ONLINE SCHEDULING ALGORITHMS FOR AVERAGE FLOW TIME AND ITS VARIANTS

BY

SUNGJIN IM

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
Professor Jeff Erickson
Professor Nitin Vaidya
Associate Professor Nikhil Bansal, Eindhoven University of Technology

Abstract

This dissertation focuses on scheduling problems that are found in a client-server setting where multiple clients and one server (or multiple servers) are the participating entities. Clients send their requests to the server(s) over time, and the server needs to satisfy the requests using its resources. This setting is prevalent in many applications including multiuser operating systems, web servers, database servers, and so on.

A natural objective for each client is to minimize the flow time (or equivalently response time) of her request, which is defined as its completion time minus its release time. The server, with multiple requests to serve in its queue, has to prioritize the requests for scheduling. Inherently, the server needs a global scheduling objective to optimize. We mainly study the scheduling objective of minimizing ℓ_k-norms of flow time of all requests, where $1 \leq k < \infty$. These objectives can be used to balance average performance and fairness.

A popular performance measure for *online* scheduling algorithms is *competitive ratio*. An algorithm is said to be c-competitive if its objective is within a multiplicative factor c of the optimal scheduler's objective for any sequence of requests. Roughly speaking, an algorithm with a small competitive ratio performs well compared to the optimal scheduler even on a worst case input. However, for some problems, competitive ratio could be large for any online algorithm. In such cases, a popular relaxation is resource augmentation where the algorithm is run on a faster machine and compared to the optimal scheduler with one speed. In particular, a scheduling algorithm is said to be scalable if it has a small competitive ratio with any amount of speed augmentation. For problems that have a large lower bound on the achievable competitive ratio, a scalable algorithm is essentially the best one can hope for, in the worst case analysis framework.

We give the *first* scalable algorithms in several scheduling settings for the ℓ_1 norm or ℓ_k norms of flow time ($k \geq 2$). These settings include broadcast scheduling, scheduling jobs of different parallelizability, and scheduling on heterogeneous machines, and are described below:

- *Broadcast scheduling.* There is a server that stores pages that contain useful data. Each request arrives over time asking for a specific page. When the server broadcasts a page p, all outstanding requests for the same page are satisfied simultaneously. This is the main difference from standard

scheduling settings where the server must process each request separately. The broadcast model is motivated by several applications such as multicast systems and wireless and LAN networks.

- *Scheduling jobs of different parallelizability.* In this model, jobs have varying degrees of parallelizability (that is, some jobs may be sped up considerably when simultaneously run on multiple processors, while other jobs may be sped up by very little) on a multiprocessor system. The most obvious settings where this problem arises are scheduling multi-threaded processes on a chip with multiple cores/processors, and scheduling multi-process applications in a server farm.

- *Scheduling on heterogeneous machines.* In this dissertation, two cases are mainly considered: related machines and unrelated machines. In the related machines setting, machines may have different speeds. In the more general unrelated machines setting, jobs may have completely different processing times depending on the machines they are assigned to.

In all the above models, the online algorithm and the optimal scheduler may have to do different amount of work to complete the same set of requests. This makes it challenging to design and analyze scheduling algorithms. The results presented in this dissertation are enabled by the development of novel analysis techniques.

This dissertation is dedicated to my parents for their endless love and support

Acknowledgments

I am deeply indebted to my Ph.D. advisor, Chandra Chekuri, for his guidance, support and encouragement throughout my Ph.D. research. He always helped me with invaluable and timely advice and continuously fostered my growth as a researcher. In particular, I am very grateful for his patience with me as a young graduate student. His high standards and passion for research and teaching will always be a great inspiration to me.

Thanks to Kirk Pruhs for inviting me many times to collaborate in Pittsburgh. During my visits, each day was full of stimulating ideas. I will never forget the fun of wrestling with a problem all day long while there, enjoying discussions with intelligent people. It was also great fun brewing beer, particularly outside in wintertime.

I wish to thank my formal and informal mentors during my summer internship. I thank Yajun Wang for his generous time sharing at MSRA. I thank Nikhil Bansal for inviting me to intern for a summer at IBM T. J. Watson in 2011. Although I had only a short overlap with him before he moved to the Netherlands, discussing my research with him was always a great inspiration. My formal mentor there, Viswanath Nagarajan, showed me how energetic one can be in his approach to research. Additionally, Maxim Sviridenko, now at the University of Warwick, taught me the importance of rigorous thinking and was always fun to talk to.

I would like to sincerely thank Jeff Erickson and Nitin Vaidya for serving on my prelim/dissertation committee. They provided invaluable comments on my research.

My lab mate, Ben Moseley, has earned my special thanks and appreciation. We collaborated on many research problems. Whenever I came to office, I was always excited by the fact that I could chat with him for hours on research problems. Many findings in this dissertation would not have been possible without him.

I also wish to thank former and current theory folks at UIUC for making my Ph.D. studies more enjoyable. They are brilliant and often provided a pleasant distraction during the busy work day.

Finally, I would like to thank my parents for their endless support, love, trust and commitment. I am extremely fortunate to have my mother and father as my parents. I also want to thank my older brother Sunghoon. I know how much he cares for me, and I deeply appreciate that. I also wish to thank my

fiancee, Myra, for the times we have cherished together at UIUC and for the life we will make together forever.

The research in this dissertation has been supported mainly by a Samsung Fellowship and in part by NSF grants CCF-0728782, CNS-0721899, and CCF-1016684.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Scheduling jobs is a fundamental problem that arises in numerous forms and in various situations. Each job, for example, can be a unit of work that arrives for service at a computer system or can be of an abstract form that needs to be done by human power. Scheduling problems can, in an abstract way, be described as assigning limited resources to jobs over time to satisfy or optimize a certain objective. Due to its wide appearance in practice, scheduling has been an important field in many disciplines such as operations research and computer science. A comprehensive overview on the topic of scheduling can be found in [79].

It was in early 1950's that operations research and management science initiated the study of scheduling problems. The motivation stemmed mainly from production planning in manufacturing process. In early 1960's, computer scientists added another angle when they designed scheduling algorithms for operating systems to expedite program executions. It was a timely research endeavor particularly due to insufficient and expensive computer resources such as CPU and memory. Consequently, computer science has become one of the main disciplines that led the research on scheduling algorithms, finding applications in compilers, parallel computing, operating systems, databases and so on. This thesis focuses on the scheduling problems arising in computer science, particularly in the client-server setting. In this setting, there are multiple clients and one server (or multiple servers). Clients submit their requests to the server(s) over time, and the server needs to satisfy the requests. This setting is prevalent in many applications including multiuser operating systems, web servers, database servers, and name servers.

Perhaps the most obvious objective for each client is to minimize the flow time (or equivalently the response time) of her request, which is defined as its completion time minus its release time. That is, each client wants her request to be scheduled as early as possible. The server, with multiple requests to serve in its queue, has to make a scheduling decision of which request to schedule first. Inherently, the server needs a global objective. One of the most popular objectives is to minimize the average (or equivalently total) flow time. However, because focusing on minimizing the average flow time may let some jobs starve for an unacceptably long time, other objective functions that take fairness under consideration may be preferred in some situations. In general, we will study the

scheduling objectives of minimizing ℓ_k-norms of flow time of all requests, where $1 \leq k < \infty$; usually k is assumed to be an integer. These objectives can be used to make a natural balance between average performance and fairness; more weight is put on fairness as k grows. The values $k = 1, 2$ and ∞ are of practical interest.

Since scheduling problems can be viewed as optimization problems, the history of scheduling research particularly in theoretical computer science is tied closely to the development of algorithms and computational complexity in the broad context of combinatorial optimization. In early days, scheduling problems were relatively simple and the focus was mainly to develop optimal algorithms. Since Karp's seminal work on NP-hardness, many scheduling problems were shown to be NP-hard, which implies that no efficient (polynomial-time) algorithms are likely to exist for such problems. This had shifted a substantial amount of research to approximation algorithms, seeking efficient algorithms that yield a solution that is as close to the optimum as possible [1]. Such a quality can be formally measured by the approximation ratio (or factor), which is defined roughly as the worst ratio between the algorithm's objective and the optimal scheduler's objective. Consequently, many classical and important approximate scheduling algorithms were developed. For more details, the reader is referred to several informative surveys [72, 77, 61].

It would be worth comparing such algorithms with theoretical approximation guarantees to heuristics that are implemented and deployed in practice. Heuristics are often observed to perform fairly well for most cases of input. However, they do not provide a guaranteed performance, i.e., for some input, their performance could be substantially far from the optimum. The approximation factors that are theoretically proven are often too large to be embraced in practice. Still, such an analysis is of fundamental and practical interest for the following reasons. In addition to the guaranteed performance aforementioned, its performance is often in practice very close to the optimum. This is because pathological instances rarely occur in practice. Perhaps more importantly, the ideas that are used in designing provably-good approximation algorithms and in showing the hardness give invaluable insights into the problem, which can be readily tuned into practical heuristics. For an overview of the field of approximation algorithms, the reader is referred to [98, 101].

When the input has *online* nature, the problem becomes more challenging. In the online setting, the scheduler becomes aware of each job only when it is released. There are largely two different models depending on how much the scheduler becomes aware of a job's properties upon its arrival. A scheduler is said to be clairvoyant if it learns all the properties of each job, in particular its size, upon its arrival. In contrast, a non-clairvoyant algorithm knows only when a job is released and completed. In either case, the online scheduler does not have the knowledge of jobs arriving in the future. The absence of knowledge

[1] Interestingly, Graham's seminal work on the list scheduling algorithm precedes the introduction of NP-hardness and is believed to be the first approximation algorithm.

of future jobs, possibly in addition to computational hardness, clearly makes the problem more difficult in the online setting.

Competitive ratio is a popular quantity that measures the worse case performance of an online algorithm. For an objective to be minimized (or maximized), an algorithm is said to be c-competitive if for any job instance, its objective is at most c (or at least $1/c$) times the optimal *offline* scheduler's objective. It is important to note that the online algorithm is compared against the optimal offline scheduler that knows all jobs along with their properties at the beginning. Although online algorithms are not required to run in polynomial time in the definition of the competitive ratio, we will assume that they are in the scheduling context, since in practice usually simple and efficient scheduling policies are preferred. We will call this analysis model the *worst case analysis*. This model has a limitation in that it does not utilize any known distribution of the input. However, it also has a positive side: the provably-good algorithms in this analysis model are truly robust to all kinds of input. An extensive overview of online algorithms under the worst case analysis model can be found in [22]. For an overview of online scheduling algorithms, the reader is referred to [87, 85].

Unfortunately, for many problems, any online algorithm has an unacceptably large competitive ratio. This is because an online algorithm often inevitably makes non-optimal scheduling decisions without the knowledge of future jobs and such repeated non-optimal decisions finally result in a huge penalty. To remedy this limitation of the worst case analysis, a popular relaxation called resource augmentation was introduced by Kalayanasudaram and Pruhs [69]. In this relaxed model, the online algorithm is given extra speed to process jobs and compared to the optimal offline scheduler. In other words, the algorithm runs on a faster machine while the optimal scheduler runs on a 1-speed machine. Surprisingly, this slight speed augmentation dramatically improves the achievable competitive ratio. More importantly, this relaxation has been successful in separating good scheduling algorithms from poor ones. In particular, a scheduling algorithm is said to be *scalable* if it has a constant competitive ratio (depending only on ϵ) when given $(1+\epsilon)$-speed. A scalable algorithm is essentially the best result one can hope for in the worst case analysis framework/model if the problem has a strong lower bound on the achievable competitive ratio. This is because a scalable algorithm, with any small amount of speed augmentation, can be compared to the optimal scheduler. Scalable algorithms are often observed to perform well in practice.

In this dissertation, we study three different scheduling models, broadcast scheduling, scheduling jobs of different parallelizability, and scheduling on heterogeneous machines; we will discuss these models in Section 1.6. In each of these models, we give the first scalable algorithm for minimizing average flow time (ℓ_1 norm) or for ℓ_k norms of flow time, $k \geq 2$ [2].

[2] In this dissertation, for the easy of analysis, ℓ_k norms are considered only when k is a positive integer. Our results can be easily extend to real values $k \geq 1$.

1.1 Notation and Terminology

We first define basic notation that will be used throughout this thesis. The notation may be changed slightly for each of the scheduling models under consideration. For job i, let r_i denote its arrival time (or release time). The (initial) size of job i is denoted as p_i. In some problems, each job may have a weight w_i which stands for its importance. Throughout this dissertation, we consider the setting where all jobs must be completed. A job, which has arrived but is not completed, is said to be unsatisfied, alive, or outstanding. We may say that outstanding jobs are in the scheduler's queue. Thus any feasible schedule σ defines each job i's finish time, C_i^σ. The superscript σ may be dropped when it is clear from the context. We let OPT denote an optimal scheduling algorithm; of course, the "optimality" hinges on the objective function. When a scheduling algorithm (policy) ALG and an instance σ of jobs are given, we let ALG(σ) denote the schedule produced by ALG. For notional simplicity, we allow ALG(σ) also denote the value of the schedule for the given objective function. Therefore, OPT(σ) may denote an optimal schedule for σ or the optimal value itself depending on the context.

We now formally define a feasible schedule and a job's completion time. In a feasible schedule, at any time, a job can be processed on at most a single machine; the scheduling model, scheduling jobs of different parallelizability that will be covered in Chapter 3 is the only exception, and we defer the discussion on that model to Section 1.6.2. We first need to define preemption.

Preemptive vs. Non-preemptive: A *preemptive* scheduler can, without any penalty or delay, preempt a job being processed to work on other jobs and resume it later (from the point of the job where it was processed last). Some algorithms such as Round Robin (RR) may preempt a job infinitely many times, since at any instantaneous time, all outstanding jobs equally share the processor (such algorithms, in practice, are implemented by making preemptions occur frequently). In contrast, a non-preemptive scheduler must complete a job without interruption once the machine starts working on it. In most cases, preemption is necessary to obtain positive results, and we will be concerned with preemptive schedulers unless stated otherwise.

The completion time C_i of a job J_i is defined as the first time that it receives p_i amount of work since its arrival. To formally define the completion time C_i, preempted finite times, the completion time C_i is defined the first time τ such that $\sum_{l=1}^{h} s_h|T_h| \geq p_i$, where $\{T_l\}_{l=1}^{h}$ are the disjoint time intervals in $[r_i, C_i]$, and s_h is the speed that J_i is processed during T_l. The speed of the machine that J_i is processed on can be non-unit for the following reasons.

- Heterogeneous machines: In the related machines setting, machines may have different speeds, s_x. A job J_j of size p_j can be completed within p_i/s_x unit times on machine M_x with speed s_x. In the more general unrelated machines setting, each job J_i has a processing time p_{ix} on machine M_x it is

assigned to. An equivalent view is that J_i has a unit size, and is processed on machine x at a speed of $1/p_{ix}$.

- Speed scaling: Most modern machines have speed scaling capability that can dynamically change the speed of the machines. Machines can be sped up by consuming more power. The speed function $Q_x(P)$ of each machine M_x is given as an input and specifies the speed that machine x runs on when using power $P \geq 0$.

- Speed augmentation: as previously mentioned, in the relaxation called resource (or speed) augmentation, the online algorithm is given extra speed (or is run on a faster machine). If the algorithm is given speed s, a job is processed s times faster compared to when the algorithm is given just one speed.

When the number of preemptions of a job J_i is unbounded, it is more suitable to define its completion time based on the rate (or instantaneous speed) at which it is processed. For example, on a single machine, the algorithm RR lets all outstanding jobs get the equal share any instantaneous time. Let $N(t)$ be the number of outstanding jobs at time t under the schedule of RR. The job J_j is processed at a speed that is $1/N(t)$ times the speed that it is processed when it exclusively uses the whole processing; this is not the case in the scheduling model where jobs have different parallelizability, and we will discuss this further in Section 1.6.2 and Chapter 3. The completion time C_i is then defined as the first time when $\int_{t=r_i}^{C^i} s(t)\mathrm{dt} \geq p_i$, where $s(t)$ is the rate at which J_i is processed at time t.

We finally define clairvoyant and non-clairvoyant schedulers depending on how much information is revealed to the scheduler upon a job's arrival.

Clairvoyant vs. Non-clairvoyant: In some cases, the scheduler may not be aware of the actual size of a job before its completion. This is often the case for the scheduler for operating systems. Such a scheduler is said to be *non-clairvoyant* and is considered to be more suitable in certain settings. We will study non-clairvoyant scheduling algorithms when jobs have different parallelizability and when machines have non-uniform speeds, in Chapters 3 and 5, respectively. These models are described briefly in Sections 1.6.2 and 1.6.3, respectively. In other sections, we will be concerned with clairvoyant algorithms that are provided with all properties of each job upon its arrival, including its size and weight.

1.2 Objective Functions

The scheduling objective of choice may vary depending on the systems' priority. In this section, we discuss the objectives that will be studied in this dissertation. Throughout this dissertation, we are interested in the settings where the server is required to complete all jobs. For an individual job i, the most popular metric

is its flow time (or response time), $C_i - r_i$, which is the length of time between when the job i (will sometimes be denoted as J_i) is released at time r_i and when it completes at time C_i. To get a quality of service measure for the entire schedule, one must combine the quality of service measures of the individual jobs. The most commonly used way to do this is to take the average, or the sum (the ℓ_1 norm) of the flow times of the individual jobs. Formally, the total flow time (the ℓ_1-norm) is defined as $\sum_i (C_i - r_i)$. Its weighted version, that is the total weighted flow time is defined as $\sum_i w_i(C_i - r_i)$. Total flow time can be viewed as equivalent to average flow time assuming that all jobs must be completed which is the case throughout this thesis.

