partite sets X and Y. There is a vertex $v_{x,i} \in X$ and a vertex $v_{y,i} \in Y$ corresponding to job J_i. Intuitively, the vertices in X correspond to jobs in the algorithm's schedule and those in Y correspond to jobs in the optimal solution's schedule. There is an edge $(s, v_{x,i})$ with capacity w_i for all $i \in [n]$. There is an edge $(v_{x,i}, t)$ with capacity $8w_i$ for all $i \in [n]$. Making

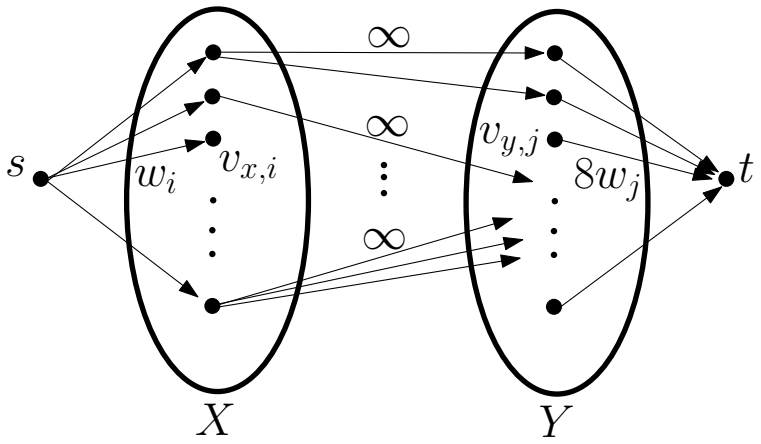


Figure 7.1: The graph G.

the capacity the of edge $(v_{y,i}, t)$ equal to $8w_i$ ensures that job J_i in the optimal solution's schedule is not overcharged. There exists an edge $(v_{x,i}, v_{y,j})$ of capacity ∞ if $F_i \leq F_j^*$ and $w_i \leq w_j$. Recall that F_i and F_i^* denote the flow time of job J_i in the algorithm's and the optimal solution's schedule respectively.

Our main task left is to show the following lemma.

Lemma 66. *The minimum cut in the graph G is $\sum_{i\in[n]} w_i$.*

Assuming that Lemma 66 holds, we can easily prove the competitiveness of HDF for unit sized jobs.

Theorem 67. HDF *is 2-speed 8-competitive for minimizing $\sum_{i\in[n]} w_i \cdot g(F_i)$ when all jobs are unit sized.*

Proof. Lemma 66 implies that the maximum flow f is $\sum_{i\in[n]} w_i$. Let $f(u, v)$ denote the flow on the edge (u, v). Note that the maximum flow is achieved only when $f(s, v_{x,i}) = w_i$ for all jobs $i \in [n]$. We charge the cost of each job in the algorithm's schedule to the optimal cost in the most obvious way as suggested by

the maximum flow. That is, by charging $w_i g(F_i)$ to $\sum_j f(v_{x,i}, v_{y,j}) g(F_j^*)$ we have:

$$\begin{aligned}
\text{HDF} &= \sum_{i\in[n]} w_i g(F_i) \\
&= \sum_{i\in[n]} \sum_{j\in[n]} f(v_{x,i}, v_{y,j}) g(F_i) \quad [\text{Since } f \text{ is conserved at } v_{x,i}] \\
&\leq \sum_{i\in[n]} \sum_{j\in[n]} f(v_{x,i}, v_{y,j}) g(F_j^*) \quad [\text{Since } (v_{x,i}, v_{y,i}) \in E \text{ only if } F_i \leq F_j^*] \\
&= \sum_{j\in[n]} f(v_{y,j}, t) g(F_j^*) \quad [\text{Since } f \text{ is conserved at } v_{y,j}] \\
&\leq \sum_{j\in[n]} 8w_j g(F_j^*) \quad [8w_j \text{ is the capacity on } v_{y,j}] \\
&= 8\text{OPT}
\end{aligned}$$

□

By Lemma 64 and Theorem 67, we obtain

Theorem 68. HDF *is* $(2+\epsilon)$*-speed* $O(\frac{1}{\epsilon})$*-competitive for minimizing* $\sum_{i\in[n]} w_i g(F_i)$ *when jobs have arbitrary sizes and weights.*

The remaining section is devoted to proving Lemma 66. Let (S, T) be a minimum s-t cut. For notational simplicity, for any pair of disjoint subsets of vertices A and B, we allow (A, B) to denote the set of edges from vertices in A to vertices in B. We let $c(e)$ denote the capacity of edge e and $c(A, B)$ the total capacity of all edges in (A, B). Let $X_s = X \cap S$, $X_t = X \cap T$, $Y_s = Y \cap S$ and $Y_t = Y \cap T$. Note that all edges in $(\{s\}, X_t)$ are cut by the cut (S, T), i.e. $(\{s\}, X_t) \subseteq (S, T)$ and $c(\{s\}, X_t) = \sum_{v_{x,i}\in X_t} w_i$. Knowing that $(Y_s, \{t\}) \subseteq (S, T)$, it suffices to show that

$$8 \sum_{v_{y,j}\in Y_s} w_j \geq \sum_{v_{x,i}\in X_s} w_i. \tag{7.3}$$

This suffices because if we assume (7.3) is true, we have $c(S, T) \geq \sum_{v_{x,i}\in X_t} w_i + 8\sum_{v_{y,j}\in Y_s} w_j \geq \sum_{v_{x,i}\in X_t} w_i + \sum_{v_{x,i}\in X_s} w_i = \sum_{i\in[n]} w_i$.

Our attention is focused on showing (7.3). For any $V' \subseteq V$, let $N(V')$ denote the set of out-neighbors of V', i.e. $N(V') = \{z \mid (v, z) \in E, v \in V'\}$. Since

(S,T) is a minimum s-t cut, (S,T) does not contain an edge connecting a vertex in X to a vertex in Y; recall that such an edge has infinite capacity. Therefore $N(X_s) \subseteq Y_s$ where $N(X_s)$ is the out-neighborhood of the vertices in X_s. For any positive integer l, define $\mathcal{S}_l(X_s) := \{v_{x,i} \mid v_{x,i} \in X_s, J_i \in \mathcal{S}_l\}$; recall that J_i is in class $\mathcal{S}_l$ if $2^l \leq w_i < 2^{l+1}$. We show the following key lemma. Here it is shown that the neighborhood of $\mathcal{S}_l(X_s)$ is large compared to $|\mathcal{S}_l(X_s)|$.

Lemma 69. *The vertices in $\mathcal{S}_l(X_s)$ have at least $\frac{1}{2}|\mathcal{S}_l(X_s)|$ neighbors in Y, i.e. $|N(\mathcal{S}_l(X_s)) \cap Y| \geq \frac{1}{2}|\mathcal{S}_l(X_s)|$.*

Proof. Consider each maximal busy time interval I where HDF is always scheduling jobs in $\mathcal{S}_{\geq l}$. Let $C(I,l)$ be the set of jobs in $\mathcal{S}_l(X_s)$ which are completed by HDF during the interval I. Let J_k be the job that is in $\mathcal{S}_l(X_s)$ which is completed during the interval I and has the highest priority in HDF's schedule (if such a job exists). This implies that the job J_k has the shortest flow time of any job in $\mathcal{S}_l(X_s)$ that is completed during the interval I. We will show that $v_{x,k}$ has at least $\frac{1}{2}|C(I,l)|$ neighbors in Y, i.e.

$$|N(\{v_{x,k}\}) \cap Y| \geq \frac{1}{2}|C(I,l)| \tag{7.4}$$

and all jobs corresponding to these neighbors were completed by HDF during I. Taking a union over all maximal busy intervals will complete the proof.

We now focus on proving (7.4). Recall that $F_k = C_k - r_k$ is the flow time of job J_k. Since J_k has the highest priority among all jobs in $C(I,l)$, J_k is not preempted during $[r_k, C_k]$ by any job in $C(I,l)$ (but could be by higher priority jobs not in $C(I,l)$). Hence J_k is the only job in $C(I,l)$ that is completed during $[r_k, C_k]$. Now we count the number of jobs in $C(I,l)$ that are completed during $I \setminus [r_k, C_k]$. Since HDF has 2-speed, HDF can complete at most $2|I| - 2F_k$ volume of work during $I \setminus [r_k, C_k]$. Since we assumed all jobs have unit size, the number of such jobs is at most $\lfloor 2|I| - 2F_k \rfloor$. Hence, using this and by including J_k itself we obtain

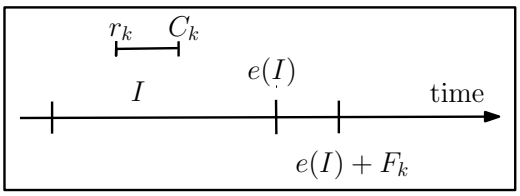


Figure 7.2: The interval I in HDF's schedule.

$$\lfloor 2|I| - 2F_k \rfloor + 1 \geq |C(I, l)| \tag{7.5}$$

We now lowerbound $|N(\{v_{x,k}\}) \cap Y|$ to show (7.4). Roughly speaking, we want show that OPT has many jobs of flow time at least F_k. Let $\mathcal{J}_{\text{HDF}}(I)$ be the set of jobs that are completed by HDF during I. Note that all jobs in $\mathcal{J}_{\text{HDF}}(I)$ must arrive during the interval I. For the sake of contradiction, suppose this is not true, i.e. there is a job J_j that arrives before the start of I and completes during I. Then HDF must be busy processing jobs of weight as high as J_j during $[r_j, C_j]$, contradicting the definition of the interval I being maximal. Consider the time at $e(I) + F_k$ where $e(I)$ is the ending time of the interval I. Since the volume of jobs in $\mathcal{J}_{\text{HDF}}(I)$ is $2|I|$ (recall that HDF has 2 speed) and OPT can process at most $|I| + F_k$ volume of work during $I \cup [e(I), e(I) + F_k]$, OPT must have at least $2|I| - (|I| + F_k) = |I| - F_k$ volume of jobs in $\mathcal{J}_{\text{HDF}}(I)$ left at the time $e(I) + F_k$. See the Figure 7.2. Therefore if $|I| - F_k$ is an integer, OPT has at least $|I| - F_k + 1$ jobs in $\mathcal{J}_{\text{HDF}}(I)$ that have flow time at least F_k; here one extra job that is completed by OPT exactly at time $e(I) + F_k$ is counted. If $|I| - F_k$ is not integral, then OPT has at least $\lceil |I| - F_k \rceil$ jobs in $\mathcal{J}_{\text{HDF}}(I)$ that have flow time at least F_k. In both cases, we conclude that OPT has at least $\lfloor |I| - F_k \rfloor + 1$ jobs in $\mathcal{J}_{\text{HDF}}(I)$ that have flow time at least F_k. All such jobs have weight at least 2^l, since they are in $\mathcal{J}_{\text{HDF}}(I)$. Hence the vertices in Y corresponding to such jobs are neighbors of $v_{x,k}$ and we have

$$|N(\{v_{x,k}\}) \cap Y| \geq \lfloor |I| - F_k \rfloor + 1 \tag{7.6}$$

The inequalities (7.5) and (7.6) proves (7.4) and the lemma follows. □

Now we are ready to complete the proof of Lemma 66. For a subset $S \subseteq X$ let $N(S)$ denote the out neighborhood S and let $N(S,l) := N(S) \cap \mathcal{S}_l$. By Lemma 69 we have,

$$\begin{aligned}
\sum_{v_{x,i} \in \mathcal{S}_l(X_s)} w_i &\leq |\mathcal{S}_l(X_s))|2^{l+1} \leq 2|N(\mathcal{S}_l(X_s))|2^{l+1} \quad \text{[By Lemma 69]} \\
&= 2\sum_{h \geq l} |N(\mathcal{S}_l(X_s), h)|2^{l+1} = 2\sum_{h \geq l} \sum_{v_{y,j} \in N(\mathcal{S}_l(X_s),h)} 2^{l+1} \\
&= 4\sum_{h \geq l} \frac{1}{2^{h-l}} \sum_{v_{y,j} \in N(\mathcal{S}_l(X_s),h)} 2^h = 4\sum_{h \geq l} \frac{1}{2^{h-l}} \sum_{v_{y,j} \in N(\mathcal{S}_l(X_s)),h)} w_j
\end{aligned}$$

Using this we have that,

$$\begin{aligned}
\sum_{v_{x,i} \in X_s} w_i &= \sum_l \sum_{v_{x,i} \in \mathcal{S}_l(X_s)} w_i \leq \sum_l 4\sum_{h \geq l} \frac{1}{2^{h-l}} \sum_{v_{y,j} \in N(\mathcal{S}_l(X_s),h)} w_j \\
&\leq 4\sum_h \sum_{l \leq h} \frac{1}{2^{h-l}} \sum_{v_{y,j} \in N(\mathcal{S}_l(X_s),h)} w_j \leq 4\sum_h \sum_{l \leq h} \frac{1}{2^{h-l}} \sum_{v_{y,j} \in N(X_s,h)} w_j \\
&\leq 8\sum_h \sum_{v_{y,j} \in N(X_s,h)} w_j \leq 8 \sum_{v_{y,j} \in Y_s} w_j
\end{aligned}$$

This completes proving (7.3) and Lemma 66.

7.3 Lower Bounds

In this section we show that there is no scalable algorithm, there is no better oblivious algorithm than HDF, and the uniform cost functions are necessary to obtain $O(1)$-speed $O(1)$-competitiveness. All these lower bounds hold even for randomized algorithms.

Theorem 70. *There exists a non-decreasing function g such that for any constant $\epsilon > 0$, no randomized online algorithm is constant competitive with speed $7/6 - \epsilon$*

for the objective of $\sum_j g(F_j)$.

Proof. We will rely on Yao's Min-max Principle to prove a lower bound on the competitive ratio of any randomized online algorithm [21]. The randomized instance is constructed as follows. Consider the cost function $g(F) = 2c$ for $F > 15$ and $g(F) = 0$ for $0 \leq F \leq 15$ where $c \geq 1$ is an arbitrary constant. The job instance is as follows.