Unfortunately, algorithms which focus only on minimizing the total flow time may be substantially unfair to individual jobs by allowing some jobs to starve for an arbitrarily long time. Designing a scheduling algorithm that is fair to all jobs overall is important for practical scheduling algorithms [96]. Due to unfairness, competitive algorithms for average flow time are not often implemented [62, 8]. In practice, it is usually more desirable for the system to be predictable for each job than optimal on average [96, 91]. Indeed, Silberschatz and Galvin's classic text *Operating Systems Concepts* [91] states "A system with reasonable and predictable response time may be considered more desirable than a system that is faster on the average, but is highly variable." and " ... for interactive systems, it is more important to minimize the variance in the response time than it is to minimize the average response time."

Hence, in some settings, the ℓ_k-norm of flow time $\sqrt[k]{\sum_{i\in[n]}(C_i - r_i)^k}$ for some $k > 1$ may be a better service measure quality than the ℓ_1 norm of job flow times. In practice, $k \in \{1, 2, \infty\}$ is usually considered. Minimizing the ℓ_k norms of flow time was first introduced by the influential paper of Bansal and Pruhs [8]. The ℓ_k-norm of weighted flow time is defined as $\sqrt[k]{\sum_{i\in[n]} w_i(C_i - r_i)^k}$. In the ℓ_k norm objective ($k \geq 2$), the algorithm is severely penalized when a job waits a substantial amount of time to be satisfied. Further, in the ℓ_k-norm, the flow time is still being considered and the algorithm must also focus on average quality of service. By optimizing the ℓ_k norm of flow time, the algorithm balances average quality of service and fairness. This makes online algorithms that are competitive for the ℓ_k-norm of flow time desirable in practice. The following simple example illustrates the difference between the ℓ_1 norm and ℓ_2 norm.

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

in order of their release time results in the optimal schedule where all jobs have flow time at most 2. Thus a schedule that is good under the ℓ_2 norm reduces the variance of the job flow times relative to an optimal schedule for the ℓ_1 norm.

We also study objectives involving power minimization. Power minimization has been an increasingly important issue with the ever-growing size of data to be processed, which is illustrated by the following quote:

> What matters most to the computer designers at Google is not speed, but power, low power, because data centers can consume as much electricity as a city. — Eric Schmidt, CEO Google

Another important motivation for power minimization is to make increasingly available portable computing devices to last longer on limited battery power. The reader is referred to the informative survey in [68] for an overview of theoretical investigations on the topic of power minimization.

Speed scaling is one of the main technologies to save energy. Here each machine can run in various speeds, consuming different amount of power depending on the speed it runs in. Generally, the power grows super-linearly in the speed. Such a relationship can be expressed as a power function $P(s)$ that specifies the power the machine consumes when it runs at speed s. The most popular power function considered in the scheduling literature is $P(s) = s^\alpha$ for some $\alpha > 1$. Recently, a general power function P has received considerable attention that needs to satisfy only the following constraints [13, 56, 59] (its inverse function $Q = P^{-1}$ is often used.) :

- P is non-negative.
- P is continuous.
- P is differentiable on all but countably many points.
- There is a maximum allowable speed or that the limit inferior of $P(s)/s$ as s approaches infinity is positive.

In this dissertation, we focus on the objective of minimizing the total flow time plus the total energy consumption. In the offline setting a most appealing objective would be to minimize the total flow time within a certain energy budget. However, in the online setting where a infinite sequence of jobs arrive, a budget constraint cannot be imposed. Hence it would be reasonable to consider the following objective.

$$\lambda \text{ (total flow time)} + \text{(total energy consumption)}$$

This objective implies that one is willing to use an additional unit of energy to improve the overall system performance, measured by the total flow, by an

amount of λ. By scaling the power function $P(\cdot)$, one can assume without loss of generality that $\lambda = 1$. This objective will be considered in the heterogeneous machine setting in Chapter 5. One can consider the ℓ_k norms of flow time plus the total energy consumed, but it does not have a natural interpretation as the above. A variant objective of minimizing $\sum_{i\in[n]}(C_i - r_i)^k + (\text{total energy consumption})$ was considered in [58].

We note that there are serveral other metrics that are considered in the scheduling literature such as the maximum flow time (the ℓ_∞-norm of flow time) or the maximum throughput (the number of jobs that are completed by their deadline). This dissertation will focus primarily on the ℓ_k norm of (weighted) flow time when $1 \le k < \infty$, and on total flow time plus energy consumption.

1.3 Analysis Framework

Many scheduling problems are NP-hard as are other combinatorial optimization problems. Hence it is strongly believed that efficient (polynomial-time) optimal algorithms do not exist. Further, in the online setting where the online scheduler cannot see the future jobs, it may be the case that no online algorithm could be optimal. Given that scheduling algorithms should be efficient enough to yield schedules in timely manner, we will aim at obtaining efficient (polynomial-time) algorithms that yield solutions as close to the optimal solution as possible. In this section, we define approximation ratio (factor) and competitive ratio which are widely accepted measures of the performance of offline and online algorithms, respectively. Since this dissertation is focused on scheduling problems, we will give the definition in the language of scheduling. We also formally describe the popular relaxation called resource augmentation that is widely used in online scheduling.

Before defining our analysis framework, we first formally define scheduling problems. A scheduling problem consists of constraints and an objective. The input is a job instance σ. For example, consider the problem of minimizing average flow time on a single machine. The job instance σ will be given as a set of jobs J_i, $i \in [n]$ with release time r_i and size p_i. Any feasible schedule must satisfy the constraints that at any time at most one job can be processed on the machine and that all jobs must be completed. The objective is, needless to say, to minimize average flow time. In general, the scheduling objective is to be minimized or to be maximized, but to simplify our discussion, we assume that the objective is to be minimized. All objectives that are considered in this dissertation are as such. Scheduling problems can vary depending on the constraints and the objective in consideration.

1.3.1 Approximation Ratio

Recall that $\text{ALG}(\sigma)$ denotes the algorithm ALG's objective on the job instance σ. Likewise, $\text{OPT}(\sigma)$ denotes the optimal scheduler OPT's objective on the same

job instance σ. Approximation ratio is used to measure the worst case performance of an *offline* algorithm ALG relative to the optimal scheduler. Note that ALG has a full access to the entire input σ as the optimal solution does. Formally, we say that ALG is a c-approximation or equivalently that the approximation ratio (factor) of ALG is c if the following holds for any job instance σ.

$$\text{ALG}(\sigma) \leq \alpha \text{OPT}(\sigma)$$

Here the algorithm is required to run in polynomial time in the input size. For more formal definition of approximation ratio, we refer the reader to Appendix A in [98].

1.3.2 Competitive Ratio

Competitive ratio is a common measure that is used to gauge the performance of an online algorithm ALG. Recall that an online algorithm is not aware of a job until it arrives. In general all properties of a job, upon its arrival, is revealed to the online algorithm ALG (as mentioned earlier, a non-clairvoyant scheduler remains ignorant of a job's size even when it arrives). The competitive ratio is defined as the upper bound on the ratio of the cost of the algorithm's schedule to that of the *optimal offline* schedule on any instance. Formally, we say that an online algorithm ALG is c-competitive or equivalently that the competitive ratio of ALG is c if the follow holds for all job instances σ:

$$\text{ALG}(\sigma) \leq \alpha \text{OPT}(\sigma)$$

This analysis model is often called *the worst case analysis*, since the competitive ratio measures the worst performance of the online algorithm relative to the optimal scheduler. Note that this model even captures adaptive adversarial inputs. That is, at each time, the (imaginary) adversary can adaptively create a sequence of future jobs on the fly that is difficult for the algorithm or that makes the algorithm's previous decisions non-optimal when considered with the new jobs. Particularly when the online algorithm is deterministic, the adversary, which has no limit in computational power, can completely predict the algorithm's behavior for all inputs. Hence any adversarial input can be simulated by this process, and is captured by the worst case analysis model.

For this reason, many online scheduling problems do not admit a small competitive algorithm. For example, any deterministic non-clairvoyant algorithm has a competitive ratio of $\Omega(n^{1/3})$ for minimizing average flow on a single machine [83]; here n is the number of jobs. If no algorithm is proved to perform well in this model, then it may not have practical implications. Further, if an algorithm has a large competitive ratio, which may be non-trivial and outstanding in theory, it may not be sufficient to convince the system designer to choose the algorithm. Hence several alternative analysis models are also considered.

Randomized algorithms: This is a useful model to remove the unfair power from the adversary that can create an adaptively adversarial input. In this model, the adversary must commit to an input instance a priori without knowing the internal random coins of the algorithm. Formally we say that a randomized algorithm is c-competitive if for all instances the following is satisfied:

$$\mathbb{E}[\mathrm{ALG}(\sigma)] \leq \alpha \mathrm{OPT}(\sigma)$$

where $\mathbb{E}[\mathrm{ALG}(\sigma)]$ is the expected objective of ALG for instance σ. The use of randomized algorithms often dramatically improves the best achievable competitive ratio. For example, there exist a $O(\log n)$-competitive non-clairvoyant randomized algorithm for minimizing average flow time on a singe machine [18]. This analysis model may not be the best when the algorithm's scheduling decisions can affect future jobs.

Average case analysis: In this model, jobs' properties such as jobs' arrival rate and sizes follow a given distribution. Popular distributions include Poisson and exponential. The algorithm's performance is averaged on the input distribution, hence the name average case analysis. Given concrete distributions on jobs' properties, one can measure the absolute expected objective of the online algorithm. This may be useful for provisioning problems, in which the goal is to buy the minimum amount of computing resources to satisfy a scheduling objective to a certain desired level. This model, however, may not be suitable when the distribution is not so predictable.

All models discussed have their advantages and disadvantages. This dissertation focuses on the worst case input model which is of fundamental interest. However, as mentioned above, in this model many problems do not admit an algorithm with small competitive ratio. Thus a relaxed analysis model called resource augmentation, that still keeps the spirit of worst case analysis, was developed [69].

1.3.3 Relaxed Worst Case Analysis: Resource Augmentation

One very popular relaxed worst case analysis model called *resource augmentation* was introduced by Kalyanasundaram and Pruhs [69]. The essence is that the online algorithm is given slightly more resource than the optimal schedule, which allows the online algorithm to compensate for its non-optimal decisions. There are largely two types of resources that are augmented. The first is speed, so the augmentation is called speed augmentation. Formally, the algorithm runs with $s(>1)$-speed on each machine, while the optimal schedule runs with 1-speed. In other words, the algorithm can complete a job in an amount of time that is $\frac{1}{s}$ times the time it takes for the optimal scheduler to complete it. We say that the algorithm is s-speed c-competitive if the algorithm with s-speed has a competitive

ratio of c compared to the optimal schedule with 1-speed. The other type of resource that can be added is a machine. This type of resource augmentation is thus called machine augmentation. In general, speed augmentation provides more power to the algorithm than machine augmentation does. Unless stated otherwise, resource augmentation will refer to speed augmentation.

This relaxed model is justified for the following reasons; for illustration, we will consider speed augmentation model. The first justification stems from how the performance degrades as the load on the system increases. In many systems, there exists a certain threshold of load such that the performance explosively degrades as the load reaches the threshold. See Figure 1.1. Thus, in a certain sense, the competitive ratio has a practical implication only when the load is below the threshold. Since the algorithm can complete each job s times faster than the optimal schedule does, we can interpret this setting as the algorithm being given an amount of load $1/s$ times that the optimal schedule is given to complete. Thus if an algorithm, with speed s given, has a small competitive ratio, then it means that the algorithm performs reasonably for a load that is $1/s$ times the threshold of load (that the optimal solution can handle). For this reason, when for any fixed $\epsilon > 0$, an algorithm with $(1+\epsilon)$-speed is $O(1)$-competitive compared to the optimal schedule with 1-speed, we say the algorithm is *scalable*. Another justification is the increasing availability of speed scaling that allows processors to be sped up by consuming more power. Thus a scalable algorithm can perform close to the optimum by consuming a small extra power. For these reasons, an algorithm being scalable can be a strong evidence to support the qualitatively superior performance of the algorithm.

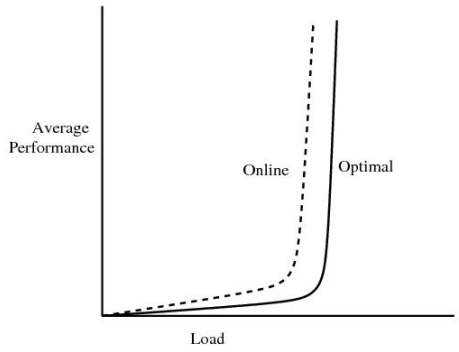

Figure 1.1: The performance curves of online algorithm and the optimal algorithm [87].

1.4 Basic Scheduling Algorithms

We give a quick overview of some of the most popular scheduling policies. Strictly speaking, scheduling algorithms are considered to be able to perform

much more sophisticated computations. For example, a scheduling algorithm can make the current scheduling decision based on the entire history of past jobs and scheduling decisions. In contrast, scheduling policies are considered to be considerably simpler, and make scheduling decisions based on simple rules that are easy to apply. In this dissertation, both scheduling algorithms and policies will be interchangeably used. Let $A(t)$ denote the set of jobs alive at time t. We may equivalently say that jobs in $A(t)$ are in the algorithm A's queue at time t. The number of machines that are used will be denoted as m. In all algorithms, ties are broken in an arbitrary but a fixed way.

- Shortest Remaining Processing Time (SRPT): Always processes the job with the least remaining work.
- Shortest Job First (SJF): Always schedules the job with the least initial work.
- Highest Density First (HDF): Always schedules the job J_i such that $d_i = \frac{w_i}{p_i}$ is maximized.
- Round Robin (RR): At each instantaneous time processes all alive jobs equally. Hence if RR is processing $n = |A(t)|$ jobs on $m \leq n$ machines at time t, then during an infinitesimal interval $[t, t + \mathrm{dt}]$, each alive job is processed on each of the m machines for a $\frac{\mathrm{dt}}{n}$ amount of time.
- Weighted Round Robin (WRR): At each instantaneous time processes all alive jobs in proportion to their weight. On a single machine, each job J_i is processed for a $\frac{w_i \mathrm{dt}}{\sum_{j \in A(t)} w_j}$ amount of time during $[t, t + \mathrm{dt}]$. We note that on multiple machines however this scheduling policy may not be well defined even if $m \leq n$.
- Shortest Elapsed Time First (SETF): Works on the job that has been processed the least. If there are ties the algorithm round robins the jobs that have been processed the least.
- Weighted Shortest Elapsed Time First (WSETF): Works on the job that has the minimum ratio of the amount by which it has processed to its weight. If there are ties, performs WRR among those jobs.
- First In First Out (FIFO): Always schedules the job with earliest arrival time.
- Latest Arrival Processor Sharing (LAPS): This is an extension of RR and takes a constant $\beta \in (0, 1]$ as a parameter. It shares the processing equally among β fraction of the latest arriving jobs. Note that this becomes RR when $\beta = 1$. Formally, $A'(t) \subseteq A(t)$ is such that $|A'(t)| = \lceil \beta |A(t)| \rceil$, and for any job $i \in A'(t)$ and $j \in A(t) \setminus A'(t)$, $r_i \geq r_j$. Then LAPS processes all jobs in $A'(t)$ equally. In other words, LAPS performs RR on the jobs in $A'(t)$.

- Weighted Latest Arrival Processor Sharing (WLAPS): Performs WRR among the most recently arriving alive jobs whose weights add up to β fraction of the total weight of all alive jobs. Formally, $A'(t)$ is the minimal set of jobs with the latest arrival times such that $\sum_{i \in A'(t)} w_i > \beta \sum_{i \in A(t)} w_i$. Then WLAPS performs WRR on the jobs in $A'(t)$; here the job in $A'(t)$ with the earliest arrival time is considered to have weight $\sum_{i \in A'(t)} w_i - \beta \sum_{i \in A(t)} w_i$.

Observe that LAPS, when running on multiple machines, requires that $\beta |A(t)| \geq m$. For its weighted version WLAPS, this may not be sufficient to guarantee a feasible schedule. This issue does not affect the scheduling model for jobs of different parallelizability. This will be further discussed in Section 1.6.2 and Chapter 3.

We note that FIFO is the only non-preemptive algorithm among the above. The algorithms SRPT, SJF and HDF are clairvoyant while RR, WRR, SETF, WSETF, FIFO, LAPS and WLAPS are non-clairvoyant.

1.5 Analysis Tools

We give a quick summary of some popular analysis tools that are frequently used in online scheduling.

1.5.1 Local Competitiveness Argument

In this section we discuss an analysis technique known as local competitiveness. Until relatively recently, this has been the most popular technique used for worst case analysis of scheduling algorithms [69, 19, 71]. Let G denote some objective and let $G_a(t)$ be the cumulative objective in the schedule for algorithm A up to time t. So $\int_0^\infty \frac{\mathtt{d}G_a(t)}{\mathtt{dt}} \mathtt{dt} = G_a$ is the final objective of A. For example, when G is total flow then $\frac{\mathtt{d}G_a(t)}{\mathtt{dt}} = |A(t)|$ and $G_a(\tau) = \int_0^\tau |A(t)| \mathtt{dt}$, where $A(t)$ denotes the set of jobs alive at time t in A's schedule. The algorithm A is said to be *locally c-competitive* if for all times t,

$$\frac{\mathtt{d}G_a(t)}{\mathtt{dt}} \leq c \cdot \frac{\mathtt{d}G_o(t)}{\mathtt{dt}} \tag{1.1}$$

For the objective of total flow time, this implies that at *all* time, the number of jobs in A's queue is comparable to the number of jobs in the optimal scheduler's queue. Most of the early competitive analyses of online scheduling algorithms used local competitiveness. For instance the performance of SRPT on a single machine can be analyzed using local competitiveness. Generally, a proof of local competitiveness uses one of the following techniques: (1) Show by induction on time an invariant concerning the algorithm's queue and the optimal solution's queue, or (2) Fix an arbitrary time t and, by examining the history, show that

optimal does not have have enough processing capability to prevent the online algorithm from being locally competitive at time t.