- J_b: one big job of size 15 that arrives at time 0.
- S_1: a set of small jobs that arrive at time 10. Each job has size $\frac{35-30s}{c}$ and the total size of jobs in S_1 is 10.
- S_2: a set of small jobs that arrive at time 15. Each job has size $\frac{35-30s}{c}$ and the total size of jobs in S_2 is 10.

For simplicity, we assume that $\frac{10c}{35-30s}$ is an integer. The job J_b and the set S_1 of jobs arrive with probability 1, while the set S_2 of jobs arrives with probability $\frac{1}{2c}$. Let $\mathcal{E}$ denote the event that the set S_2 of jobs arrives.

Consider any deterministic algorithm A. We will consider two cases depending on whether A finishes J_b by time 15 or not. Note that A's scheduling decision concerning whether A completes J_b by time 15 or not does not depend on the jobs in S_2, since jobs in S_2 arrive at time 15. We first consider the case where A did not finish the big job J_b by time 15. Conditioned on $\neg\mathcal{E}$, A's cost is at least $2c$. Hence A has an expected cost at least $2c(1 - \frac{1}{2c}) \geq c$. Now consider the case where A completed J_b by time 15. For this case, say the event $\mathcal{E}$ occurred. Let $V(S,t) := \sum_{j \in S} p^A(t)$ denote the remaining volume, under A's schedule, of all jobs in some set S at time t. Let $s = 7/6 - \epsilon$ be the speed that A is given where $\epsilon > 0$ is a fixed constant. Since A spent $\frac{15}{s}$ amount of time during $[0, 15]$ working on J_b, A could have spent at most $15 - \frac{15}{s}$ time on jobs in S_1. Hence it follows $V(S_1, 15) \geq 10 - s(15 - \frac{15}{s}) = 25 - 15s$ and $V(S_2, 15) = 10$. Since A can process at most $15s$ volume of work during $[15, 30]$, we have $V(S_1 \cup S_2, 30) \geq 35 - 30s = 30\epsilon$. Since each job in $S_1 \cup S_2$ has size $\frac{35-30s}{c}$, the number of jobs left is at least c. Since

at time 30, each job has flow time at least 15, the algorithm A has total cost no smaller than $2c^2$. Recalling that $\Pr[\mathcal{E}] = \frac{1}{2c}$, A's expected cost is at least c.

Now let us look at the adversary's schedule. Conditioned on $\neg\mathcal{E}$, the adversary completes J_b first and all jobs in S_1 by time 25, thereby having no cost. Conditioned on $\mathcal{E}$, the adversary delays the big job J_b until it completes all jobs in S_1 and S_2 by time 20 and 30, respectively. Note in this schedule that each job in $S_1 \cup S_2$ has flow time at most 15. The adversary has cost $2c$ only for the big job. Hence the expected cost of the adversary is $\frac{1}{2c}(2c) = 1$. This together with the above argument that A's expected cost is at least c shows that the competitive ratio of any online algorithm is at least c. Since this holds for any constant c, the theorem follows.

□

Theorem 71. *For any constant $\epsilon > 0$, there is no oblivious randomized online algorithm that is $O(1)$-competitive for the objective of $\sum_j g(F_j)$ with speed augmentation $2 - \epsilon$ for all non decreasing functions g.*

Proof. We appeal to Yao's Min-max Principle [21]. Let A be any deterministic online algorithm. Consider the cost function g such that $g(F) = 2c$ for $F > D$ and $g(F) = 0$ for $0 \leq F \leq D$. The constant D is hidden to A, and is set to 1 with probability $\frac{1}{2c}$ and to $n+1$ with probability $1 - \frac{1}{2c}$. Let $\mathcal{E}$ denote the event that $D = 1$. At time 0, one big job J_b of size $n+1$ is released. At each integer time $1 \leq t \leq n$, one unit sized job J_t is released. Here n is assumed to be sufficiently large such that $\epsilon(n+1) - 1 > c$. Note that the event $\mathcal{E}$ has no effect on A's scheduling decision, since A is ignorant of the cost function.

Suppose the online algorithm A finished the big job J_b by time $n+1$. Further, say the event $\mathcal{E}$ occurs; that is $D = 1$. Since $2n+1$ volume of jobs in total were released and A can process at most $(2-\epsilon)(n+1)$ amount of work during $[0, n+1]$, A has at least $2n+1-(2-\epsilon)(n+1)$ volume of unit sized jobs unfinished at time $n+1$. Each of such unit sized jobs has flow time greater than 1, hence A has total cost at least $2c(\epsilon(n+1)-1)) > 2c^2$. Knowing that $\Pr[\mathcal{E}] = \frac{1}{2c}$, A has an expected

cost greater than c. Now suppose A did not finish J_b by time $n+1$. Conditioned on $\neg\mathcal{E}$, A has cost at least $2c$. Hence A's expected cost is at least $2c(1-\frac{1}{2c}) > c$.

We now consider the adversary's schedule. Conditioned on $\mathcal{E}$ ($D = 1$), the adversary completes each unit sized job within one unit time hence has a non-zero cost only for J_b, so has total cost $2c$. Conditioned on $\neg\mathcal{E}$ ($D = n+1$), the adversary schedules jobs in a first in first out fashion thereby having cost 0. Hence the adversary's expected cost is $\frac{1}{2c}(2c) = 1$. The claim follows since A has cost greater than c in expectation. □

Theorem 72. *There is no randomized online algorithm that is $O(1)$-speed $O(1)$-competitive for the objective of $\sum_j g_j(F_j)$ even if the algorithm knows all functions g_j and the functions are non-decreasing.*

Proof. To show a lower bound on the competitive ratio of any randomized algorithm, we appeal to Yao's Min-max Principle [21] and construct a distribution on job instances for which any deterministic algorithm performs poor compared to the optimal schedule. All cost functions g_i have a common structure. That is, each job J_i is completely defined by two quantities d_i and λ_i, which we call J_i's relative deadline and cost respectively: $g_i(F_i) = 0$ for $0 \leq F_i \leq d_i$ and $g_i(F_i) = \lambda_i$ for $F_i > d_i$. Hence J_i incurs no cost if completed by time $r_i + d_i$ and cost λ_i otherwise. Recall that r_i and l_i are J_i's arrival time and size respectively. For this reason, we will say that J_i has deadline $r_i + d_i$. For notational convenience, let us use a compact notation $(r_i, r_i + d_i, l_i, \lambda_i)$ to characterize all properties of each job J_i where l_i is J_i's size.

Let h, T, L be integers such that $h \geq 2s$, $T = 2^h$, $L > 2cT^2$. For each integer $0 \leq l \leq h = 2s$, there is a set $\mathcal{C}_l$ of jobs (according to a distribution we will define soon, some jobs in $\mathcal{C}_l$ may or may not arrive). All jobs have deadlines no greater than T. We first describe the set $\mathcal{C}_0$. In $\mathcal{C}_0$, all jobs have size 1 and relative deadline 1, and there is exactly one job that arrives at each unit time. The job with deadline t has cost L^t. More concretely, $\mathcal{C}_0 = \{(t-1, t, 1, L^t) \mid t \text{ is an integer such that } 1 \leq t \leq T\}$. Note that $|\mathcal{C}_0| = T$. We now describe the other sets of jobs $\mathcal{C}_l$ for each

integer $1 \leq l \leq h$. All jobs in $\mathcal{C}_l$ have size 2^{l-1} and relative deadline 2^l, and at every 2^l time steps, exactly one job in $\mathcal{C}_l$ arrives. The job with deadline t has cost L^t. Formally, $\mathcal{C}_l = \{(2^l(j-1), 2^l j, 2^{l-1}, L^{2^l j}) \mid j \text{ is an integer such that } 1 \leq j \leq 2^{h-l}\}$. Note that $|\mathcal{C}_l| = 2^{h-l}$. Let $\mathcal{C} = \bigcup_{0 \leq l \leq h} \mathcal{C}_l$. Notice that all jobs with deadline t have cost L^t.

As we mentioned above, jobs in $\mathcal{C}$ do not arrive according to a probability distribution. To formally define such a distribution on job instances, let us group jobs depending on their arrival time. Let $\mathcal{R}_t$ denote the set of jobs in $\mathcal{C}$ that arrive at time t. Let $\mathcal{R}_{\leq t} := \bigcup_{0 \leq t' \leq t} \mathcal{R}_{t'}$. We let $\mathcal{E}_t$, $0 \leq t \leq T-1$ denote the event that all jobs in $\mathcal{R}_{\leq t}$ arrive and these are the only jobs that arrive. Let $\mathtt{Pr}[\mathcal{E}_t] = \frac{1}{L^t \theta}$ where $\theta = \sum_{0 \leq j \leq T-1} \frac{1}{L^j}$ is a normalization factor to ensures that $\sum_{0 \leq t \leq T-1} \mathtt{Pr}[\mathcal{E}_t] = 1$.

The following lemma will reveal a nice structure of the instance we created. Let $\mathcal{D}_t$ denote the set of jobs in $\mathcal{C}$ that have deadline t. Let $\mathcal{D}_{>t} := \bigcup_{t < t' \leq T} \mathcal{D}_{t'}$.

Lemma 73. *Consider the occurrence of event $\mathcal{E}_t$, $0 \leq t \leq T-1$. There exists a schedule with speed 1 that completes all jobs in $\mathcal{R}_{\leq t} \cap \mathcal{D}_{>t}$ before their deadline. Further such a schedule has cost at most $2TL^t$.*

Proof. We first argue that all jobs in $\mathcal{R}_{\leq t} \cap \mathcal{D}_{>t}$ can be completed before their deadline. Observe that there exists exactly one job in $\mathcal{C}_l \cap \mathcal{R}_{\leq t} \cap \mathcal{D}_{>t}$ for each l. This is because the intervals $\{[2^l(j-1), 2^l j] \mid j \text{ is an integer s.t. } 1 \leq j \leq 2^{h-l}\}$ defined by the arrival time and deadline of jobs in $\mathcal{C}_l$ is a partition of the time interval $[0, T]$. We schedule jobs in $\mathcal{R}_{\leq t} \cap \mathcal{D}_{>t}$ in increasing sizes. Hence the first job we schedule is the job in $\mathcal{C}_0 \cap \mathcal{R}_{\leq t} \cap \mathcal{D}_{>t}$ and it has no choice other than being scheduled exactly during $[t, t+1]$. Now consider each job J_i in $\mathcal{C}_l \cap \mathcal{R}_{\leq t} \cap \mathcal{D}_{>t}$. It is not difficult to see that either of $[2^l(j-1), 2^l(j-1)+2^{l-1}]$ or $[2^l(j-1)+2^{l-1}, 2^l j]$ is empty and therefore is ready to schedule the job J_i of size 2^{l-1} in $\mathcal{C}_l \cap \mathcal{R}_{\leq t} \cap \mathcal{D}_{>t}$. Finally, we upper bound the cost of the above schedule. Since all jobs with deadline greater than t are completed before their deadline under the schedule, each job can incur cost at most L^t. Knowing that there are at most $2T$ jobs, the

total cost is at most $2TL^t$. □

Corollary 74. $\mathbb{E}[\text{OPT}] \leq \frac{2T^2}{\theta}$.

Proof. Recall that $\Pr[\mathcal{E}_t] = \frac{1}{L^t\theta}$. By Lemma 73, we know, in case of the occurrence of event $\mathcal{E}$, that there exists a feasible schedule with speed 1 that results in cost at most $2TL^t$. Hence we have $\mathbb{E}[\text{OPT}] \leq \sum_{0\leq t<T} 2TL^t \frac{1}{L^t\theta} = \frac{2T^2}{\theta}$. □

We now show any deterministic algorithm A performs much worse than in expectation than the optimal schedule OPT.

Lemma 75. *Any deterministic algorithm A given speed less than s has cost at least $\frac{L}{\theta}$ in expectation.*

Proof. Note that the total size of jobs in $\mathcal{C}_0$ is T and the total size of jobs in each $\mathcal{C}_l$, $1 \leq l \leq h$ is $T/2$. Hence the total size of all jobs in $\mathcal{C}$ is at least $(h/2+1)T \geq (s+1)T$. The algorithm A, with speed s, cannot complete all jobs in $\mathcal{C}$ before their deadline, since all jobs have arrival times and deadlines during $[0, T]$. Let J_i be a job in $\mathcal{D}_{t+1}$ that A fails to complete before its deadline for an integer $0 \leq t \leq T-1$. Note that J_i arrives no later than t since all jobs have size at least 1. Further, the decision concerning whether A completes J_i before its deadline or not has nothing to do with jobs in $\mathcal{R}_{t+1}$. Hence it must be the case that for at least one of the events $\mathcal{E}_0, ..., \mathcal{E}_t$, A does not complete J_i by time $t+1$, which incurs an expected cost of at least $\frac{1}{L^t\theta}L^{t+1} \geq \frac{L}{\theta}$. □

By Yao's Min-max Principle, Corollary 74 and Lemma 75 shows the competitive ratio of any randomized algorithm is at least $\frac{L}{\theta} \Big/ \frac{2T^2}{\theta} = \frac{L}{2T^2} > c$. □

7.4 Analysis of FIFO for Unit Size Jobs

In this section we will show that FIFO is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive for minimizing $\sum_{i\in[n]} g(F_i)$ when jobs have uniform sizes and unit weights. Without loss of generality, we can assume that all jobs have size 1, since jobs are allowed

to arrive at arbitrary times. The proof follows similarly as in the case where jobs have unit size and arbitrary weight. Recall that in the previous section we charged the flow time of a job in the algorithm's schedule to jobs in the optimal solution's schedule that have larger flow time. In this case we can get a tighter bound on the number of jobs in the optimal solution's schedule that a job in FIFO's schedule can charge to, which allows us to reduce the resource augmentation.

Consider an input sequence σ and fix a constant $0 < \epsilon \leq \frac{1}{2}$. Let F_i denote the flow time of job J_i in FIFO's schedule and F_i^* be the flow time of J_i in OPT's schedule. Let $G = (V, E)$ be a flow network. There are source and sink vertices s and t, respectively. As before, there are two partite sets X and Y. There is a vertex $v_{x,i} \in X$ and a vertex $v_{y,i} \in Y$ corresponding to job J_i for all $i \in [n]$. There is an edge $(s, v_{x,i})$ with capacity 1 for all $i \in [n]$. There is an edge $(v_{y,i}, t)$ with capacity $\frac{4}{\epsilon^2}$ for all $i \in [n]$. There exists an edge $(v_{x,i}, v_{y,j})$ of capacity ∞ if $F_i \leq F_j^*$. The focus of this section is showing the following lemma.