1.5.2 Potential Functions for Online Scheduling

For problems where local competitiveness is not possible, one alternative form of analysis is amortized local competitiveness. To prove that an online scheduling algorithm A is c-competitive using an amortized local competitiveness argument, it suffices to give a potential function $\Phi(t)$ such that the following conditions hold.

Boundary condition: Φ is zero before any job is released and Φ is non-negative after all jobs are finished.

Completion condition: Summing over all job completions by the optimal solution and the algorithm, Φ does not increase by more than $\beta \cdot G_o$ for some $\beta \geq 0$. Most commonly $\beta = 0$.

Arrival condition: Summing over all job arrivals, Φ does not increase by more than $\alpha \cdot G_{OPT}$ for some $\alpha \geq 0$. Most commonly $\alpha = 0$.

Running condition: At any time t when no job arrives or is completed,

$$\frac{\mathtt{d}G_a(t)}{\mathtt{dt}} + \frac{\mathtt{d}\Phi(t)}{\mathtt{dt}} \leq c \cdot \frac{\mathtt{d}G_o(t)}{\mathtt{dt}} \tag{1.2}$$

Integrating these conditions over time one gets that $G_a - \Phi(0) + \Phi(\infty) \leq (\alpha + \beta + c) \cdot G_o$ by the boundary, arrival and completion conditions. Note that when Φ is identically 0, equation (1.2) is equivalent to the local competitiveness equation (1.1). Generally the value of the potential $\Phi(t)$ depends only on the state (generally how much work is left on each of the jobs) of the online algorithm and the optimal schedule at time t.

The value of the potential function can be thought of as a bank account. If the increase in the online algorithm's objective, $\frac{\mathtt{d}G_a(t)}{\mathtt{dt}}$, is less than c times the increase in the benchmark's objective, $\frac{\mathtt{d}G_o(t)}{\mathtt{dt}}$, then the algorithm can save some money in the bank. Otherwise, it withdraws some money from the bank to pay for its cost, $\frac{\mathtt{d}G_a(t)}{\mathtt{dt}}$. Because of the boundary condition that guarantees a non-negative deposit at the end, the total amount of money that the algorithm withdraws never exceeds its total deposit.

The concept of using a potential function to prove competitiveness goes back at least to the seminal papers by Sleator and Tarjan [94, 95]. The first use of a potential function to use a amortized local competitive argument was in [11]; although, the origination of the idea traces back to [38]. [38] contains a "potential function" but the potential at time t depends not only on the states of the online algorithm and the optimal schedule at time t, but also on the future job arrivals and future schedules. So arguably the amortization argument in [38] is probably closer to a charging argument than to a potential function argument.

The more interested reader is referred to the recent tutorial on potential functions for online scheduling [67].

1.5.3 Conversion between Fractional and Integral Objectives

A recent technique to obtain a competitive algorithm for an (integral) objective is to first obtain an algorithm that is competitive for a fractional objective. The fractional objective for an algorithm A is defined to be $\sum_{i\in[N]} \int_{t=0}^{\infty} \frac{p_i^a(t)}{p_i} \frac{\mathtt{d}G_a(t,J_i)}{\mathtt{dt}} \mathtt{dt}$ where $G(t, J_i)$ is the total cost of job J_i up to time t and $p_i^a(t)$ is the remaining work of job J_i under A's schedule at time t. The fractional objective is usually considered when $G(t, J_i)$ depends only on the flow time of job J_i. As an example. consider the objective of weighted flow. The fractional weighted flow time of a job J_i is $\int_{r_i}^{C_i} w_i \frac{p_i^a(t)}{p_i} \mathtt{dt}$. We call $\frac{p_i^a(t)}{p_i}$ the remaining fraction of J_i at time t. The total weighted fractional flow time objective is $\int_{t=0}^{\infty} \sum_{J_i\in A(t)} w_i \frac{p_i^a(t)}{p_i} \mathtt{dt}$. An interpretation of the fractional weighted flow time objective is that a job contributes to the objective in proportion to the amount of remaining work the job has. Notice that the total fractional weighted flow time of any schedule is at most the integral weighted flow time of the schedule.

The concept of fractional objectives has proved to be useful for analyzing online scheduling algorithms. Generally it is easier for an online algorithm to be competitive for fractional objectives. Further, fractional objectives are often easier to reason about. To the best of our knowledge, the use of fractional objectives to aid in the analysis of online scheduling algorithms originates from [19]. It is generally possible to convert a scheduler A that is good for a fractional objective into an algorithm A' that is good for an integer objective with minimal speed augmentation in the following way: The algorithm A' always schedules the exact same jobs as A at any time, except a $(1+\epsilon)$ factor faster in rate of speed, unless the job has been completed in A''s schedule. If A is s-speed c-competitive for a fractional objective then generally A' is $(1+\epsilon)s$-speed $O(c/\epsilon)$-competitive for the corresponding integer objective.

1.6 Problem Definition and Overview of Contributions

In this section, we define the problems addressed in this dissertation, and provide a summary of our contributions. As mentioned before, we study the scheduling models – broadcast scheduling, scheduling of jobs with different parallelizability and scheduling on heterogeneous machines. A common technical challenge arising in these models is that two different schedules may have to do different amount of work. This intuitively makes it hard to compare the online algorithm to the optimal scheduler. Particularly, we cannot use the popular local competitiveness

argument to prove our algorithms' competitiveness ratio. Our results are enabled by the development of new and more sophisticated analysis tools, which could be of potential use for other scheduling problems.

Before covering each of the scheduling models, we compare them at a high level to help the reader understand the differences between them. Broadcast scheduling is a single machine setting, while other models have multiple machines. However, broadcast scheduling is unique in that the machine can process multiple jobs (requests) simultaneously. This is fundamentally different from the time sharing preemptive scheduling where the processing power is divided among jobs. The model of scheduling jobs with different parallelizability distinguishes itself from others in that a job can be processed simultaneously by multiple machines. The best situation that captures this model is when jobs are scheduled on a multi-core processor in which a job's execution can be expedited by spawning multiple threads for a job. The heterogeneous machines setting is a more classic model. In the model, at any time a job can only be processed on a single machine. In all scheduling models covered this dissertation, preemption is allowed and incurs no penalty.

1.6.1 Broadcast Scheduling

In this model, there is a server that stores n pages, each of which contains an individual useful data. Each request $J_{p,i}$ arrives over time asking for a specific page p. The server must satisfy all requests. When the server *broadcasts* a page p, all outstanding requests for the same page p are satisfied *simultaneously* [3]. This is the main difference from standard scheduling settings where the server must process each request separately. The broadcast model is motivated by several applications such as multicast systems and wireless and LAN networks [102, 1, 2, 60]. Broadcast scheduling can be seen as a special case of batch scheduling that has been studied in stochastic and queueing theory literature on related models [37, 36, 99, 100].

We focus on the problem of minimizing average flow time (the ℓ_1 norm). There is a long series of work for this objective. The most notable results in the offline setting are as follows. The problem was shown to be NP-hard [46, 29]. The first (offline) scalable algorithm was given by Bansal et al. in [10], and the best approximation without speed augmentation is a $O(\log^2 n/\log\log(n))$-approximation [12].

In the online setting, it was shown that without resource augmentation any online deterministic algorithm is $\Omega(n)$-competitive [71]. Further, any randomized online algorithm has a lower bound of $\Omega(\sqrt{n})$ on the competitive ratio [10]. These strong lowerbounds without resource augmentation has led previous work to focus on finding $O(1)$-speed $O(1)$-competitive algorithms. Previously, there have

[3] In broadcast scheduling, we will use the term request rather than job to emphasize this unique aspect of broadcast scheduling.

been largely two different directions. The first was based on a reduction from the problem of minimizing average flow time in broadcast scheduling to a non-clairvoyant scheduling problem. However, the best result that one could obtain using this reduction was a $(2+\epsilon)$-speed $O(1)$-competitive algorithm [38, 40, 42].

Another approach was based on a natural greedy algorithm called Longest Wait First (LWF). The algorithm LWF always broadcasts a page that has accumulated the largest flow time. Edmonds and Pruhs showed that LWF is 6-speed $O(1)$-competitive, but they also proved that it is not scalable [41]. Since the algorithm LWF seemed more natural then the algorithms based on reduction, we aimed at better understanding the novel analysis techniques introduced in [41], hoping that it might lead to a scalable algorithm. In joint-work with Chekuri and Moseley, we were able to simplify these techniques to make the key ideas more transparent. Using this, we were able to show LWF is $(4+\epsilon)$-speed $O(1)$-competitive [32].

With the better understanding of LWF, we were able to give the first scalable algorithm which is $(1+\epsilon)$-speed $O(1/\epsilon^{11})$ competitive [65]. The algorithm we introduced was a variant of LWF that fixed the drawback of LWF. The main problem of LWF is that it does not care about how fast each page accumulates flow time. We observed that it might be more effective to broadcast a page p over page q if page p has more recent requests, even if they have accumulated similar total flow time. This is because page p will accumulate flow time more quickly than page q. By formalizing this idea and using our simplified analysis of LWF in [32], we were able to fix the main problem of LWF, thereby successfully giving the first scalable algorithm.

This thesis presents the simpler analysis of LWF and the first scalable algorithm in [32, 65].

Remark 1. *Later Bansal et al. gave an elegant and improved algorithm that is $(1+\epsilon)$-speed $O(1/\epsilon^3)$ competitive [15] via novel view of fractional broadcast schedule and online rounding scheme. Their result also works for non-uniform sized pages. Then we extended their result to the ℓ_k-norms of flow time. We first gave an algorithm that is $(k+\epsilon)$-speed $O(k)$-competitive [55], and then improved the analysis to show the same algorithm is in fact scalable [44]. We note that LWF is $O(k)$-competitive with $3k$-speed [32].*

1.6.2 Arbitrary Speed-up Curves (Scheduling Jobs of Different Parallelizability)

This model is useful when jobs have varying degrees of *parallelizability* on a multiprocessor system. That is, some jobs may be sped up considerably when simultaneously run on multiple processors, while other jobs may be sped up by very little. The most obvious settings where this problem arises are scheduling multi-threaded processes on a chip with multiple cores/processors, and scheduling multi-processor applications in a server farm.

In this model, which was proposed in [43], there are m identical machines and n jobs of different parallelizability. Formally, each job consists of a sequence of phases. Each phase needs to finish some amount of work, and has a speedup function that specifies the rate at which work is processed in that particular phase (as a function of the number of processors assigned to the job). A speedup curve function is assumed to be concave which implies that the processing power per machine does not increase as more machines are used. This model is also known as the arbitrary speed up curves model, since each job is associated with possibly a different speedup curve. The scheduler at each time needs to allocate m machines to the current outstanding jobs. To further capture the practical setting, we require that our algorithm be *non-clairvoyant*. That is, the algorithm is completely unaware of the processing time of the jobs or their parallelizability. This model has received a considerable interest since it models a very general parallel computing models. [38, 42, 40, 90, 26]

It is well known that no non-clairvoyant algorithm can be $O(1)$-competitive for average flow time, even in the standard single machine model. Due to this strong lower bound, we will be assuming a resource augmentation model [69]. The first positive result for average flow time in the speed up curves setting was given by Edmonds [38]. It was shown that that EQUI (Round-Robin) [4] is $(2+\epsilon)$-speed $O(1)$-competitive. Recently, [42] gave an elegant potential function analysis to show that Latest Arrival Processor Sharing (LAPS) is scalable.

We extended their result to ℓ_k-norms of flow time. We first gave an algorithm that is $(k+\epsilon)$-speed $O(k)$-competitive [55], and then improved the analysis by showing that the same algorithm is $(1+12\epsilon)$-speed $O(k12^k/\epsilon^{2k+1})$-competitive [44]. This dissertation presents the improved scalable algorithm as discussed in [44].

1.6.3 Heterogeneous Machines

When there are multiple machines, we can consider the following three settings in the order of increasing complexity in terms of the relationship between jobs and machines. The second and third settings are examples of the heterogeneous machines.

- Identical machines: All machines are identical. Each job takes the same amount of time to be processed on all machines.
- (Uniformly) related machines: Machine i processes jobs with speed s_i. Hence each job j takes p_j/s_i amount of time on machine i.
- Unrelated machines: Each job j may have a completely different processing time p_{ij} and weight w_{ij} on each machine i it is assigned to. Here $p_{ij}, w_{ij} \in$

[4] EQUI stands for Equi-partition and it allocates machines uniformly to jobs. This has a spirit very similar to RR (Round Robin). The algorithm name RR is more commonly used in the standard setting where a job can be processed on at most one machine at all times. In this dissertation, RR may be used to refer to EQUI.

$[0, \infty]$. Equivalently it can be viewed that j has a unit size and is processed with speed $1/p_{ij}$ on machine j. This is probably the most general model for multiple machines and captures the above two settings. Further, this model captures more general settings where a job can be scheduled only on specific machines or can be substantially sped up on some highly specialized machines for a certain purpose.

When there are multiple machines present, there are two properties that are desired from online algorithms. The first one is *immediate-dispatch*. An algorithm is said to be immediate-dispatch if it immediately sends an arriving job to a specific machine. This is particularly useful when the main scheduler does not have enough memory to hold a lot of incoming jobs. The next property is *non-migratory*. We say that an algorithm is non-migratory if a job is sent to a machine, then the job cannot move to other machines. This property is desirable when it is costly to move jobs across machines.

Even in the simplest identical parallel machine setting, the problem of minimizing the average flow time is non-trivial. The best approximation ratio is $O(min(\log P, \log n/m))$, where P is the ratio of the maximum job size to the minimum job size. This competitive ratio is achieved by SRPT and there exists a matching lower bound [78]. Avrahami and Azar gave an immediate-dispatch and non-migratory algorithm which has the same asymptotic competitive ratio [5]. We note that these remain the best result even for the offline case.

In addition, there exist other interesting results with resource augmentation. Phillips et al. showed that one can obtain a schedule with 2-speed that is as good as the optimal schedule [84]. Chekuri et al. gave the first scalable algorithm which is $(1+\epsilon)$-speed $O(1/\epsilon^3)$-competitive [31]. Their algorithm was surprisingly simple: assign each arriving job to a machine chosen uniformly randomly and run SRPT or SJF on each machine.

The problem, either in the offline setting or in the online setting, becomes more challenging in the related machine setting. Until fairly recently, there were very few positive results known [50, 51]. Garg and Kumar gave the first non-trivial algorithm which is $O(\log^2 P \cdot \log S)$-competitive where S is the ratio of the highest speed and the smallest speed of a machine. Later the same authors gave an $O(\log P)$-approximation for the offline case and an $O(\log P \cdot \log S)$-competitive algorithm for the online case.

In the unrelated machine setting, there exist only a few positive results without speed augmentation [52, 53, 93]. Even for the special case called restricted assignment case where each job can be assigned to a subset of machines, ,i.e., $p_{ij} \in \{p_j, \infty\}$ the best approximation ratio known is $O(\log P)$ and there is a lower bound of $\Omega(\log P/\log\log P)$ [52].

In a breakthrough result, Chadha et al. gave a very simple scalable algorithm based on a novel potential function [24]. The algorithm is surprisingly simple and the analysis is based on an elegant potential function.

Clairvoyant Scheduling on Unrelated Machines

Inspired by the breakthrough result by Chadha et al. [24], we gave the first scalable algorithm for the ℓ_k-norms of flow time which is $(1+\epsilon)$-speed $O(k/\epsilon^{2+2/k})$-competitive [66] . Our algorithm, based on a sophisticated potential function, is also immediate-dispatch and non-migratory. We also showed any immediate-dispatch and non-migratory algorithm has a competitive ratio of $\Omega(k)$. To enable our analysis, we gave a new lower bound on the optimal scheduler's objective.

Remark 2. *Recently, Anand et al. gave another algorithm that has a slightly better competitive ratio [3]. More specifically, their lgorithm is* $(1+\epsilon)$*-speed* $O(k/\epsilon^{2+1/k})$*-competitive. Interestingly, their analysis is based on linear programming and dual fitting.*

Non-clairvoyant Scheduling on Related Machines

The first non-trivial non-clairvoyant scheduling on related machines was given by [56]. They reduced the problem to a singe machine Round Robin scheduling (RR) and gave a $(2+\epsilon)$-speed $O(1)$-competitive algorithm. We showed Latest Arrival Processor Sharing (LAPS), a now well-known extension of RR, is scalable [59]. We note that our result is the first non-clairvoyant scalable algorithm in the heterogeneous machines setting. Our result extends to a more general setting where each machine is associated with a speed function that specifies the speed for the power at which the machine is run and the goal is to minimize the total (unweighted) flow time plus total energy.

One may wonder if our algorithm and analysis easily follow from those in [42]. In [42], Edmonds and Pruhs showed that LAPS is scalable using a novel potential function. Recall that LAPS performs Round-Robin among the βn fraction of alive jobs that arrived most recently, where n is the number of jobs that are currently alive and β is the parameter that LAPS is given. However, it is not clear how this extension should be done in the multiple machines setting. Namely, we cannot run βn jobs on n machines. Surprisingly, we showed that running the βn jobs on the fastest βn jobs suffices to yield a scalable algorithm. Hence our algorithm is fairly different from the ones presented in [42, 56].

However, we note that our algorithm works only for unweighted jobs. When jobs have varying weights, the problem becomes much more challenging. Suppose one job has a very heavy weight and we run RR (or LAPS). Then the heavy weight job should be given all processing power, which yields a non-feasible schedule. We also show several natural algorithms are not scalable with any constant speed up.