Lemma 76. *The maximum flow in G is n.*

Assuming that this lemma is true, then the following theorem can be shown.

Theorem 77. FIFO *is $(1+\epsilon)$-speed $\frac{4}{\epsilon^2}$-competitive for minimizing $\sum_{i\in[n]} g(F_i)$ when all jobs are unit sized.*

Proof. Lemma 76 states that the maximum flow in G is n. Let f denote a maximum flow in G and let $f(u, v)$ be the flow on an edge (u, v). Note that the

maximum flow is achieved only when $f(s, v_{x,i}) = 1$ for all $i \in [n]$. We have that,

$$\begin{aligned}
\text{FIFO} &= \sum_{i\in[n]} g(F_i) = \sum_{i\in[n]} f(s, v_{x,i}) g(F_i) \\
&= \sum_{i\in[n]} \sum_{j\in[n]} f(v_{x,i}, v_{y,j}) g(F_i) \qquad [f \text{ is conserved at } v_{x,i}] \\
&\leq \sum_{i\in[n]} \sum_{j\in[n]} f(v_{x,i}, v_{y,j}) g(F_j^*) \qquad [v_{x,i}, v_{y,j} \in E \text{ only if } F_i \leq F_j^*] \\
&\leq \sum_{j\in[n]} \frac{4}{\epsilon^2} g(F_j^*) \qquad [f \text{ is conserved at } v_{y,j} \text{ and the capacity of } v_{y,j}t \text{ is } \tfrac{4}{\epsilon^2}] \\
&= \frac{4}{\epsilon^2} \text{OPT}
\end{aligned}$$

□

Thus it only remains to prove Lemma 76. Clearly the min-cut value is at most n, thus we focus on lower bounding the min-cut value. Let (S, T) be a minimum cut such that S contains the source s and T contains the sink t. To simplify the notation let $X_s = X \cap S$, $X_t = X \cap T$, $Y_s = Y \cap S$ and $Y_t = Y \cap T$. By definition each edge connecting s to a vertex in X_t is in (S, T) and the total capacity of these cut edges is $\sum_{v_{x,i}\in X_t} 1$. Knowing that each edge from a vertex in Y_s to t is in (S, T), it suffices to show that

$$\sum_{v_{y,j}\in Y_s} \frac{4}{\epsilon^2} \geq \sum_{v_{x,i}\in X_s} 1. \tag{7.7}$$

As in the proof of Lemma 66, Y_s is a subset of the out neighborhood of the vertices in X_s since the edges connecting vertices in X and Y have capacity ∞. We now show a lemma similar to Lemma 69

Lemma 78. *The vertices in X_s have at least $\frac{\epsilon^2}{4}|X_s|$ neighbors in Y, i.e. $|N(X_s) \cap Y| \geq \frac{\epsilon^2}{4}|X_s|$.*

Proof. Consider a maximal time interval I where FIFO is always busy scheduling jobs. Let J_k be the job (if exists) in X_s that has arrived the earliest (thus has

highest priority in FIFO) out of all the jobs in X_s scheduled during I. Let $C(I)$ be the jobs in X_s scheduled by FIFO during I. We will show that $v_{x,k}$ has at least $\frac{\epsilon^2}{4}|C(I)|$ neighbors in Y such that for each such neighbor $v_{y,j}$, FIFO completed the corresponding job J_j during the interval I. By taking a union over all possible intervals I, we will have that the neighborhood of X_s has size at least $\frac{\epsilon^2}{4}|X_s|$.

Notice that FIFO does a $(1+\epsilon)(|I|-F_k)$ volume of work during $I \setminus [r_k, C_k]$ since FIFO is given $(1+\epsilon)$ speed and is busy during this interval. Knowing that jobs are unit sized, FIFO completes at most $\lfloor (1+\epsilon)(|I|-F_k) \rfloor$ jobs during $I \setminus [r_k, C_k]$. The job J_k is the only job in $C(I)$ scheduled during $[r_k, C_k]$ because J_k has the highest priority in FIFO's schedule of the jobs in $C(I)$. This implies that $|C(I)| \leq \lfloor (1+\epsilon)(|I|-F_k) \rfloor + 1$. FIFO completes a volume of $(1+\epsilon)|I|$ work during I. Further, every job FIFO completes during I arrived during I since FIFO was not busy before I and FIFO scheduled these jobs during I. Let $e(I)$ denote the ending time point of I and $\mathcal{J}_{\text{FIFO}}(I)$ be the jobs completed by FIFO during I. The previous argument implies that at least a $(1+\epsilon)|I| - |I| - F_k = \epsilon|I| - F_k$ volume of work corresponding to jobs in $\mathcal{J}_{\text{FIFO}}(I)$ remains in OPT's queue at time $e(I) + F_k$ since OPT has 1 speed. If $\epsilon|I| - F_k$ is integral then at least $\epsilon|I| - F_k + 1$ jobs in $\mathcal{J}_{\text{FIFO}}(I)$ have flow time at least F_k in OPT's schedule; here there is one job that could be completed exactly at time $e(I) + F_k$ that is counted. Otherwise, OPT has $\lceil \epsilon|I| - F_k \rceil = \lfloor \epsilon|I| - F_k \rfloor + 1$ jobs in $\mathcal{J}_{\text{FIFO}}(I)$ that have flow time at least F_k. In either case, at least $\lfloor \epsilon|I| - F_k \rfloor + 1$ jobs in $\mathcal{J}_{\text{FIFO}}(I)$ have flow time at least F_k in OPT's schedule.

First consider the case where $F_k \leq \frac{\epsilon}{2}|I|$. In this case at least $\lfloor \epsilon|I| - F_k \rfloor + 1 \geq \lfloor \frac{\epsilon}{2}|I| \rfloor + 1$ jobs wait at least F_k time in OPT. Knowing that $|C(I)| \leq \lfloor (1+\epsilon)(|I| - F_k) \rfloor + 1 \leq \lfloor (1+\epsilon)|I| \rfloor + 1$, the neighborhood of $v_{x,k}$ contains at least $\lfloor \frac{\epsilon}{2}|I| \rfloor + 1 \geq \frac{\epsilon}{2(1+\epsilon)} \lfloor (1+\epsilon)|I| \rfloor - \frac{\epsilon}{2(1+\epsilon)} + 1 \geq \frac{\epsilon}{2(1+\epsilon)} (\lfloor (1+\epsilon)|I| \rfloor + 1) \geq \frac{\epsilon}{2(1+\epsilon)}|C(I)| \geq \frac{\epsilon^2}{2}|C(I)|$ nodes. The last inequality follows from $\epsilon < 1/2$.

Let us consider the other case that $F_k > \frac{\epsilon}{2}|I|$. Let t^* be the earliest time before C_k such that FIFO only schedules jobs that arrived no later than r_k during $[t^*, C_k]$. Equivalently, t^* is the beginning of the interval I by definition of FIFO. Notice

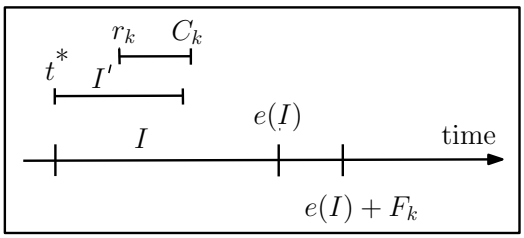


Figure 7.3: The interval I and I' in FIFO's schedule.

that $t^* \leq r_k$. Let $I' = [t^*, C_k]$. We know that FIFO completes a $(1+\epsilon)|I'|$ volume of work during $|I'|$. Let $\mathcal{J}_{\text{FIFO}}(I')$ denote the jobs completed by FIFO during I'. Any job in $\mathcal{J}_{\text{FIFO}}(I')$ arrives after t^* because FIFO was not scheduling a job before t^* by definition of I'. Note that any job in $\mathcal{J}_{\text{FIFO}}(I')$ will have flow time at least F_k if it is not satisfied until time C_k, since the jobs arrived no later than r_k. See Figure 7.3. Therefore, at least a $(1+\epsilon)|I'| - |I'| = \epsilon|I'| \geq \epsilon F_k > \frac{\epsilon^2}{2}|I|$ volume work corresponding to jobs in $\mathcal{J}_{\text{FIFO}}(I')$ remains unsatisfied in OPT's schedule at time C_k because OPT has unit speed and it was assumed that $F_k > \frac{\epsilon}{2}|I|$. Thus at least $\lceil \frac{\epsilon^2}{2}|I| \rceil$ jobs in $\mathcal{J}_{\text{FIFO}}(I')$ have flow time at least F_k in OPT. We also know that $|C(I)| \leq \lfloor (1+\epsilon)(|I| - F_k) \rfloor + 1 \leq (1+\epsilon)|I|$ since $F_k \geq \frac{1}{1+\epsilon}$. Together this shows that $v_{x,k}$ has at least $\frac{\epsilon^2}{2(1+\epsilon)}|C(I)| \geq \frac{\epsilon^2}{4}|C(I)|$ neighbors in Y knowing that $\epsilon \leq \frac{1}{2}$. □

Using Lemma 78 we cam complete the proof of Lemma 76. By Lemma 78 we have $|X_s| \leq \frac{4}{\epsilon^2}|N(X_s) \cap Y| \leq \frac{4}{\epsilon^2}|Y_s|$ which implies (7.7) and Lemma 76.

7.5 Analysis of WLAPS for Concave Functions

In this section we consider the objective function $\sum_{i \in [n]} w_i g(F_i)$ where w_i is a positive weight corresponding to job J_i and $g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a twice differentiable, non-decreasing, concave function. We let g' and g'' denote the derivative of g and the second derivative function of g. For this objective we will show a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$ competitive algorithm that is *non-clairvoyant*. The algorithm we consider is a generalization of the algorithm WLAPS [40, 46, 37].

Consider any job sequence σ and let $0 < \epsilon \leq 1/3$ be fixed. Without loss of

generality it is assumed that each job has a distinct arrival time. We assume that WLAPS is given $(1+3\epsilon)$-speed. At any time t, let $A(t)$ denote the jobs in WLAPS's queue. The algorithm at each time t finds the set of the most recently arriving jobs $A'(t) \subseteq A(t)$ such that $\sum_{J_i \in A'(t)} w_i g'(t-r_i) = \epsilon \sum_{J_i \in A(t)} w_i g'(t-r_i)$. The algorithm WLAPS distributes the processing power amongst the jobs in $A'(t)$ according to their current increase in the objective. That is $J_j \in A'(t)$ receives processing power $(1+3\epsilon) w_j g'(t-r_j) / (\epsilon \sum_{J_i \in A'(t)} w_i g'(t-r_i))$.

In case where there does not exist such a set $A'(t)$ such that the sum of $w_i g'(t-r_i)$ over all jobs $A'(t)$ is exactly $\epsilon \sum_{J_i \in A(t)} w_i g'(t-r_i)$, we make the following small change. Let $A'(t)$ be the smallest set of the most recently arriving jobs such that the sum of $w_i g'(t-r_i)$ over all jobs in $A'(t)$ is no smaller than $\epsilon \sum_{J_i \in A(t)} w_i g'(t-r_i)$. We let the job J_k, which has arrives the earliest in $A'(t)$, to receive processing power $\sum_{J_i \in A'(t)} w_i g'(t-r_i) - \epsilon \sum_{J_i \in A(t)} w_i g'(t-r_i)$. For simplicity, throughout the analysis, we will assume that there exists such a set $A'(t)$ of the most recently arriving jobs such that $\sum_{J_i \in A'(t)} w_i g'(t-r_i) = \epsilon \sum_{J_i \in A(t)} w_i g'(t-r_i)$. This is done to make the analysis more readable and the main ideas transparent.

To prove the competitiveness of WLAPS we define the following potential function. For a survey on potential functions for scheduling problems see [50]. For a job J_i let $p_i^A(t)$ be the remaining size of job J_i in WLAPS schedule at time t and let $p_i^O(t)$ be the remaining size of job J_i in OPT's schedule at time t. Let $z_i(t) = \max\{p_i^A(t) - p_i^O(t), 0\}$. The potential function is,

$$\Phi(t) = \frac{1}{\epsilon} \sum_{J_i \in A(t)} w_i g'(t-a_i) \sum_{J_j \in A(t), r_j \geq r_i} z_j(t).$$

We will look into non-continuous changes of $\Phi(t)$ that occur due to job arrivals and completions, and continuous changes of $\Phi(t)$ that occur due to WLAPS's processing, OPT's processing and time elapse. We will aggregate all these changes later.

Job Arrival: Consider when job J_k arrives at time t. There the change in the

potential function is $\frac{1}{\epsilon} w_k g'(0) z_k(r_k) + \frac{1}{\epsilon} \sum_{J_i \in A(t)} w_i g'(t - a_i) z_k(r_k)$. When job J_i arrives $z_i(r_i) = 0$, so there is no change in the potential function.

Job Completion: The optimal solution completing a job has no effect on the potential function. When the algorithm completes a job J_i at time t, some terms may disappear from $\Phi(t)$. It can only decrease the potential function, since all terms in $\Phi(t)$ are non-negative.

Continuous Change: We now consider the continuous changes in the potential function at time t. These include changes due to time elapse and changes in the z variable due to OPT and WLAPS's processing of jobs. First consider the change in due to time. This is equal to

$$\frac{\mathrm{d}}{\mathrm{dt}} \Phi(t) = \frac{1}{\epsilon} \sum_{J_i \in A(t)} w_i g''(t - a_i) \sum_{J_j \in A(t), r_j \geq r_i} z_j(t)$$

We know that w_i and $z_i(t)$ are positive for all jobs $J_i \in A(t)$. Further g'' is always non-positive since g is concave. Therefore, time changing can only decrease the potential.