1.7 Dissertation Outline

Our results on broadcast scheduling and scheduling for jobs with different parallelizability are given in Chapters 2 and 3, respectively. Our results on scheduling for heterogeneous machines are presented in Chapters 4 and 5, which cover the unrelated and related machine settings, respectively. In each of these chapters, several open problems will be described. This dissertation concludes with some future research directions in Chapter 6.

Chapter 2

Broadcast Scheduling

2.1 Introduction

We consider the pull-based broadcast scheduling model. In this model, there are n pages (representing some form of useful information) available at a server and requests arrive for pages over time. When the server *broadcasts* a page p, all outstanding requests for the same page p are satisfied simultaneously. This is the main difference from standard scheduling settings where the server must process each request separately. The broadcast model is motivated by several applications such as multicast systems and wireless and LAN networks [102, 1, 2, 60]. Work has also been done in stochastic and queueing theory literature on related models [37, 36, 99, 100].

In this chapter we concentrate on the online model with the goal of minimizing the total (or equivalently average) flow time [1]. This is one of the most popular quality of service metrics. The ith request for page p will be denoted $J_{p,i}$. Request is often referred to as job in the scheduling literature. The request $J_{p,i}$ arrives at time $r_{p,i}$ and, in the online model, this is when the server is first aware of the request. Time is slotted and a *single* page can be broadcasted in a time slot. This model also captures the algorithmic difficulty of the problem and this is the model almost exclusively addressed in previous literature. The total flow time of a given schedule can be written as $\sum_p \sum_i (C_{p,i} - r_{p,i})$, where $C_{p,i}$ is the time when $J_{p,i}$ is satisfied.

Besides the practical interest in the model, broadcast scheduling has seen substantial interest in algorithmic scheduling literature both in the offline and online settings [16, 2, 1, 17, 60, 71, 46, 48, 49, 10, 12]. It was because the model is very simple to describe and nevertheless poses algorithmic challenges. To have a feel of the difficulty, consider the algorithm Most Requests First (MRF) which broadcasts the page that has the largest number of unsatisfied requests. This algorithm may seem like the most natural candidate for the problem. However, it was shown that MRF is not $O(1)$-competitive even when given any fixed extra speed [71]. A simple example shows that MRF may repeatedly broadcast the same page, while ignoring requests which eventually accumulate a large amount of flow time. The optimal solution can take advantage of the

[1] Flow time is often referred to response time or wait time.

broadcast setting and satisfy the requests MRF was busy working on by a single broadcast. This leaves the optimal solution free to work on other requests that are unsatisfied under MRF's schedule.

In a nutshell, a main difficulty in the analysis of algorithms comes from the fact that two different schedules may have to do different amount of work to satisfy the same set of requests. Intuitively, this makes it hard to compare the algorithm's status to the optimal scheduler's status. In fact, it was shown that no online algorithm can be *locally competitive* with an adversary, even with a constant speed-up [71] [2]. Local competitiveness has been one of the most popular methods of analysis in standard scheduling [69, 19, 71].

The difficulty is also indicated by the strong lower bounds on the achievable competitive ratio. It was shown that without resource augmentation any *online* deterministic algorithm is $\Omega(n)$-competitive [71]. Further, any randomized online algorithm has a lower bound of $\Omega(\sqrt{n})$ on the competitive ratio [10]. Due to these strong lowerbounds we focus on the resource augmentation model [69] where an algorithm A is given $s \geq 1$ speed and is compared to an optimal offline solution that has 1 speed; see Section 1.3.3. We will let A_s be the flow time accumulated for an algorithm A when given s speed; sometimes we will allow A_s to denote the algorithm itself with s speed if there is no confusion in the context.

Even though broadcast scheduling has been studied extensively over the last decade, the complexity of the problem is not well understood. In the *offline* setting, minimizing average flow time was first studied using non-trivial linear programming techniques coupled with resource augmentation [71, 48, 49]. It was not until later that a complex reduction showed that this problem was in fact NP-Hard [46]. Later, a simpler proof of this fact was found [29]. Following this line of work, a $(1+\epsilon)$-speed $O(1)$-approximation algorithm was eventually given in [10]. 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 [10]. More recently, a $O(\log^2 n/\log\log(n))$-approximation was shown in [12]. 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. Previously, there have been two main approaches in this direction. The first was given by Edmonds and Pruhs in [40]. 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.

[2] 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. See Section 1.5.1 for the formal definition of local competitiveness.

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 [38, 40]. More recently, the same authors used this reduction to show another algorithm is $(2+\epsilon)$-speed $O(1)$-competitive [42]. Both of these algorithms can be extended to the case where pages have varying sizes. Notice that a factor of 2 in the speed is lost in the reduction and, therefore, the reduction cannot be used to show a scalable algorithm.

The second was based on the natural greedy algorithm Longest Wait First (LWF), which was first introduced in [71]. LWF 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 [41]. In this work, new novel techniques were introduced to avoid a local argument. However, LWF was shown to be $n^{\Omega(1)}$-competitive when given speed less than 1.618 [41].

2.1.1 Our Results

For the problem of minimizing total flow time in broadcast scheduling, we give the first online *scalable* algorithm LA-W for Latest Arrival time with Waiting. We prove that LA-W is $(1+\epsilon)$-speed $O(1/\epsilon^{11})$-competitive for any $0 < \epsilon \leq 1$, giving a positive answer to 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 [42]. The algorithm LA-W focuses on *pages* which have requests that arrived recently. This is fundamentally different from the algorithm given in [42], 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.

The analysis of LA-W was enabled by our simpler analysis of the algorithm LWF. Although the techniques presented in [41] were novel, they were quite complex. In joint work with Chekuri and Moseley, we were able to simplify these techniques to make the key ideas more transparent. Using this, we were able to show LWF is $(4+\epsilon)$-speed $O(1/\epsilon^2)$-competitive [32] [3]. More importantly, the key ideas and analysis tools were generalized to design and analyze the first scalable algorithm LA-W.

In this dissertation, we first present the easy analysis showing that LWF is $O(1)$-competitive with 5-speed in Section 2.3. We then present the scalable algorithm LA-W and its analysis in Section 2.4.

[3] Further it was shown that LWF is $O(1)$-competitive with 3.4-speed in [32]. Later we improved the lower bound on the competitive ratio to show that LWF is not $O(1)$-competitive with $2-\epsilon$-speed for any $\epsilon > 0$ [64].

Remark 3. *Following our work, Bansal et al. gave another scalable algorithm which is* $(1+\epsilon)$*-speed* $O(1/\epsilon^3)$*-competitive [15]. They showed that LAPS is fractionally scalable and converted the fractionally scalable schedule into an integrally scalable one by using a small amount of extra speed. Their algorithm works also for varying sized pages.*

2.1.2 Related Work

In this section we give an overview of related work in broadcast scheduling. Charikar and Khuller considered a generalization of average flow time where the goal is to minimize the average flow time for a fraction of the requests [30]. Besides work on minimizing the total flow time, other objective functions have been considered in the broadcast model. In [17, 29, 33], it was shown that the algorithm First In First Out (FIFO) is 2-competitive for the problem of minimizing the maximum response time. This is the case even for varying sized pages. It was further shown that for any $\epsilon > 0$, no online algorithm can have a competitive ratio of $2 - \epsilon$ [34]. We note that this remains the best result even for the offline case. The only known hardness result is that the problem is NP-complete [29]. For the problem of minimizing the maximum weighted flow time, a $(1+\epsilon)$-speed $O(1)$-competitive algorithm was been given by [33] [4].

When each request has a deadline, constant competitive algorithms were given by [73, 28, 103, 35] with the the objective of maximizing the number of requests satisfied by their deadlines. For the problem of minimizing the ℓ_k-norms of flow time and the delay factor, [32] gave $O(k)$-speed $O(k)$-competitive algorithms. The first scalable algorithm for the ℓ_k-norm was given in [44]. For the objective of minimizing total flow time plus total energy consumption, Moseley gave the first scalable algorithm [82]. For empirical evaluation, see [60, 2].

2.2 Formal Problem Statement and Notation

In the formal model, the server has n distinct unit-sized pages of information; the non-uniform sized page case will be discussed soon. The clients send their respective requests to the server asking for a specific page. This model is called pull-based, since the clients initiate the requests (in the push-based model, the server broadcasts the pages according to some frequency). We use $J_{p,i}$ to denote i'th request for a page $p \in \{1, \ldots, n\}$. We let $r_{p,i}$ denote the arrival time of the request $J_{p,i}$. The finish time $C^A_{p,i}$ of a request $J_{p,i}$ under a given schedule/algorithm A is defined to be the earliest time after $r_{p,i}$ when the page p is transmitted by the scheduler; for notational convenience we may omit A and simply use $C_{p,i}$ when the algorithm under consideration is clear in the context. Note that multiple requests for the same page can have the same finish time. The total flow time for an algorithm over a sequence of requests

[4] The result was extended to the varying sized page case in the journal version.

is now defined as $\sum_p \sum_i (C_{p,i} - r_{p,i})$. The ℓ_k norms of flow time is defined as $\sqrt[k]{\sum_p \sum_i (C_{p,i} - r_{p,i})^k}$.

In broadcast scheduling, considering the weight of requests does not increase the generality of the problem. This is because the weight $w_{p,i}$ of request $J_{p,i}$ can be easily captured by $w_{p,i}$ copies of the same request of unit weight.

Time model: For simplicity, when the speed $s > 1$ that the server is given is an integer, we assume the discrete time model. This is the time model we adopt in the analysis of LWF in Section 2.3. 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 that 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. The algorithm with an integer speed s given can transmit (at most) s pages in each time slot.

When the speed $s > 1$ that the server is given is fractional, we assume that the server schedules a page every $1/s$ time-steps starting from time 0. We use this time model for the analysis of LA-W in Section 2.4. 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 results will hold in either case and this assumption improves the readability of the analysis.

Another model was also used in some broadcast scheduling literature [15, 55, 44]. If the speed $s > 1$ is fractional, let $\epsilon := s - \lfloor s \rfloor$. Usually $\epsilon > 0$ is considered to be arbitrary small, hence one can assume that $1/\epsilon$ is integral. Then at each integer time, the algorithm broadcasts $\lfloor s \rfloor$ pages, and at every $1/\epsilon$ time slots, it broadcasts an extra page.

We note that all the above time models are essentially equivalent in the sense that all existing results translate from one model to another, with a loss of at most a constant factor in the competitive ratio.

Varying sized pages: Finally, we discuss the case where pages may have non-uniform sizes. In this case, we need to carefully define when a request for a page is satisfied if the request arrives midway through the transmission of the page. The most popular model is the sequential model used in [40, 42, 15, 55, 44]. Let each page p consist of L_p pieces of information, $(1, p), (2, p), ..., (L_p, p)$. A request $J_{p,i}$ can be satisfied only when it receives each piece of page p sequentially. Preemption is allowed. In other words, the request $J_{p,i}$is satisfied at the first time when it receives all the pieces $(1, p), (2, p), ..., (L_p, p)$ in this order (since its

release time $r_{p,i}$), and there could be other transmits between these transmits. There are other interesting models when the client has a buffering capability. For more details, see [86].

2.3 Longest Wait First

This section is devoted to the analysis showing that LWF is $O(1)$-competitive with 5-speed. In the broadcast setting LWF with integer 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 $\sum_{J_{p,i} \in U(t)} (t - a_{p,i})$.

We first 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.

Our analysis of LWF is inspired by that in [41]. Here we summarize our approach and indicate the main differences from the analysis in [41]. 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 $\text{LWF}_s = \text{LWF}_s^S + \text{LWF}_s^N \leq \rho \text{OPT}^S + \rho \text{OPT}^N + \delta \text{LWF}_s \leq \rho \text{OPT} + \delta \text{LWF}_s$. This shows that $\text{LWF}_s \leq \frac{\rho}{1-\delta} \text{OPT}$, which will complete our analysis. Perhaps the key idea in [41] is the idea of charging LWF_s^N to LWF_s with a $\delta < 1$; as shown in [71], 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 [41], 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 simpler proof improves the speed bounds, and can be easily extended to other objectives such as the ℓ_k norms of flow.

2.3.1 Preliminaries

To show that $\text{LWF}_s^N \leq \delta\text{LWF}_s + \rho\text{OPT}^N$, 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 $\ell_{u,v}$ where $\ell_{u,v}$ is not necessarily an integer. Note that a vertex can be matched to multiple vertices.

Definition 1 (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 $u \in X$, $\sum_{(u,v)\in E} \ell_{u,v} = 1$ and for each $v \in Y$, $\sum_{(u,v)\in E} \ell_{u,v} \leq c$.*

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

Lemma 4 (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 Section 2.3, 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 4, we show another lemma which will be used throughout the analysis of LWF. Lemma 5 says that the union of some fraction of time intervals is comparable to that of the whole time interval.

Lemma 5. *Let $0 \leq \lambda \leq 1$ be a constant. 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 $t \geq t_1$. □

2.3.2 Analysis

A fair amount of notation is needed to clearly illustrate our ideas. Following [41], 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 2.3.

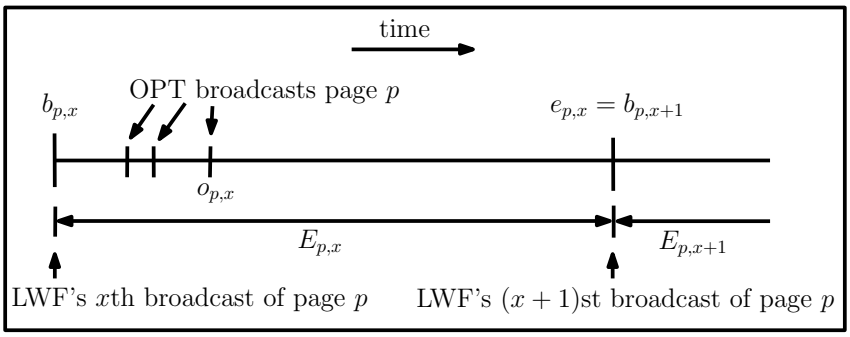

Figure 2.1: Events for page p.

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 $\text{LWF}^S_s = \sum_{p,x:E_{p,x}\in S} F_{p,x}$ and $\text{LWF}^N_s = \sum_{p,x:E_{p,x}\in N} F_{p,x}$ denote the the total flow time for self-chargeable and non self-chargeable events respectively. Similarly, let $\text{OPT}^S = \sum_{p,x:E_{p,x}\in S} F^*_{p,x}$ and $\text{OPT}^N = \sum_{p,x:E_{p,x}\in N} F^*_{p,x}$. 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 $\text{LWF}^{N^e}_s$ and $\text{LWF}^{N^l}_s$ 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 6. $\text{LWF}^S_s \leq \rho\text{OPT}^S$.

Lemma 7. $\text{LWF}^{N^l}_s \leq \rho\text{OPT}^N$.

Thus the main task is to bound $\text{LWF}^{N^e}_s$. 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 8. *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. We will prove the following main lemma that bounds the flow-time of early requests of non self-chargeable events.

Lemma 9. *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.

Theorem 1. $\text{LWF}_5 \leq 90\text{OPT}$.

Proof. By combining Lemma 6, 7 and 9, 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 9. 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 8, 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 4) 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 8, 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 5, $|\mathcal{I}'| \geq \frac{\rho-1}{2\rho}|\mathcal{I}|$ 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 4, 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 $\mathrm{LWF}_5^{N^e}$.

$$
\begin{aligned}
& \mathrm{LWF}_5^{N^e} \\
= & \sum_{u_{p,x} \in X} F^e_{p,x} \text{ [By Definition]} \\
= & \sum_{u_{p,x}v_{q,y} \in E} \ell_{u_{p,x},v_{q,y}} F^e_{p,x} \text{ [By Def. 1, i.e. for } \forall u_{p,x} \in X, \textstyle\sum_{v_{q,y} \in Y} \ell_{u_{p,x},v_{q,y}} = 1] \\
\leq & \sum_{u_{p,x}v_{q,y} \in E} \ell_{u_{p,x},v_{q,y}} 2F_{q,y} \text{ [By Lemma 8]} \\
\leq & \frac{4\rho}{5(\rho-1)} \sum_{v_{q,y} \in Y} F_{q,y} \text{ [Change order of } \textstyle\sum \text{ and } \ell \text{ is a } \frac{2\rho}{5(\rho-1)}\text{-covering]} \\
\leq & \frac{4\rho}{5(\rho-1)} \mathrm{LWF}_5. \text{ [Since } Y \text{ includes all events]}
\end{aligned}
$$

This finishes the proof of Lemma 9.

Remark 10. *One can easily extend the analysis of* LWF *given in this section to show that* LWF *is* $(4+\epsilon)$*-speed* $O(1/\epsilon^2)$*-competitive.*

2.4 First Scalable Algorithm: Latest Arrival time with Waiting (LA-W)

In this section, we give the first scalable algorithm for minimizing the total flow time in broadcast scheduling for unit-sized pages. We start with giving an overview of the algorithm LA-W. 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, [41] 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 [33]. We however were unable to determine its performance for average flow time when given less than 2 speed.

2.4.1 Algorithm

We assume that all pages have a unit size, and 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. For further discussion of our time model, see Section 2.2.

Before introducing our algorithm, we state notation that will be used throughout Section 2.4. 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 $\beta < 1$ 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

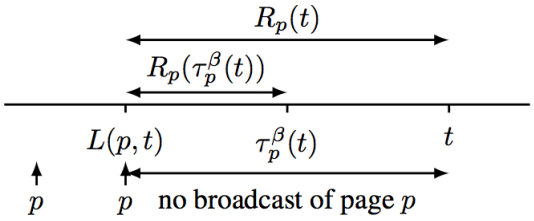

Figure 2.2: $R_p(t)$ denotes the alive requests of page p at time t, i.e. the requests of page p which arrived during $[L(p,t),t]$. Likewise, $R_p(\tau_p^\beta(t))$ denotes the requests which arrived during $[L(p,t),\tau_p^\beta(t)]$.