Now consider the change due to OPT's processing. It can be seen that the most OPT can increase the potential function is to work exclusively on the job which has the latest arrival time. In this case, for any job $J_i \in A(t)$ the variable $\sum_{J_j \in A(t), r_j \geq r_i} z_j(t)$ changes at rate 1 because OPT has 1 speed. The increase in the potential due to OPT's processing is at most

$$\frac{\mathrm{d}}{\mathrm{dt}} \Phi(t) \leq \frac{1}{\epsilon} \sum_{J_i \in A(t)} w_i g'(t - a_i)$$

Now consider the change in the potential function due to the algorithm's processing. The algorithm decreases the z variable and therefore can only decrease the potential function. Recall that a job $J_j \in A'(t)$ is processed by WLAPS at a rate of $(1 + 3\epsilon) w_j g'(t - r_j) / (\sum_{J_i \in A'(t)} w_i g'(t - r_i))$ because WLAPS is given $(1 + 3\epsilon)$-speed. Therefore, for each job $J_j \in A'(t) \setminus O(t)$ the variable z_j decrease

at a rate of $(1+3\epsilon)w_j g'(t-r_j)/(\sum_{J_i \in A'(t)} w_i g'(t-r_i))$. Hence we can bound the change in the potential as,

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) &\leq -\frac{1}{\epsilon} \sum_{J_i \in A(t) \setminus A'(t)} w_i g'(t-a_i) \sum_{J_j \in A'(t) \setminus O(t)} \frac{(1+3\epsilon) w_j g'(t-r_j)}{\sum_{J_k \in A'(t)} w_k g'(t-r_k)} \\
&\leq -\frac{1-\epsilon}{\epsilon} \sum_{J_i \in A(t)} w_i g'(t-a_i) \sum_{J_j \in A'(t) \setminus O(t)} \frac{(1+3\epsilon) w_j g'(t-r_j)}{\sum_{J_k \in A'(t)} w_k g'(t-r_k)} \\
&\quad \text{[By definition of } A'(t)\text{]} \\
&= -\frac{1-\epsilon}{\epsilon^2} \sum_{J_j \in A'(t) \setminus O(t)} (1+3\epsilon) w_j g'(t-r_j) \\
&\leq -\frac{1+\epsilon}{\epsilon^2} \sum_{J_j \in A'(t)} w_j g'(t-r_j) + \frac{2}{\epsilon^2} \sum_{J_j \in O(t)} w_j g'(t-r_j) \\
&\quad \text{[Since } 0 < \epsilon \leq 1/3\text{]} \\
&\leq -\frac{1+\epsilon}{\epsilon} \sum_{J_j \in A(t)} w_j g'(t-r_j) + \frac{2}{\epsilon^2} \sum_{J_j \in O(t)} w_j g'(t-r_j) \\
&\quad \text{[By definition of } A'(t)\text{]}
\end{aligned}$$

By combining the changes due to OPT and the algorithm's processing and the change due to time, the continuous change in the potential function is at most,

$$\begin{aligned}
&\frac{1}{\epsilon} \sum_{J_i \in A(t)} w_i g'(t-a_i) - \frac{1+\epsilon}{\epsilon} \sum_{J_j \in A(t)} w_j g'(t-r_j) + \frac{2}{\epsilon^2} \sum_{J_j \in O(t)} w_j g'(t-r_j) \\
= &- \sum_{J_j \in A(t)} w_j g'(t-r_j) + \frac{2}{\epsilon^2} \sum_{J_j \in O(t)} w_j g'(t-r_j)
\end{aligned}$$

Completing the Analysis: At this point we are ready to complete the analysis. We know that $\Phi(0) = \Phi(\infty) = 0$ by definition of Φ, which implies that total sum of non-continuous changes and continuous changes of $\Phi(t)$ is 0. Further there are no increases in Φ for non-continuous changes. Hence we have $\int_{t=0}^{\infty} \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \geq 0$. Let WLAPS denote the algorithm's final objective and OPT denote the optimal

solution's final objective. Let $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) = \sum_{J_j \in A(t)} w_j g'(t - r_j)$ denote the increase in WLAPS objective at time t and let $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) = \sum_{J_j \in O(t)} w_j g'(t - r_j)$ denote the increase in OPT's objective at time t. We have that,

$$\begin{aligned}
\mathrm{WLAPS} &= \int_{t=0}^{\infty} \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) \\
&\leq \int_{t=0}^{\infty} \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \\
&\leq \int_{t=0}^{\infty} \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) - \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) + \frac{2}{\epsilon^2}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) \\
&\leq \frac{2}{\epsilon^2}\mathrm{OPT}
\end{aligned}$$

This proves the following theorem.

Theorem 79. *The algorithm* WLAPS *is* $(1+\epsilon)$*-speed* $O(\frac{1}{\epsilon^2})$*-competitive for minimizing* $\sum_{i \in [n]} w_i g(F_i)$ *when* $g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ *is a concave nondecreasing positive function that is twice differentiable.*

7.6 Conclusions and Discussions

One obvious question is if there exists an online algorithm that is $O(1)$-competitive with speed less than two or not. To obtain such an algorithm (if it exists), one must exploit the structure of cost functions. We have preliminary evidence that our analysis can be extended to show that HDF is $O(1)$-speed $O(1)$-competitive on identical parallel machines.

Chapter 8

Conclusions and Future Directions

This thesis has considered several scheduling problems in the online setting. The main focus of this thesis was the design of algorithms in the broadcast scheduling model. We showed online scalable algorithms for average flow time, ℓ_k-norms of flow time and the delay factor. Further, we gave strong guarantees for LWF and its variants for a variety of metrics. We believe that the algorithms and analyses given in this thesis has lead to better understanding the broadcast scheduling setting, especially online. Further, it has given insights into the types of algorithms and algorithmic ideas that perform well in the broadcast setting. Most of the work given in this thesis is of theoretical interest, however we believe that the underlying ideas have the potential to be very useful in practice. There are several open problems left in the broadcast model, many of which were mentioned throughout this thesis. Here we point out two problems that would be quite interesting to resolve.

One of the interesting things to note about the algorithms given in this thesis that are scalable in the broadcast setting is that *all* of them have knowledge of the speed given to the algorithm. That is, the algorithms are $(1+\epsilon)$-speed $O(1)$-competitive, but the algorithm uses the value of ϵ to make scheduling decisions. This is a somewhat surprising phenomenon, which as far as the author knows, has only been observed recently in the last four years. Besides the broadcast setting, this algorithmic property has also been seen in the scalable algorithms given for average flow time and the ℓ_k-norms of flow time in the arbitrary speed of curves scheduling setting [40, 46, 37]. As mentioned in this thesis, the arbitrary speed up curves setting is closely related to the broadcast setting.

An interesting question that emerges is whether or not their exists an algorithm that is $(1+\epsilon)$-speed $O(1)$-competitive for any of these problem (say average flow time in the broadcast setting) without knowledge of ϵ. An algorithm that is scalable without knowledge of ϵ has been called *universally* scalable. A universally scalable algorithm is arguably more natural because the algorithm does not need the parameter ϵ when implementing the algorithm. In practice, the 'right' value of ϵ would have to be determined. From a theoretical viewpoint, it would be interesting to determine whether the knowledge of ϵ somehow gives additional power to the algorithm. In particular, it would be quite interesting if for some problems no algorithm can be $(1+\epsilon)$-speed $O(1)$-competitive for all ϵ without knowledge of ϵ. However, no similar result is known in scheduling theory and it is not clear how one could show that this is the case.

Perhaps the most interesting question that remains open in the broadcast setting is whether or not there exists an $O(1)$-approximation algorithm for average flow time when pages have unit sizes. A central open problem in scheduling theory is whether or not there exists an $O(1)$-approximation algorithm for average weighted flow time on a single machine in the standard setting. Of course, average flow time in the broadcast setting is more general than average weighted flow time in the standard model, so if one were to consider the setting where pages have varying sizes it would be prudent to consider the standard setting first. However, in the standard setting, if jobs have unit size there exists an optimal algorithm for average weighted flow time. Thus, it is interesting to consider designing a constant approximate algorithm for average flow time in broadcast setting when pages have unit size. Currently the best known algorithm has an approximation ratio of $O(\log^2 n/\log\log(n))$ which was shown in [7].

One of the other contributions of the work given in this thesis is the introduction of the general cost function scheduling metric. In this thesis, we showed that somewhat surprisingly the algorithm HDF is $(2+\epsilon)$-speed $O(1/\epsilon)$-competitive for this very general scheduling setting. Several questions arise out of this work. First is whether or not an algorithm can be constant competitive with speed less than 2.

We showed in this thesis that this cannot be the case for an algorithm that does not take the cost function into consideration when making scheduling decisions. We also showed that no algorithm can be scalable in this setting. However, it would be interesting to break the barrier of 2 and show that some algorithm can be constant competitive with say 1.9-speed. The challenge is that our lower bound shows that this algorithm cannot be HDF and therefore new algorithmic techniques will need to be developed to approach this problem. Another interesting question is to consider the general cost function objective in more general scheduling settings. For instance, in the multiple machine scheduling setting. We have preliminary evidence that the analysis we gave for HDF cane be extended to the identical machines setting to show a $O(1)$-speed $O(1)$-competitive algorithm. However, the speed required is larger than that used in the single machine setting. It would be interesting if one could show an algorithm in the identical machines setting whose guarantees are similar to that of what we showed for HDF in the single machine setting.

References

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

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

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

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

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

[6] Nikhil Bansal, Moses Charikar, Sanjeev Khanna, and Joseph (Seffi) Naor. Approximating the average response time in broadcast scheduling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 215–221, 2005.

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

[8] Nikhil Bansal, Ravishankar Krishnaswamy, and Viswanath Nagarajan. Better scalable algorithms for broadcast scheduling. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2010.

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

[10] Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 407–414, 2010.

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

[12] Cynthia Barnhart, Dimitris Bertsimas, Constantine Caramanis, and Douglas Fearing. An improved analysis of longest wait first. In *Equitable and Efficient Coordination in Traffic Flow Management*, 2010.

[13] Yair Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 558–559, 2000.

[14] Erhan Bayraktar and Savas Dayanik. Poisson disorder problem with exponential penalty for delay. *Math. Oper. Res.*, 31(2):217–233, 2006.

[15] 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.

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

[17] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 270–279, 1998.

[18] Michael A. Bender, Raphaël Clifford, and Kostas Tsichlas. Scheduling algorithms for procrastinators. *J. Scheduling*, 11(2):95–104, 2008.

[19] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Improved algorithms for stretch scheduling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 762–771, 2002.

[20] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Approximation algorithms for average stretch scheduling. *J. Scheduling*, 7(3):195–222, 2004.

[21] Allan Borodin and Ran El-Yaniv. On ranomization in online computation. In *IEEE Conference on Computational Complexity*, pages 226–238, 1997.

[22] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[23] A. Burns and S. Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008.

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

[25] Wun-Tat Chan, Tak Wah Lam, Hing-Fung Ting, and Prudence W. H. Wong. New results on on-demand broadcasting with deadline via job scheduling with cancellation. In *International Computing and Combinatorics Conference (COCOON)*, pages 210–218, 2004.

[26] Jessica Chang, Thomas Erlebach, Renars Gailis, and Samir Khuller. Broadcast scheduling: algorithms and complexity. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 473–482, 2008.

[27] Moses Charikar and Samir Khuller. A robust maximum completion time measure for scheduling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 324–333, 2006.

[28] Chandra Chekuri, Avigdor Gal, Sungjin Im, Samir Khuller, Jian Li, Richard McCutchen, Benjamin Moseley, and Louiqa Raschid. New models and algorithms for throughput maximization in broadcast scheduling. In *Workshop on Approximation and Online Algorithms (WAOA)*, 2010.

[29] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multiprocessor scheduling to minimize flow time with epsilon resource augmentation. In *ACM Symposium on Theory of Computing (STOC)*, pages 363–372, 2004.

[30] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Longest wait first for broadcast scheduling. In *Workshop on Approximation and Online Algorithms (WAOA)*, 2009.

[31] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Minimizing maximum response time and delay factor in broadcasting scheduling. In *European Symposium on Algorithms (ESA)*, pages 444–455, 2009.

[32] Chandra Chekuri and Benjamin Moseley. Online scheduling to minimize the maximum delay factor. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1116–1125, Philadelphia, PA, USA, 2009.

[33] Marek Chrobak, Leah Epstein, John Noga, Jiri Sgall, Rob van Stee, Tomás Tichý, and Nodari Vakhania. Preemptive scheduling in overloaded systems. *J. Comput. Syst. Sci.*, 67(1):183–197, 2003.

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

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

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

[37] Jeff Edmonds, Sungjin Im, and Benjamin Moseley. Online scalable scheduling for the ℓ_k-norms of flow time without conservation of work. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011.

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

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

[40] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 685–692, Philadelphia, PA, USA, 2009.

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

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

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

[44] Naveen Garg and Amit Kumar. Better algorithms for minimizing average flow-time on related machines. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 181–190, 2006.

[45] Naveen Garg, Amit Kumar, and V. N. Muralidhara. Minimizing total flow-time: The unrelated case. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 424–435, 2008.

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

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

[48] Sungjin Im and Benjamin Moseley. An online scalable algorithm for average flow time in broadcast scheduling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[49] Sungjin Im and Benjamin Moseley. An online scalable algorithm for minimizing ℓ_k-norms of weighted flow time on unrelated machines. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011.

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

[51] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Online scheduling with general cost functions. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1254–1265, 2012.

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

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

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

[55] Insup Lee, Joseph Y-T. Leung, and Sang Son, editors. *Handbook of Real-Time and Embedded Systems*. CRC Press, 2007.

[56] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *J. Comput. Syst. Sci.*, 73(6):875–891, 2007.

[57] Joseph Leung, Laurie Kelly, and James H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., Boca Raton, FL, USA, 2004.

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

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

[60] Kirk Pruhs, Jiri Sgall, and Eric Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. 2004.

[61] Kirk Pruhs and Patchrawat Uthaisombut. A comparison of multicast pull models. *Algorithmica*, 42(3-4):289–307, 2005.

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

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

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

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

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

[67] Feifeng Zheng, Stanley P. Y. Fung, Wun-Tat Chan, Francis Y. L. Chin, Chung Keung Poon, and Prudence W. H. Wong. Improved on-line broadcast scheduling with deadlines. In *International Computing and Combinatorics Conference (COCOON)*, pages 320–329, 2006.