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 2.2.

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 $Q(t) = \{q \mid F_q(t) \ge \frac{1}{c}F_{max}(t)\}$ 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.

2.4.2 Analysis

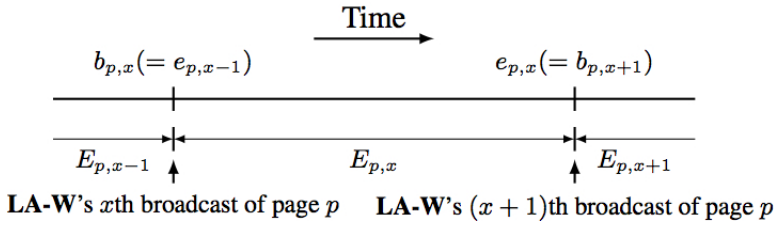

Figure 2.3: Events for page p.

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 σ. Otherwise, our arguments can be applied to each maximal time interval where LA-W$_{1+\epsilon}$ is busy. Following the lead of [41, 32], 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 2.3.

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

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 [32] 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 16.

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 [33] and [41] 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 11 and Lemma 16, 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 2. *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 $I = [\tau^\beta_p(e_{p,x}), e_{p,x})$ 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 12. *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 $[\tau^\beta_p(e_{p,x}), e_{p,x})$.*

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 $(b_{p,x}, \tau^\beta_p(e_{p,x}))$. 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 12, 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 13. *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 14. *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. $F(R', t + \eta(e_{p,x} - t)) = \sum_{J_{p,i} \in R'}(t + \eta(e_{p,x} - t) - r_{p,i}) = \sum_{J_{p,i} \in R'}((1 - \eta)t + \eta e_{p,x} - r_{p,i}) \geq \sum_{J_{p,i} \in R'}((1-\eta)r_{p,i} + \eta e_{p,x} - r_{p,i}) = \eta \sum_{J_{p,i} \in R'}(e_{p,x} - r_{p,i}) = \eta F(R')$. 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 [41, 32]. This lemma shows how to charge the flow time of some events to the total flow time LA-W$_{1+\epsilon}$ accumulates.

Lemma 15. *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,*

$$F(\mathcal{A}) \leq (\frac{2}{\lambda \kappa \mu})(\frac{d+1}{d}) F(\mathcal{B}) \leq (\frac{2}{\lambda \kappa \mu})(\frac{d+1}{d}) \text{LA-W}_{1+\epsilon}.$$

Proof. 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

$$\lambda |Z| \leq \frac{d+1}{d} |\bigcup \mathcal{I}(Z)|. \tag{2.1}$$

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,

$$|\bigcup \mathcal{I}'| = |\bigcup \mathcal{I}(Z)|. \tag{2.2}$$

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

$$|\bigcup \mathcal{I}'_{odd}| \geq \frac{1}{2} |\bigcup \mathcal{I}'|. \tag{2.3}$$

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

$$|N_G(Z)| \geq \mu|\bigcup \mathcal{I}'_{odd}|. \tag{2.4}$$

From (2.1), (2.2), (2.3) and (2.4), 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 4. Let ℓ be such a covering.

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

□

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 $F(\mathcal{A}) \ll$ [5]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 15, 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 [32], the definition of τ and Lemma 15. The proof is similar to that given in [32].

Lemma 16. $F(\mathcal{N}_1) \leq O(\frac{1}{\epsilon^{11}})$OPT.

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

Proof. We apply Lemma 15 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 12, 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 13, 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 $I_{p,x} \setminus I'_{p,x}$. 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 14 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 15,

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

□

We now focus on bounding the flow time of events in $\mathcal{N}_2$. To exploit Lemma 15, $\mathcal{N}_2$ 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 15 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 15. 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-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 $\tau^{\beta,i}_{p,x} = \tau^{\beta(\frac{10}{\epsilon}i+1)}_p(e_{p,x})$. 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 $\lceil \frac{\epsilon s}{10} |[\tau^{\beta,i}_{p,x}, e_{p,x})| \rceil$ broadcasts by our algorithm on the interval $[\tau^{\beta,i}_{p,x}, e_{p,x})$.
- $\mathbb{T}_2$ if and only if $E_{p,x} \notin \mathbb{T}_1$ and for all $0 \leq i \leq \lceil 1000c \rceil$, $\tau^{\beta,i}_{p,x} - \tau^{\beta,i+1}_{p,x} \geq \frac{\epsilon}{10}(e_{p,x} - \tau^{\beta,i}_{p,x})$
- $\mathbb{T}_3$ 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 $\tau^{\beta,i}_{p,x}$ must exist for all $0 \leq i \leq \lceil 1000c \rceil + 2$. The rest of the this chapter is organized as follows. In Section 2.4.2 we will show that $F(\mathbb{T}_1) \ll \text{LA-W}_{1+\epsilon}$. Then in Section 2.4.2 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 2.4.2. 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 $\tau^{\beta,\lceil 1000c \rceil + 2}_{p,x} = \tau^{\beta k}_p(e_{p,x})$ for any page p by definition of k and $\tau^{\beta,i}_{p,x}$. 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.

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 $t_{p,x} = \tau^{\beta,i}_{p,x}$ 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 17. *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 14,

$$F(S_{[b_{p,x},t_{p,x}]}, t) \geq \frac{\epsilon}{20}(1-\beta k)F_{p,x}. \tag{2.5}$$

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 (2.5), 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 13 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 15, 17, 12 and 13.

Lemma 18. $F(\mathbb{T}_1) < \frac{83}{100}$LA-W$_{1+\epsilon}$.

Proof. We apply Lemma 15 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 12, 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 17 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 13 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

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

□

Bounding $\mathbb{T}_2$ events

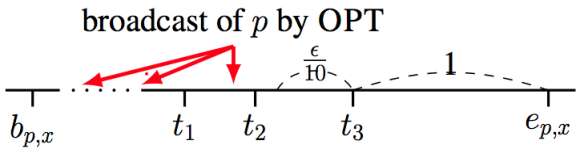

Figure 2.4: For any event $E_{p,x}$ in $\mathbb{T}_2$, OPT must broadcast page p during $[t_1, t_3)$.

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 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 19. *Let $E_{p,x}$ be an event in $\mathbb{T}_2$. For any integer i s.t. $0 \le i \le \lceil 1000c \rceil$, 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 $0 \le i \le \lceil 1000c \rceil$, 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 $t_3 - t_2 \ge \frac{\epsilon}{10}(e_{p,x} - t_3)$ and $t_1 < t_2 < t_3$, since $E_{p,x} \in \mathbb{T}_2$. See Figure 2.4. 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 $F(S_{(t_2,e_{p,x})}) \le \beta(\frac{10}{\epsilon}(i+1)+1)F_{p,x}$. Thus we have,

$$F(S_{[t_1,t_2]}) = F(S_{[t_1,e_{p,x})}) - F(S_{(t_2,e_{p,x})}) > \frac{10}{\epsilon}\beta F_{p,x}. \tag{2.6}$$

With the fact $t_3 - t_2 \ge \frac{\epsilon}{10}(e_{p,x} - t_3)$, the fact that the requests in $S_{[t_1,t_2]}$ arrive by time t_2, and (2.6), by applying Lemma 14 we have

$$F(S_{[t_1,t_2]}, t_3) \ge (\frac{\epsilon}{10})(\frac{10}{\epsilon})\beta F_{p,x} = \beta F_{p,x}. \tag{2.7}$$

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

$$F^*_{p,x} \ge F(S_{[t_1,t_2]}, t_3) \ge \beta F_{p,x} \ge \frac{1}{\gamma} F_{p,x}. \tag{2.8}$$

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

Corollary 1. *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 20. *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 13 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 14 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 20. 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 21. *For $0 < \epsilon \leq 1$, $F(\mathbb{T}_2) < \frac{2}{100}\text{LA-W}_{1+\epsilon}$.*

Proof. We apply Lemma 15. Let $E_{p,x} \in \mathbb{T}_2$ and $I_{p,x} = [\tau^{\beta k}_{p,x}, e_{p,x})$. By Corollary 1 we can set $\lambda = 500c$. By letting $\mathcal{B}_{p,x}$ be the set of events found for $E_{p,x}$ in Lemma 20, we can set $\kappa = \frac{1}{2c}(1-\beta k)$ and $\mu = \frac{49}{100}(1+\epsilon)$. Using Lemma 13 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. □

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

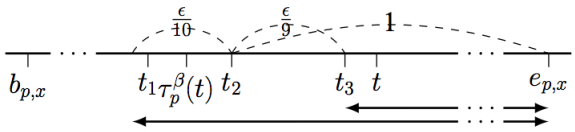

Figure 2.5: For an event $E_{p,x}$ in $\mathbb{T}_3$, during $[t_1, e_{p,x})$ OPT must make a unique broadcast for most events which end during $[t_3, e_{p,x})$.

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 $\tau^{\beta,i}_{p,x} - \tau^{\beta,i+1}_{p,x} < \frac{\epsilon}{10}(e_{p,x} - \tau^{\beta,i}_{p,x})$ because $E_{p,x} \notin \mathbb{T}_2$. Let $t_1 = \tau^{\beta,i+1}_{p,x}$ and $t_2 = \tau^{\beta,i}_{p,x}$. 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 2.5.

Recall that by Lemma 12, for any $E_{q,y} \in \mathcal{E}$, the optimal solution must broadcast page q on the interval $[\tau^{\beta}_q(e_{q,y}), e_{q,y})$ 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 $\tau^{\beta}_q(e_{q,y}) \in [t_1, e_{p,x})$ 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 $\tau^{\beta}_p(t) \geq t_1$ 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 $\tau^{\beta}_p(t) \leq \tau^{\beta}_q(t)$. Hence, for any $E_{q,y} \in \mathcal{E}$ we will have that $t_1 \leq \tau^{\beta}_p(e_{q,y}) \leq \tau^{\beta}_q(e_{q,y})$ and OPT broadcasts page q on $[t_1, e_{p,x})$.

Lemma 22. *For the event $E_{p,x} \in \mathbb{T}_3$, at any time $t \in [t_3, e_{p,x})$, $\tau^{\beta}_p(t) \geq t_1$.*

Proof. For the sake of contradiction assume that $\tau^{\beta}_p(t) < t_1$. Let $t' = \tau^{\beta}_p(t)$. 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,

$$F(S_{[t_1,t_2]}) = F(S_{[t_1, e_{p,x})}) - F(S_{(t_2, e_{p,x})}) > \frac{10}{\epsilon}\beta F_{p,x}. \tag{2.9}$$

By the definition of $t' = \tau^{\beta}_p(t)$, 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 14, $\frac{\epsilon}{9}F(S_{(t',t_2]}, e_{p,x}) \leq F(S_{(t',t_2]}, t)$. Thus we have,

$$F(S_{(t',t_2]}) = F(S_{(t',t_2]}, e_{p,x}) \leq \frac{9}{\epsilon}F(S_{(t',t_2]}, t) \leq \frac{9}{\epsilon}\beta F_{p,x}. \tag{2.10}$$

Knowing that $F(S_{(t',t_2]}) \geq F(S_{[t_1,t_2]})$, this is a contradiction to (2.9). □

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 23. *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 13); 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 13, we have

$$(1-\frac{34}{100}\epsilon)(1+\epsilon)(e_{p,x}-t_2) \leq |\mathcal{E}|. \tag{2.11}$$

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

$$|[t_1, e_{p,x}]| < (1+\frac{\epsilon}{10})(e_{p,x}-t_2). \tag{2.12}$$

As discussed previously, Lemma 22 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 13, it must be the case that

$$|\mathcal{E}| \leq |[t_1, e_{p,x})| + 1 \leq (1+\frac{\epsilon}{10000})|[t_1, e_{p,x})|. \tag{2.13}$$

By combining (2.11), (2.12), and (2.13), 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 18, 21 and 23 we have that $F(\mathcal{N}_2) \leq \frac{85}{100}\text{LA-W}_{1+\epsilon}$. The proof of Theorem 2 follows easily by combining this and lemmas 11 and 16.

2.5 Concluding Remarks

In this chapter, we gave the first scalable algorithm Latest Arrival time with Waiting (LA-W) for the objective of minimizing average flow time for unit-sized pages in broadcast scheduling. The algorithmic development depended crucially on the better understanding of LWF and its analysis. We currently do not know if the Rule 2 in LA-W is required to obtain a scalable algorithm. The algorithm of using only the Rule 1 will be more natural.

As mentioned in Remark 3, following our results, Bansal et al. gave another scalable algorithm which is $(1+\epsilon)$-speed $O(1/\epsilon^3)$-competitive. Moreover, their algorithm works also for varying-sized pages. The novelty of their analysis is a new interpretation of LAPS. They showed the algorithm LAPS is fractionally scalable

and gave an online rounding scheme that converts the fractional schedule into an integral schedule. Their work was extended to ℓ_k norms of flow time [55, 44].

There remain several interesting open problems in broadcast scheduling. Assume that all pages are unit-sized. The most interesting open problem is to improve the $O(\log n/\log\log n)$-approximation for the ℓ_1 norm, which is currently the best approximation known [12]. The only known complexity result is that the problem is NP-hard [46, 29].

Open Problem 1. *For the problem of minimizing average flow time for unit-sized pages in broadcast scheduling, give an approximation algorithm whose approximation ratio is* $o(\log/\log\log n)$*, or show that the problem is APX-hard.*

Another interesting open problem is regarding minimizing the maximum flow time. It is known that the problem is NP-hard [29]. The algorithm First In First Our (FIFO) is 2-competitive [29, 33]. Further it is known that for any $\epsilon > 0$, no randomized algorithm has a competitive ratio of $2 - \epsilon$ [34].

Open Problem 2. *For the problem of minimizing the maximum flow time for unit-sized pages in broadcast scheduling, give an approximation algorithm whose approximation ratio is strictly less than 2, or show that the problem is APX-hard.*

Chapter 3

Non-clairvoyant Scheduling with Arbitrary Speedup Curves

3.1 Introduction

We consider scheduling dynamically arriving jobs that have varying degrees of parallelizability (that is, some jobs may be sped up considerably when simultaneously run on multiple processors, while other jobs may be sped up by very little) on a multiprocessor system. The most obvious settings where this problem arises is scheduling multi-threaded processes on a chip with multiple cores/processors, and scheduling multi-processor applications in a server farm. We adopt the following general model of parallelizability, which was apparently first introduced in [43] and later used in [38, 40, 42, 90, 89, 26]: we have m identical fixed speed processors. Each job i arrives at time r_i, and consists of a sequence of phases. Each phase needs to finish some amount of work, and has a speedup function that specifies the rate at which work is processed in that particular phase (as a function of the number of processors assigned to the job). The speedup functions have to be nondecreasing (a job doesn't run slower if it is given more processors), and sublinear (a job satisfies Brent's Theorem: increasing the number of processors doesn't increase the efficiency of computation), but the functions are unconstrained otherwise.

The scheduler needs an *assignment policy* to determine how many processors are allocated to each job at each point in time. In order to be implementable in a real system, we require that this policy be *online*, since the scheduler will not in general know about jobs arriving in the future. This policy also ideally should be *nonclairvoyant*, since a scheduler usually does not know the size/work of a job when the job is released, nor the degree to which that job is parallelizable. So a non-clairvoyant algorithm only knows when jobs have been released and which have finished in the past, and how many processors have been allocated to each job at each point of time in the past. We will focus on the objective of minimizing the ℓ_k norms of flow time ($k \geq 1$), i.e., $\sqrt[k]{\sum_{i \in [n]} (C_i - r_i)^k}$. When $k \geq 2$, this objective is known to make a natural balance between average performance and fairness. See Section 1.2.

To recap, we address the problem of designing an online non-clairvoyant assignment policy that is competitive for the objective of the ℓ_k norm of the flow time for jobs with arbitrary speed-up curves.

Context and Developing the Right Intuition

To get a feel for the problem, let us consider two special cases. Consider first the case that all jobs are *fully parallelizable* (i.e., increasing the number of processors assigned to a job by a factor f reduces the time required by a factor of f), which is essentially equivalent to having a single processor. In the initial paper popularizing resource augmentation analysis [69], the algorithm Shortest Elapsed Time First (SETF) which shares the processors evenly among all jobs that have been processed the least (which necessarily are the later arriving jobs) was shown to be scalable for the ℓ_1 norm of flow. This was generalized by [8] to show that SETF is scalable for all ℓ_k norms of flow for $1 \leq k < \infty$. So intuitively, in the case of parallel work, the "right" algorithm is independent of the norm, equivalently the "right" algorithm for the ℓ_1 norm extends to all ℓ_k norms.

Now consider the more general case of jobs with arbitrary speed-up curves, but only for the ℓ_1 norm. [38] showed that the assignment policy EQUI that shares the processors evenly among all active jobs is $(2+\epsilon)$-speed $O(1)$-competitive for average flow time. Subsequently, [42] introduced the algorithm Latest Arrival Processor Sharing (LAPS), which shares the processors evenly among the latest arriving constant fraction of the jobs, and showed that LAPS is scalable for average flow time. The intuition behind LAPS was to mimic SETF, by giving more processors to later arriving jobs, but to spread the processing power more evenly in case that the latest arriving jobs are sequential (their processing rate does not increase even if they are assigned more processors). In this special case, the "right" strategy for arbitrary speed-curves is basically the same as the "right" strategy for parallel work.

Given these two special cases, it would be natural to think that LAPS should be scalable for the problem of minimizing ℓ_k norms of flow for jobs with arbitrary speed-up curves. However, we show in Section 3.3 that LAPS is not $O(1)$-speed $O(1)$-competitive for the ℓ_k norm of flow when $k \geq 2$.

Considering why LAPS fails to be $O(1)$-competitive for ℓ_k norms, even with faster processors, offers insight into how to design a scalable algorithm. Let us consider the most popular case of the ℓ_2 norm. Just as the total flow time $\sum_i F_i$ is the integral over time of the number of jobs unfinished at that time, the sum of the squares of the flow times $\sum_i F_i^2$ is proportional to the integral over time, of the *ages* of the jobs at that time. The key observation in [8] was that if SETF, with a $(1+\epsilon)$-speed processor, has unfinished jobs of a particular age, then any schedule on a unit speed processor must have a comparable number of unfinished jobs that are at least as old: this observation implies the competitiveness of SETF for all ℓ_k norms rather directly. The lower bound instance in Section 3.3 shows that for jobs with arbitrary speed-up curves, LAPS may have devoted too much processing to sequential jobs in the past, and hence it may have many more old jobs than is necessary. And as LAPS focuses on new jobs, these old jobs will remain unfinished, driving up the ℓ_2 norm of flow for

LAPS relative to optimal. This lower bound instance, and the simple instance of the stream of unit work jobs, suggests that the online algorithm must give a greater share of the processing power to older jobs for ℓ_2 norm of flow time for jobs with arbitrary speed-up curves.

Our algorithm addresses this by distributing the processors in proportion to the age of the jobs, which is the rate at which that job is currently increasing the ℓ_2 norm of flow (more precisely, the sum of squares of flow time, $\sum_{i\in[n]}(C_i - r_i)^2$) Combining this intuition with an idea used in [42] to focus only on a fraction of the recent jobs, we design an algorithm WLAPS that is scalable for the ℓ_2 norm of flow. Essentially, WLAPS distributes the processors to the latest arriving constant fraction of jobs. However, the proportion of resources a job gets is related to the age of the job. The algorithm WLAPS naturally extends to ℓ_k norms for $k \geq 1$.

3.1.1 Our Results

Our main result in this chapter is the following:

Theorem 3. *There exists a $(1+12\epsilon)$-speed $O(\frac{k}{\epsilon^{2k+1}})$-competitive algorithm for the ℓ_k-norm objective for each fixed k where $0 < \epsilon \leq \frac{1}{24k}$.*

Note that our result uses resource augmentation. It is known that no algorithm is $O(1)$-competitive without resource augmentation for ℓ_k norms, $k \geq 2$ [8]. The competitive ratio of WLAPS, which is shown in our analysis, grows with k. This is supported by the lower bound on the competitive ratio for the ℓ_∞-norm [88]. We conjecture the competitive ratio of WLAPS grow linearly as k grows.

3.1.2 Related Results

Consider first the case that all the work is fully parallelizable and and the ℓ_1 norm. It is well known that the online clairvoyant algorithm Shortest Remaining Processing Time is optimal. The competitive ratio of any deterministic non-clairvoyant algorithm is $\Omega(n^{1/3})$, and the competitive ratio of every randomized algorithm against an oblivious adversary is $\Omega(\log n)$ [83]. A randomized version of the Multi-Level Feedback Queue algorithm has a matching asymptotic competitive ratio [70, 18].

Chan et al. [26] consider the problem of non-clairvoyant scheduling of jobs with varying degrees of parallelizability on a multiprocessor, where each machine can be scaled at a different speed. They give a $O(\log m)$-competitive algorithm for the problem of minimizing the sum of average flow time plus energy, where the power function varies as s^α for constant α, under some assumptions about the jobs.

3.2 Formal Problem Statement and Notation

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

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

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

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

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

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

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

3.3 Limitation of Latest Arrival Processor Sharing (LAPS) for the ℓ_k-norms

In this section, we show that LAPS has a large competitive ratio for minimizing the ℓ_k-norms of flow time for jobs with arbitrarily speedup curves. This is the case even when LAPS is given any constant speed. The main idea of constructing the adversarial example is to repeatedly request fully sequential jobs to prevent LAPS from working on parallel jobs. Consequently, LAPS wastes its processing power procrastinating parallel jobs substantially; unlike in L_1-norm flow time, these delayed jobs will cause a huge penalty.

Theorem 4. *Consider any fixed integer $k \geq 2$ and any fixed speed $s \leq 1$. Further consider any constant $0 < \beta \leq 1$ which parameterizes LAPS. Then the algorithm LAPS is not $O(1)$-competitive with speed s for the problem of minimizing the ℓ_k norm of flow time where jobs have arbitrarily speed up curves.*

Proof. Recall that LAPS works on only β fraction of alive jobs which arrived most recently; for the definition of LAPS, see Section 1.4. Let σ denote the adversarial instance. For simplicity of our argument, suppose that LAPS is given an integer speed $s > 1$. Let $\mathrm{LAPS}_s(\sigma)$ and $\mathrm{OPT}_1(\sigma)$ denote the k^{th} power of flow time for the given instance σ; the subscript s and 1 are used to denote the speed LAPS and OPT are given, respectively. Let $M > 0$ be a sufficiently large integer which will be defined later. The instance σ is constructed as follows.

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

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

3.4 Non-clairvoyant Algorithm Weighted LAPS (WLAPS)

We first describe our non-clairvoyant preemptive algorithm WLAPS for Weighted Latest Arrival Processor Sharing. As can be deduced from its name, WLAPS is inspired by LAPS [42], a scalable algorithm for minimizing the total flow

time of the jobs (i.e. when $k = 1$). Before we describe our algorithm, let us introduce some notation. First and foremost, we will assume that the algorithm WLAPS is given a speed-up of a factor of s. In other words, we can assume that WLAPS is given a s-speed processor while the optimal adversary is given only a unit-speed processor. Let β be a scaling parameter that determines the fraction of weight we consider at any instant of time. The speed-up s will depend on β, and we will fix this parameter later.

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

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

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

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

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

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

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

3.5 Analysis

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; see Section 1.5.1. 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, it can be easily seen that in the arbitrary speedup curves setting, no non-clairvoyant algorithm is local competitive for the ℓ_k norm of flow time for any integer $k \geq 1$. To avoid a local argument we use a potential function analysis. This has recently become popular in scheduling theory [38, 42, 55, 24]. For a quick overview of potential functions in online scheduling, see Section 1.5.2. In this chapter we introduce an interesting potential function which takes insights from [42, 55, 24, 15]. The potential function of [55] is most closely related to our potential function.

In Section 3.5.1, we first show that we only need to consider "extreme" jobs. Namely, we will show that it suffices to consider only the case where each job is fully parallelizable or sequential. We then formally define the potential function that will be used, and give an intuition behind it, in Section 3.5.2 and 3.5.3, respectively. The main analysis is given in Section 3.5.4.

3.5.1 Restricted Instances are Sufficient

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

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

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

occur before or after t). Let the optimal solution devote p_i^o processors towards doing this work w. Notice that the definition of $[t^*, t^* + dt^*)$ and p_i^o imply that $\Gamma(p_i^o)dt^* = w = \Gamma(p_i^a)dt$, which in turn implies that

$$\frac{\Gamma(p_i^o)}{\Gamma(p_i^a)} = \frac{dt}{dt^*} \tag{3.1}$$

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

$$\frac{p_i^o}{p_i^a} \geq \frac{\Gamma(p_i^o)}{\Gamma(p_i^a)} = \frac{dt}{dt^*} \tag{3.2}$$

and therefore an optimal solution for $\mathcal{I}'$ can fit in the w' amount of fully parallelizable work at same time interval $[t^*, t^* + dt^*)$ when the optimal solution for $\mathcal{I}$ worked on the corresponding w amount of i. Here, the equation (3.2) follows from the sublinear nature of the speed-up function.

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

$$1 \geq \frac{\Gamma(p_i^o)}{\Gamma(p_i^a)} = \frac{dt}{dt^*} \tag{3.3}$$

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

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

3.5.2 Potential Function

We assume without loss of generality that all jobs arrive at distinct times. Let $N_o(t)$ be the set of released, yet unsatisfied jobs in the optimal solutions schedule. Let $x_i(t)$ denote the amount of parallel work for job i which OPT has done but WLAPS has not at time t; if WLAPS have processed more parallel work for job i than OPT, then $x_i(t)$ is zero. An analogous quantity $y_i(t)$ is defined for sequential work of job i. Let σ_i be the total sequential work for job i. To analyze WLAPS we will use a potential function analysis. For this analysis we will define a potential function $\Phi(t)$. The potential function $\Phi(t)$ will not increase when a job arrives nor when a job is completed by OPT or WLAPS. Further, $\Phi(0) = \Phi(\infty) = 0$. During all times $[t, t+\mathrm{dt}]$ when no job arrives or is completed, it will be shown that $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq c\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$, where c is some constant. Here $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) = \sum_{i \in N_a(t)} w_i(t)$ and $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) = \sum_{i \in N_o(t)} w_i(t)$. If $\Phi(t)$ meets each of these conditions then WLAPS is c competitive. Our potential function $\Phi(t)$ is defined as follows:

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

3.5.3 Intuition Behind the Potential Function

Let $\Phi_1(t)$ be the first term of $\Phi(t)$ and $\Phi_2(t)$ be the second term of $\Phi(t)$. The boundary conditions of our potential function are satisfied trivially. When job i arrives at time t, the potential function has no change since $t - a_i = 0$ and $x_i = 0$ on arrival. The optimal solution completing a job has no effect on the potential function. When algorithm completes a job i the potential function can only decrease, since all terms are positive.

Before analyzing the change in Φ, we discuss high level intuition of the analysis. As stated, in [8] it was shown that SRPT and SJF are scalable algorithms when all jobs have only one phase which is fully parallelizable. To prove this, the authors used a local argument. They showed that at each time t, the sum of the age^{k-1} of the jobs that are still alive under the algorithm's schedule is at most a constant c times the corresponding value for those that are alive under the optimal schedule. When jobs can have a varying degree of parallelizability, this local property no longer holds. This is why we resort to a potential function based argument. When the algorithm's current costs are less than c times the optimal's, the algorithm saves some into a bank account so that when the algorithm's current costs are higher than this, he can pay for them by withdrawing these reserves. The potential function measures how much

is currently in the bank. The proof must show that at each point in time, these costs balance, namely that $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq c\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$.

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

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

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

3.5.4 Main Analysis

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

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

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

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

$$\frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t) \quad \leq \quad -\frac{1}{\epsilon}\left(\sum_{j \in \mathcal{P}'(t) \setminus N_o(t)} s\frac{w_j(t)}{\beta w(t)}\right) \cdot \left(\sum_{i \in N_a(t) \setminus N'_a(t)} W_i(t)\right) \tag{3.4}$$

For the final analysis we need to obtain an upper bound on (3.4). The following propositions and lemmas will be useful tools. The two following easy propositions were shown in [55].

Proposition 1. *For any job i, $\sum_{r_j \geq r_i, j \in N_a(t)} x_j(t) \leq t - r_i$.*

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

Proposition 2. *For any job i, $y_i(t) \leq t - r_i$.*

The following proposition is easily obtained by applying proposition 1.

Proposition 3.

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

By Proposition 1, we can bound a part of (3.4),

$$\begin{aligned}
&\sum_{i\in N_a(t)\setminus N'_a(t)} W_i(t) = \sum_{i\in N_a(t)} W_i(t) - \sum_{i\in N'_a(t)} W_i(t)\\
&\geq \sum_{i\in N_a(t)} W_i(t) - (1+\frac{1}{\epsilon})^{k-1} \sum_{i\in N'_a(t)} w_i(t)\\
&= \sum_{i\in N_a(t)} W_i(t) - \beta(1+\frac{1}{\epsilon})^{k-1} \sum_{i\in N_a(t)} w_i(t)\\
&\geq \Big[1-\beta(1+\frac{1}{\epsilon})^{k-1}\Big] \sum_{i\in N_a(t)} W_i(t) \qquad (3.5)
\end{aligned}$$

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

$$\begin{aligned}
&\sum_{j\in\mathcal{P}'(t)\setminus N_o(t)} \frac{w_j(t)}{\beta w(t)}\\
\geq\ & \frac{1}{\beta w(t)}\Big[\sum_{j\in N'_a(t)} w_j(t) - \sum_{j\in\mathcal{S}(t)} w_j(t) - \sum_{j\in N_o(t)} w_j(t)\Big]\\
=\ & 1 - \frac{1}{\beta w(t)}\Big[\sum_{j\in\mathcal{S}(t)} w_j(t) + \sum_{j\in N_o(t)} w_j(t)\Big] \qquad (3.6)
\end{aligned}$$

From (3.4), (3.5) and (3.6), we have

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t) \ \leq\ & -s\frac{1}{\epsilon}\Big(1-\beta(1+\frac{1}{\epsilon})^{k-1}\Big)\cdot\\
& \Big(1-\frac{1}{\beta w(t)}\Big[\sum_{j\in\mathcal{S}(t)} w_j(t) + \sum_{j\in N_o(t)} w_j(t)\Big]\Big) \sum_{i\in N_a(t)} W_i(t) \qquad (3.7)
\end{aligned}$$

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

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

Our goal is now to bound $\mathtt{dt}\Phi_2(t)$ by $\mathtt{dt}$WLAPS(t). To this end, we partition jobs in $N_a(t)$ into 'old' jobs $\mathcal{O}(t)$ and 'young' jobs $\mathcal{Y}(t)$. Recall that σ_i is the total sequential work for job i. A job $i \in N_a(t)$ is in $\mathcal{Y}(t)$ if $(t-r_i) \leq k(\frac{2}{\epsilon})^{2k+1}\sigma_i$; otherwise, the job is in $\mathcal{O}(t)$. The increase in $\Phi_2(t)$ due to jobs in $\mathcal{Y}(t)$ will be charged directly to the optimal solution's cost in the following lemma. This idea is similar to that given in [55] and the proof can be found in the appendix.

Lemma 24. $\displaystyle\int_0^\infty \sum_{i\in\mathcal{Y}(t)} k(k-1)(t-r_i)^{k-2}y_i(t)\mathrm{dt} \leq k^{k+1}(\frac{2}{\epsilon})^{k(2k+1)}\mathrm{OPT}.$

The change in $\Phi_2(t)$ due to old jobs is at most $(\frac{2}{\epsilon})^{2k+1}\sum_{i\in\mathcal{O}(t)} k(k-1)(t-r_i)^{k-2}\sigma_i(t) \leq \sum_{i\in\mathcal{O}(t)} k(t-r_i)^{k-1} \leq \sum_{i\in N_a(t)} k(t-r_i)^{k-1}$, by definition of old

jobs. Thus after excluding the young jobs, the total increase in $\Phi(t)$ due to the change in time is, $\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq 2\sum_{i\in N_a(t)} W_i(t)$.

Completing the Analysis: For the final analysis, we add the upper bound on the change for each of the cases we studied in the previous section. Let $\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)$ denote the change (rate) that is obtained from $\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t)$ by removing the increase due to time elapse for the young jobs. We will show that $\mathtt{dt}\mathrm{WLAPS}(t)+\mathtt{dt}\Phi'(t) \leq 2(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$. Then we will have

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

The first inequality comes from Lemma 24, which gives an upper bound on the total increase due to time elapse over all times for the young jobs. Recall that we have been considering the objective of minimizing the sum of the kth power flowtime. Since we are actually interested in ℓ_k-norms we take the outer kth root, which proves Theorem 3.

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

$$\mathtt{dt}\mathrm{WLAPS}(t) + \mathtt{dt}\Phi'(t) \leq$$

$$\Big(3+\frac{1}{\epsilon}\Big)\sum_{i\in N_a(t)} W_i(t) \tag{3.8}$$

$$-s\frac{1}{\epsilon}\Big(1-\beta(1+\frac{1}{\epsilon})^{k-1}\Big)\Big(1-\frac{1}{\beta w(t)}\Big[\sum_{j\in\mathcal{S}(t)} w_j(t) + \sum_{j\in N_o(t)} w_j(t)\Big]\Big)\sum_{i\in N_a(t)} W_i(t) \tag{3.9}$$

$$-(\frac{2}{\epsilon})^{2k+1}\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t) \tag{3.10}$$

$$+(\frac{2}{\epsilon})^{2k+1}\mathtt{dt}\mathrm{OPT}(t) \tag{3.11}$$

We remind the reader that (3.9) and (3.10) come from the change due to WLAPS's processing jobs in a parallel phase and jobs in a serial phase, respectively. Recall that $\beta = \epsilon^k$ and $s \geq 1 + 12\epsilon$, where $0 < \epsilon \leq \frac{1}{24k}$. We consider three cases.

Case (a): $\sum_{i\in N_o(t)} w_i(t) \geq \epsilon\beta\sum_{i\in N_a(t)} w_i(t)$. This is the easiest case where OPT has jobs whose total weight is comparable to that of the jobs in WLAPS's

queue. In this case, by Proposition 3 and simple algebra, (3.8) + (3.11) $\leq \frac{4}{\epsilon}(\frac{2}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}A(t)$ + (3.11) $\leq 2(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$.

Case (b): $\sum_{i\in\mathcal{S}(t)\setminus N_o(t)} w_i(t) \geq \epsilon\beta\sum_{i\in N_a(t)} w_i(t)$. In this case, the decrease due to WLAPS's processing jobs in a sequential phase will offset other positive terms. Again, by Proposition 3 and an easy calculation, (3.8)+(3.10) $\leq \frac{4}{\epsilon}(\frac{2}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}A(t) - \epsilon\beta(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}A(t) \leq 0$. And clearly, (3.11) $\leq 2(\frac{2}{\epsilon})^{2k+1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$.

Case (c): Neither case (a) nor case (b). Then we have $\sum_{i\in N_o(t)} w_i(t) + \sum_{i\in\mathcal{S}(t)} w_i(t) \leq 3\epsilon\beta\sum_{i\in N_a(t)} w_i(t)$. This is the case where most (in terms of weights) of the jobs WLAPS are processing are in a parallel phase. By simple algebra, (3.8) + (3.9) $\leq \frac{1+3\epsilon}{\epsilon}\sum_{i\in N_a(t)} W_i(t) - \frac{1}{\epsilon}s(1-\epsilon(1+\frac{\epsilon}{k})^{k-1})(1-3\epsilon)\sum_{i\in N_a(t)} W_i(t) \leq 0$. In all cases the desired inequality holds, and this completes the analysis.

3.6 Concluding Remarks

In this chapter, we considered a natural extension of LAPS, WLAPS for Weighted Latest Arrival Processor Sharing and showed it is scalable for ℓ_k-norms of flow time when jobs have arbitrary speed up curves. We believe that the competitive ratio of WLAPS increases as k does, since any algorithm has a competitive ratio of $\Omega(\log n)$ for the ℓ_∞-norm, even with any constant speed-up [88]. However, we conjecture that the competitive ratio should not grow exponentially as k grows.

Conjecture 1. *Consider the problem of minimizing ℓ_k norms of flow time for jobs with arbitrary speed up curves. Assuming that WLAPS is given a fixed speed $s > 1$, its competitive ratio is $\theta(k)$.*

Even for the ℓ_1-norm, the algorithm LAPS takes a parameter β as input. For LAPS to be scalable, the parameter must depend on the speed it is given. Edmonds conjectures that any non-clairvoyant deterministic algorithm, to be scalable, must know the speed it is given.

Conjecture 2. *[39] For the problem of minimizing the total flow time for jobs with arbitrary speed up curves, any deterministic non-clairvoyant algorithm, without the knowledge of the speed it is given, is not scalable.*

Chapter 4

Scheduling on Unrelated Machines

4.1 Introduction

In this chapter, we study online scheduling problems on heterogeneous machines. In the *online* setting job i is released at time r_i and this is the first time the schedule becomes aware of the job. We will focus on the objective of minimizing ℓ_k norms of flow time, $\sqrt[k]{\sum_{i\in[n]}(C_i - r_i)^k}$, where n is the number of jobs and C_i is job i's competition time [1]. The quantity $C_i - r_i$ is called job i's flow time or response time and measures the time job i waits until it is completed. As discussed in Section 1.2, ℓ_k norms ($k \geq 2$) can be used to make a balance between average performance and fairness. When $k = 1$, the objective is often called total flow time or equivalently average flow time. In this chapter we will study probably the most general heterogeneous machines setting, which is known as the unrelated machines setting. To give the reader a feel of this setting, we will discuss serval scheduling settings in increasing order of their complexity and finally the unrelated machines setting.

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

The simplest multiple machines setting is where all machines are *identical*. That is, each job has the same processing time on all machines and any job can be scheduled on any machine. Average flow time has been studied extensively in the identical machine model [78, 6, 5, 84, 18]. Even in this simple case, the

[1] Later the notation will be changed on heterogeneous machines. Further the more general objective of weighted ℓ_k norms of flow time will be considered. These will be formally described later in this chapter.

best competitive ratio is $O(\min(\log P, \log n/m))$ and there exists a matching lower bound. Here P is the ratio of the biggest job's processing time to the smallest job's processing time. When a strong lower bound exists, a popular model of analysis is the *resource augmentation* model [69, 85]. In this model, an algorithm A with processors of speed s is compared to the optimal solution with processors of speed 1. For any jobs instance, if the objective value achieved by A is within a factor c of that by the optimal solution, the algorithm A is said to be s-speed c-competitive for the objective. An algorithm that is $(1+\epsilon)$-speed $O(1)$-competitive algorithm is said to be *scalable*, since it is $O(1)$-competitive when given the smallest amount of resources over the adversary. See Section 1.3.3 for further elaboration on this model. In the identical machines setting, [31] gave a scalable algorithm.

In practice machines may not be identical. For instance, machines may have different speed processors. One model that captures this situation is the *related* machines model. Here, each machine x has some speed s_x. Job i requires $\frac{p_i}{s_x}$ time to complete if it is assigned to machine x. The related machines model is of practical interest. However, finding good algorithms has been difficult. There are few positive results known [50, 51]. The best known algorithm without speed augmentation is $O(\log^2 P)$-competitive [51].

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

Designing algorithms that are competitive for average flow time on unrelated machines has been difficult. In [52] it was shown that no online algorithm can have a bounded competitive ratio for minimizing the average flow time on unrelated machines without resource augmentation. This lower bound was shown in the restricted case where there are only 3 machines and jobs have either unit size on a machine or infinite size. This example shows that simply restricting jobs to certain machines makes the problem much harder than the related or identical models. Another challenge when designing and analyzing algorithms for unrelated machines is that different schedules can do different amounts of work to satisfy the same set of requests. Here scheduling mistakes are significantly more costly if the optimal solution is doing less work than the algorithm. This differs from the standard scheduling setting where any schedule does the same amount of

work to satisfy the same set of jobs. See [32, 55] for other examples. Until recently no non-trivial algorithms were known in the online setting for average flow time. In a breakthrough result a $(1+\epsilon)$-speed $O(1)$-competitive algorithm was given for any fixed $0 < \epsilon \leq 1$ [24]. This was also the first $O(1)$-speed $O(1)$-competitive algorithm shown for the more restricted related machines setting.

For the ℓ_k-norm of flow time it is well known that without resource augmentation that every deterministic algorithm is $n^{\Omega(1)}$-competitive when $1 < k < \infty$, even on a single machine [8]. This contrasts with average flow time, where SRPT is an optimal algorithm on a single machine. It was shown in [9] that SRPT is a scalable algorithm to minimize the ℓ_k norm of flow time on a single machine for all k. Later a scalable algorithm to minimize the ℓ_k norm of flow time on identical machines was given for any k [31]. There are no known *offline* approximation algorithms for minimizing the ℓ_k norms of flow time for any $k \in [1, \infty)$ without resource augmentation on unrelated machines; note that the ℓ_1 norm is equivalent to average flow time. Further, there are no known non-trivial online algorithms for minimizing the ℓ_k norm of flow time even on related machines with any amount of resource augmentation where $1 < k < \infty$.

In this chapter we will be considering the *weighted* ℓ_k norms of flow time of a non-migratory schedule. This is a generalization of the ℓ_k norm objective. Here a job i has a weight w_{ix} when assigned to machines x. The goal of the scheduler is to minimize $\sqrt[k]{\sum_{i\in[n]} w_{iM(x)}(C_i - r_i)^k}$ where $M(x)$ is the machine job x is assigned to. For the weighted ℓ_1 norm of flow time, it was recently shown that no algorithm can be $O(1)$-competitive without resource augmentation on a single machine [7]. It is well known that the algorithm Highest Density First (HDF) is $(1+\epsilon)$-speed $O(1)$-competitive for the ℓ_1 norm of weighted flow time on a single machine [19]. The algorithm HDF is a natural generalization of SRPT where the scheduler always processes the job i such that $\frac{w_i}{p_i}$ is minimized. In [9], it was shown that HDF is also $(1+\epsilon)$-speed $O(1)$-competitive for the ℓ_k norms of weighted flow time for $k \geq 1$ on a single machine. [31] gave a scalable algorithm for minimize the weighted ℓ_k norms of flow time on identical parallel machines. The algorithm of [24] for minimizing the ℓ_1 norm of flow time on unrelated machines also considers the case where jobs have weights. Their algorithm is $(1+\epsilon)$-speed $O(1)$-competitive algorithm for the weighted ℓ_1 norm of flow time.

4.1.1 Our Results

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

algorithm is immediate-dispatch and non-migratory. More specifically, we show the following theorem.

Theorem 5. *For any integer $k \geq 1$ and for any $0 < \epsilon \leq 1$, there exists a $(1+\epsilon)$-speed $O(\frac{k}{\epsilon^{2+2/k}})$-competitive algorithm for minimizing weighted ℓ_k-norm of flowtime on unrelated machines. In particular, the algorithm is immediate dispatch and non-migratory.*

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

Theorem 6. *For the problem of minimizing ℓ_k norm of flow time on unrelated machines, any randomized immediate-dispatch non-migratory online algorithm, with any speed $s \geq 1$ given, has competitive ratio $\Omega(\frac{k}{s})$.*

It is important to note that our results translate into the problem of minimizing the ℓ_k norm of stretch. The ℓ_k norm of stretch is $\sqrt[k]{\sum_{i\in[n]} \left(\frac{C_i - r_i}{p_{iM(i)}}\right)^k}$ for some fixed schedule. There is a similar lower bound for the ℓ_k norm of stretch as there is for the ℓ_k norm of flow time on a single machine and jobs have no weight [8]. The ℓ_k norm of stretch can be reduced to the weighted ℓ_k norm of flow time by setting $w_{i,x} = (\frac{1}{p_{i,x}})^k$. Stretch is a popular metric and is used to capture the fact that users expect long jobs to take more time than short jobs. That is, a user is likely to expect to wait for a job to complete in proportion to the job's processing time. This objective is commonly considered in database applications [20, 21]. By using this reduction, our result extends to the stretch setting.

4.1.2 Our Techniques

Our analysis is based on a new novel potential function. The potential function we introduce takes insights from [55, 24]. Most closely related to our potential function is that given in [24] which was used to give a scalable algorithm for minimizing average flow time on unrelated machines. In [24], the algorithm used was to place a job on the machine which increases the ℓ_1 norm of flow time the least and then each machine runs HDF on the jobs assigned to it. Although this algorithm is simple and natural, the potential function used to prove its competitiveness is quite non-trivial. The main idea of the potential function used in [24] is to keep track of the volume of remaining work of the algorithm as compared to the adversary. However, this potential cannot be used in the ℓ_k norm setting because the *ages* of jobs contribute to the increase in the algorithm's objective when $k > 1$, not just the volume of unfinished jobs. Indeed, the increase in the ℓ_k-norm objective at any time grows in proportion to the time for which each job has been unsatisfied. This is in contrast to the ℓ_1-norm where the increase in the objective at each time only depends on the

number of unsatisfied jobs. Any potential function which does not incorporate the amount of time each job in the algorithm's queue has been unsatisfied for will not be useful for upper-bounding the algorithm's ℓ_k-norm flow time.

In this chapter, we show a novel potential function that incorporates the volume of remaining work and ages of jobs in the algorithm's queue as compared to the adversary' queue. The potential function combines the ages and volume of jobs in an interesting way. We tried natural generalizations of the potential function of [24] for the ℓ_k norm, however these generalizations do not seem to lead to a $O(1)$-speed $O(1)$-competitive algorithm for the ℓ_k norm of flow time when $k > 1$. Like [24], our algorithm runs HDF on each individual machine. However the machine assignment our algorithm uses of jobs comes from the potential function we derive. We tried to analyze the natural algorithm that assigns a job to a machine that increases the ℓ_k norm flow time the least. However, we were unable to find a potential function that can show this algorithm to be $O(1)$-competitive with speed less than $k + \epsilon$.

It is worth noting that in the analysis of [24], the optimal solution was restricted to using HDF on each machine. In our analysis, we will be assuming an *arbitrary* optimal solution. We note that designing a potential function is quite non-trivial for minimizing ℓ_k-norm flow time even on a single processor for any $k \geq 1$. One novelty of our result is a potential function-based argument showing that HDF is scalable on a single machine for any fixed $k \geq 1$. Some lemmas we present in the analysis, which compare our algorithm's status with the arbitrary adversary's status, may be of independent interest.

4.2 Formal Problem Statement and Notation

The problem we consider is formally defined as follows. There are m machines, and n jobs arrive over time in online fashion. Hence the scheduler becomes aware of each job only when it arrives. Each job i can have a different weight w_{ix} and a different processing time p_{ix} on each machine x it is assigned to. Such quantities are revealed to the scheduler upon the job i's arrival. Our goal is to find an online schedule that is immediate-dispatch and non-migratory to minimize weighted ℓ_k norms of flow time.

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

For the rest of this chapter, if we say ℓ_k norm flow time, we mean the weighted ℓ_k norm flow time, i.e. we may omit the term 'weighted'. To bound the ℓ_k norm flow time, we will drop the outer kth root and focus on bounding $\sum_{i\in[n]} w_{i,M(i)}(C_i - r_i)^k$, the integral kth power flow time. To do this, we will focus on bounding the *fractional* kth power flow time. It is known that some algorithm is s-speed c-competitive for the fractional k power flow time can be translated $(1+\epsilon')s$-speed $c(1+\frac{1}{\epsilon'})$-competitive for the integral kth power flow time, by increasing the speed augmentation by an extra factor of $(1+\epsilon')$. Henceforth, we will focus on bounding the kth power fractional flow time.

The total kth power fractional flow time of a schedule is defined to be $\int_{t=0}^{\infty} \sum_{i\in U(t)} k(t-r_i)^{k-1} p_i^A(t) d_{iM(i)} \mathrm{dt}$, where $U(t)$ is the set of unsatisfied jobs in the given schedule at time t and $M(i)$ is the machine job i is assigned to. Equivalently, the fractional kth power flow time of a schedule is $s\int_{t=0}^{\infty} \sum_{i\in J(t)} (t-r_i)^k d_{iM(i)} \mathrm{dt}$ where $J(t)$ is the set of at most m jobs being processed at time t. There are at most m jobs because a machine can be processing at most 1 job at a time. We will focus mostly on the second definition. We will say that $s\sum_{i\in J(t)}(t-r_i)^k d_{iM(i)}$ is $\frac{\mathrm{dA}}{\mathrm{dt}}$, the increase rate of the fractional flow time of are algorithm during $[t, t+\mathrm{dt}]$. Likewise, $\sum_{i\in J^*(t)}(t-r_i)^k d_{iM^*(i)}$ is $\frac{\mathrm{dO}}{\mathrm{dt}}$ the increase rate of the optimal solutions flow time during $[t, t+\mathrm{dt}]$ where $J^*(t)$ are the jobs OPT works on during $[t, t+\mathrm{dt}]$ and $M^*(i)$ is the machine OPT assigns job i to.

4.3 Algorithm and Potential Function

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

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

immediately assigned to a machine x which minimizes the following expression,

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

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

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

$$\begin{aligned}
&\Phi_x(t)\\
&=\sum_{i\in A_x(t)} d_{ix}\int_{\tau=0}^{p_i^A(t)}\Big(\epsilon(t-r_i)+V_{>d_{ix}}^{A_x(t)}+\tau\Big)^k\mathrm{d}\tau &(4.1)\\
&-\sum_{i\in O_x(t)} d_{ix}\int_{\tau=0}^{p_i^O(t)}\Big(\epsilon(t-r_i)+V_{>d_{ix}}^{A_x(t)}+\tau\Big)^k\mathrm{d}\tau &(4.2)\\
&-\sum_{i\in A_x(t)} d_{ix}\int_{\tau=0}^{p_i^A(t)}\Big[\Big(\epsilon(t-r_i)+V_{>d_{ix}}^{A_x(t)}+V_{>d_{ix}}^{O_x(t)}+\tau\Big)^k &(4.3)\\
&\qquad\qquad-\Big(\epsilon(t-r_i)+V_{>d_{ix}}^{A_x(t)}+\tau\Big)^k\Big]\mathrm{d}\tau
\end{aligned}$$

For the sake of analysis we let $\Phi_{x,1}(t)$, $\Phi_{x,2}(t)$ and $\Phi_{x,3}(t)$ denote the part of $\Phi_x(t)$ in (4.1), (4.2) and (4.4) respectively. The second term $\Phi_{x,2}$ is included to relate the algorithm's queue to the optimal solution's queue. The third term $\Phi_{x,3}$, with the second term, is designed to eliminate the changes in $\Phi_{x,1}$ due to jobs arriving and being placed on machines by the algorithm.

Overview of Analysis

Notice that Φ does not increase when jobs are completed by OPT and our algorithm. The potential function Φ is designed such that its total increase due to the arrival of jobs is at most 0. (Also observe that Φ is 0 before jobs arrive and 0 after all jobs are completed by OPT and the algorithm) We call this the non-continuous change in Φ because it happens only at instantaneous times when jobs arrive or are completed. This change is analyzed in Section 4.4.

We then bound the change in Φ during an infinitesimal time where no jobs arrive or are completed. This is the continuous change in Φ. We will show that the total continuous change over all times is at most $-\gamma A + \delta \mathrm{OPT}$ where γ and δ are constants that can depend on k. These are all the events that effect Φ. Knowing that $\Phi = 0$ at time 0 and time ∞ (any time after all jobs are completed by our algorithm and the adversary), we have the total change in Φ is at most 0. Thus, knowing that $-\gamma A + \delta \mathrm{OPT}$ is an upper bound on the total continuous change of Φ and that non-continuous changes do not increase Φ, we have that $0 \leq -\gamma A + \delta \mathrm{OPT}$. This implies that $A \leq \frac{\delta}{\gamma}\mathrm{OPT}$. This will complete our analysis.

The analysis of the continuous change, due to the change in time, processing by OPT and the algorithm, is given in Section 4.5. After introducing several useful analysis tools, we will analyze $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,1}(t)$, $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,2}(t)$ and $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,3}(t)$, respectively.

4.4 Upperbound: Non-continuous Changes

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

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

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

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

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

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

4.5 Upperbound: Continuous Changes

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

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

In the analysis, we will be mostly focusing on each specific machine x and bounding the continuous changes in Φ_x. Hence we may drop "x" from the

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

Before addressing the continuous change in Φ_x, in Section 4.5.1 we show some useful lemmas that will be used throughout the analysis; for better readability, some of the lemmas will be proven in the following section. For the analysis of $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_x(t)$, we analyze each of $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,1}(t)$, $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,2}(t)$ and $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,3}(t)$.

4.5.1 Analysis Tools

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

Lemma 25. *Consider any given instance $\mathcal{I}$ where all jobs are assigned to a single machine. Then for any valid schedule B, with speed s' given, on the instance $\mathcal{I}$,* $\int_{t=0}^{\infty} \sum_{i \in B(t)} d_i \int_{\tau=0}^{\frac{p_i^B(t)}{s'}} k\Big(\frac{V_{>d_i}^{B(t)}}{s'} + \tau\Big)^{k-1} \mathrm{d}\tau\mathrm{dt} \leq \frac{1}{s'} B(\mathcal{I})$.

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

Lemma 26. *Consider any given instance $\mathcal{I}$ where all jobs are assigned to a single machine. Suppose B is a valid schedule with s' speed given on the instance*

$\mathcal{I}$. Let v be any constant and $V_{>v}^{B(t)} = \sum_{\substack{i \in B(t) \\ d_i > v}} p_i^{B(t)}$. Then,

$$(V_{>v}^{B(t)})^k \leq \sum_{\substack{i \in B(t) \\ d_i > v}} d_i \int_{\tau=0}^{p_i^B(t)} k(V_{>v}^{B(t)} + \tau)^{k-1} \mathrm{d}\tau.$$

In particular, for any function $g(t) : [0, \infty) \to \mathbb{R}$, it holds that

$$\int_{t=0}^{\infty} g(t) \Big(V_{>g(t)}^{B(t)} / s' \Big)^k \mathrm{dt} \leq \frac{1}{s'} B(\mathcal{I}).$$

The above lemma, with Lemma 25, will enable the analysis without making any assumption on the adversary's scheduling. Especially, Lemma 26 is interesting for the following reason. In the lemma, (LHS) is an expression involving two quantifies which are not related at all. One is a quantity regarding to volume of alive jobs under some schedule B, and the other is any arbitrary function $g(t)$. Due to this lemma, we will be able to bound the changes involving our algorithm's queue status and the adversary's schedule, without explicitly correlating the two. The following corollaries are immediate from the two lemmas.

Corollary 2. $\int_{t=0}^{\infty} d_{a(x,t)} (V_{>d_{a(x,t)}}^{O(t)})^k \mathrm{dt} \leq \mathrm{OPT}_x$.

Corollary 3. $\int_{t=0}^{\infty} \sum_{i \in O_x(t)} d_{ix} \int_{\tau=0}^{p_i^O(t)} k(V_{>d_{ix}}^{O(t)} + \tau)^{k-1} \mathrm{d}\tau \mathrm{dt} \leq \mathrm{OPT}_x$.

The following proposition will be used throughout the analysis.

Proposition 4. *For any constant $0 \leq \epsilon < 1$ and any integer $k \geq 1$, $(1 + \frac{\epsilon}{k})^k \leq 1 + 2\epsilon$.*

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

Lemma 27.

$$\int_{t=0}^{\infty} d_{b(x,t)} \Big(\epsilon(t - r_{b(x,t)}) + p_{b(x,t)}^O(t) \Big)^k \mathrm{dt} \leq 2(1+\epsilon)^k \mathrm{OPT}_x \leq 2^{k+1} \mathrm{OPT}_x$$

Proof. By considering whether $(t - r_i) \geq \tau$ or not, we have

$$(LHS) \quad \leq 2^k \Big[\int_{t=0}^{\infty} d_{b(x,t)} (t - r_{b(x,t)})^k \mathrm{dt} + \int_{t=0}^{\infty} d_{b(x,t)} (p_{b(x,t)}^O(t))^k \mathrm{dt} \Big]$$

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

Lemma 28.

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

Proof. By considering whether $(t - r_i) \geq \tau$ or not, we have

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

By the definition of kth power of fractional flow time and Corollary 3, the lemma easily follows. □

4.5.2 Proof of Lemma 25 and 26

We first explain a "slicing" technique that will be used for the proof of Lemma 25 and 26. For this technique, we will focus on an instance where there is a single machine and since we are focusing on a single machine, we drop the machine x notation. Let s' denote the speed the schedule considered is given. In the slicing technique each job i is replaced with a set of jobs $\mathcal{J}_i$ of uniform processing time $\Delta' = s'\Delta$ and uniform weight $w_i' = \frac{w_i \Delta'}{p_i}$, where Δ is a sufficiently small constant. Note that each new job having size $s'\Delta$ in $\mathcal{J}_i$ requires Δ amount of time to be finished. There are a total of $\frac{p_i}{\Delta'}$ jobs in $\mathcal{J}_i$. Notice that each job's density is the same as that of job i and total volume of jobs in $\mathcal{J}_i$ is the same as the size of job i, i.e. $p_i = V^{\mathcal{J}_i}$. These jobs all arrive at the same time job i arrives. This method was used in [9] to reduce the problem of minimizing ℓ_k-norm of weighted flow time to its unweighted version. To our best knowledge, it has not been formally stated anywhere that the slicing transformation does not affect the weighted *fractional* ℓ_k norm of flow time.

To formally define the transformation, we need more notation. For the (old) given instance of jobs $\mathcal{I}$, let $\mathcal{B}(\mathcal{I})$ denote the schedule under the scheduling policy B. We call jobs in $\mathcal{I}'$ new to distinguish them from jobs in $\mathcal{I}$. The new schedule $\mathcal{B}'(\mathcal{I}')$ for the new instance $\mathcal{I}'$ is naturally defined from the old schedule $\mathcal{B}(\mathcal{I})$. That is, at any time t, job i is processed under $\mathcal{B}(\mathcal{I})$ if and only if a job in $\mathcal{J}_i$ is processed under $\mathcal{B}'(\mathcal{I}')$; this mapping is well-defined since the slicing preserves the volume in replacing each job i with the jobs $\mathcal{J}_i$. For each i, jobs in $\mathcal{J}_i$ are ordered in an arbitrary but fixed way. We let $\mathcal{J}_i(t)$ denote the jobs in $\mathcal{J}_i$ which are alive at time t. Note that $p_i(t)$ in $\mathcal{B}$'s schedule is the same as $V^{\mathcal{J}_i(t)}$ in $\mathcal{B}'$'s schedule. We say that an objective function (or expression)

fFUNC is resilient to slicing if it gives the same value for two instances $\mathcal{I}$ and $\mathcal{I}'$. When $\mathcal{I}$ and $\mathcal{I}'$ are well-understood in the context, they may be dropped. The following lemma easily follows from the definition of kth power of weighted fractional flow time and the slicing transformation.

Lemma 29. *The weighted kth power of fractional flow time is resilient to slicing for any schedule $\mathcal{B}$.*

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

We now discuss the relationship between *integral* kth power flow time of the new instance and the *fractional* kth power flow time of the old instance. We assume that the time is partitioned into unit time slots of size Δ and during each time slot exactly one job is completed. Without loss of generality, we can further assume that jobs arrive only at the beginning of a time slot. These are valid assumptions assuming Δ is sufficiently small. We let $\mathcal{T}$ denote the set of unit times. We will be interested in considering the *integral* kth power flow time of the new instance. Let $N^{(set)}_{(condition)}$ denote the number of alive jobs in the *(set)* that satisfy the *(condition)*. In the slotted time model, we will consider dFUNC, the discrete version of fFUNC for the new instance $\mathcal{I}'$. This will be explicitly defined when it is used. When there is a need to stress that $\mathcal{I}'$ is obtained by slicing jobs into $(s'\Delta)$-sized jobs, we will use $\mathcal{I}'(\Delta)$.

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

We are now ready to prove Lemma 25 and 26.

Proof of [Lemma 25] By Lemma 29, we know that (RHS) is resilient to slicing. We first show that so is (LHS). Recall that we use $\mathcal{I}'$ to denote the new instance obtained by slicing jobs. Let fFUNC denote the function which takes an instance and gives the value of (LHS) on the schedule by B on the instance. To save notation, for job $i \in B(t)$ and for any job $j \in \mathcal{J}_i(t)$, we will abuse the notation $_{>d_j}$ to include not only the jobs of density at least d_j but also the jobs in $\mathcal{J}_i(t)$ of the same density which are finished before job j in the schedule B'. Consider an infinitesimal interval $[t, t+\mathrm{dt})$. Then the change in $\texttt{fFUNC}(B(\mathcal{I}))$ for a job i is $d_i \int_{\tau=0}^{p_i^B(t)/s'} k((V^{B(t)}_{>d_i}/s') + \tau)^{k-1} \mathrm{d}\tau\mathrm{dt}$. It can be easily shown to be equal to $\sum_{j \in \mathcal{J}_i(t)} d_i \int_{\tau=0}^{\Delta} k((V^{B(t)}_{>d_i}/s') + (V^{\mathcal{J}_i(t)}_{>d_j}/s') + \tau)^{k-1} \mathrm{d}\tau\mathrm{dt} = \sum_{j \in \mathcal{J}_i(t)} d_j \int_{\tau=0}^{\Delta} k((V^{B'(t)}_{>d_j}/s') + \tau)^{k-1} \mathrm{d}\tau\mathrm{dt}$, which is exactly the increase in $\texttt{fFUNC}(B(\mathcal{I}'))$ for jobs $\mathcal{J}_i$, thus proving (LHS) being resilient to slicing.

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

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

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

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

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

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

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

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

Proof of [Lemma 26] For this lemma we need the following proposition,

Proposition 5. *For any* $x_1, x_2, ..., x_n \ge 0$,

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

Proof.

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

□

By Proposition 5, we have

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

By multiplying both sides by s'^k, we obtain the first inequality. The above inequality, with Lemma 25, immediately implies the second desired inequality. □

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

$$\begin{aligned}&\frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,1}(t) \\ =& \sum_{i\in A(t)\setminus\{a(t)\}} d_i\int_{\tau=0}^{p_i^A(t)}(\epsilon-s)k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau \\ &+d_{a(t)}(\epsilon-s)\Big(\epsilon(t-r_{a(t)})+p_{a(t)}^A(t)\Big)^k \\ &-\epsilon d_{a(t)}\Big(\epsilon(t-r_{a(t)})\Big)^k \\ =& -(s-\epsilon)\sum_{i\in A(t)} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau \\ &-sd_{a(t)}\Big(\epsilon(t-r_{a(t)})\Big)^k \qquad (4.4)\end{aligned}$$

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

$$
\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,2}(t) \\
= \quad & -\frac{\mathrm{d}}{\mathrm{dt}} \sum_{i \in O(t)} d_i \int_{\tau=0}^{p_i^O(t)} \Big(\epsilon(t-r_i) + V_{>d_i}^{A(t)} + \tau\Big)^k \mathrm{d}\tau = \\
& +(s-\epsilon) \sum_{\substack{i \in O(t) \\ d_i < d_{a(t)}}} d_i \int_{\tau=0}^{p_i^O(t)} k\Big(\epsilon(t-r_i) + V_{>d_i}^{A(t)} + \tau\Big)^{k-1} \mathrm{d}\tau \qquad (4.5) \\
& -\epsilon \sum_{\substack{i \in O(t) \\ d_i \geq d_{a(t)}}} d_i \int_{\tau=0}^{p_i^O(t)} k\Big(\epsilon(t-r_i) + \tau\Big)^{k-1} \mathrm{d}\tau \qquad (4.6) \\
& +d_{b(t)}\Big(\epsilon(t-r_{b(t)}) + V_{>d_{b(t)}}^{A(t)} + p_{b(t)}^O(t)\Big)^k \qquad (4.7)
\end{aligned}
$$

Lines (4.5) and (4.6) are due to the change in time and the algorithm's processing. Line (4.7) is from OPT's processing. We are concerned with upper bounding the change in Φ, so can ignore (4.6). We will first bound $\int_{t=0}^{\infty}$(4.5) and then we will concentrate on bounding $\int_{t=0}^{\infty}$(4.7).

Bounding the total change of (4.5) over time

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

$$
\begin{aligned}
&\int_{t=0}^{\infty} (4.5)\mathrm{dt} \leq \\
&(s-\epsilon)(1+\frac{k}{\epsilon})^{k-1} \int_{t=0}^{\infty} d_i \sum_{\substack{i \in O(t) \\ d_i < d_{a(t)}}} \int_{\tau=0}^{p_i^O(t)} k(\epsilon(t-r_i)+\tau)^{k-1}\mathrm{d}\tau\mathrm{dt} \qquad (4.8) \\
&+(s-\epsilon)(1+\frac{\epsilon}{k})^{k-1} k \int_{t=0}^{\infty} \sum_{\substack{i \in O_x(t) \\ d_i < d_{a(t)}}} d_i p_i^O(t) (V_{>d_i}^{A(t)})^{k-1}\mathrm{dt} \qquad (4.9)
\end{aligned}
$$

Via simple algebra and Lemma 28, we have (4.8) $\leq (\frac{4k}{\epsilon})^k \mathrm{OPT}_x$. The second term (4.9) can be bounded by the following lemma.

Lemma 30.

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

Proof. Consider any fixed time t. We can assume that all jobs in $A(t)$ and $O(t)$ have infinitesimal size, since both sides are resilient to slicing. We partition $O_x(t)$ into $I(t)$ and $I'(t)$ depending on whether a job i satisfies $V_{>d_i}^{A(t)} \leq \frac{k}{\epsilon} V_{>d_i}^{O(t)}$ or not. For the set $I(t)$, by Corollary 3, we have

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

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

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

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

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

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

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

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

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

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

This completes the proof. □

By summing each of these expressions we have,

$$\begin{aligned} & \int_{t=0}^{\infty} (4.5) \mathrm{dt} \\ \leq \; & 3(\frac{4k}{\epsilon})^k \mathrm{OPT}_x \\ & + 9\epsilon(s-\epsilon) \int_{t=0}^{\infty} \int_{\tau=0}^{p_i^A(t)} k(V_{>d_i}^{A(t)} + \tau)^{k-1} \mathrm{d}\tau \mathrm{dt} \end{aligned} \tag{4.10}$$

Bounding the total change of (4.7) over time

By considering whether $V_{>d_{b(t)}}^{A(t)} \leq \frac{k}{\epsilon}\Big(\epsilon(t - r_{b(t)}) + p_{b(t)}^O(t)\Big)$ or not, we have

$$\begin{aligned} & \int_{t=0}^{\infty} (4.7) \mathrm{dt} \\ \leq \; & (1+\frac{\epsilon}{k})^k \int_{t=0}^{\infty} d_b(t)(V_{>d_{b(t)}}^{A(t)})^k \\ & + (1+\frac{k}{\epsilon})^k d_b(t)\Big(\epsilon(t - r_{b(t)}) + p_{b(t)}^O(t)\Big)^k \mathrm{dt} \\ \leq \; & (1+2\epsilon) \int_{t=0}^{\infty} \sum_{\substack{i \in A(t) \\ d_i > d_{b(t)}}} d_i \int_{\tau=0}^{p_i^A(t)} k\Big(V_{>d_i}^{A(t)} + \tau\Big)^{k-1} d\tau \mathrm{dt} \\ & + 2(\frac{4k}{\epsilon})^k \mathrm{OPT}_x \end{aligned} \tag{4.11}$$

The last inequality follows by applying Lemma 26 and 27.

4.5.5 Analyzing $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,3}(t)$

We first study the change coming from our algorithm's processing and time elapse, which is

$$(s-\epsilon)\sum_{i\in A_x(t)} d_i \int_{\tau=0}^{p_i^A(t)} k\Big[\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+V_{>d_i}^{O(t)}+\tau\Big)^{k-1} - \Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\Big]\mathrm{d}\tau \tag{4.12}$$

$$+sd_{a(t)}\Big[\Big(\epsilon(t-r_{a(t)})+V_{>d_{a(t)}}^{O(t)}\Big)^k - \Big(\epsilon(t-r_{a(t)})\Big)^k\Big] \tag{4.13}$$

We need the following lemma whose proof is very similar to that of Lemma 30. The lemma is slightly different from Lemma 30; roughly speaking the (LHS) is from the algorithm's perspective rather than from the optimal solution's.

Lemma 31.

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

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

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

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

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

The remaining proof is very similar to that of Lemma 30. As in the proof of Lemma 30, we can assume that jobs in $A(t)$ and $O(t)$ have infinitesimal size. Also similarly, we can find a family of disjoint sets $G_i(t)\subseteq O(t), i\in I'(t)$ such that

1. $\forall i\in I'(t)$, $V^{G_i(t)} = \frac{\epsilon^2}{k^2}p_i^A(t)$.

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

3. $\forall i \in I'(t)$, $\forall j \in G_i(t)$, $d_j \geq d_i$.

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

$$\begin{aligned} & V^{(\bigcup_{i' \in [i-1]} G_i(t))} + \frac{\epsilon^2}{k^2}p^A_i(t) \\ = \quad & \frac{\epsilon^2}{k^2}\sum_{i' \in [i]} p^A_i(t) \leq \frac{\epsilon^2}{k^2}V^{A(t)}_{>d_i} \leq \frac{1}{k}V^{O(t)}_{>d_i}. \end{aligned}$$

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

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

$$\begin{aligned} & \sum_{j \in G_i(t)} d_j \int_{\tau=0}^{p^O_j(t)} (V^{O(t)}_{>d_j} + \tau)^{k-1} \mathrm{d}\tau \\ \geq \quad & d_i V^{G_i(t)}((1-\frac{1}{k})V^{O(t)}_{>d_i})^{k-1} \geq \frac{\epsilon^2}{3k^2} d_i p^A_i(t)(V^{O(t)}_{>d_i})^{k-1} \end{aligned}$$

Hence we have

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

The last inequality is due to Corollary 3. This completes the proof. □

We first bound $\int_{t=0}^{\infty}$(4.12)dt. We can assume that $k \geq 2$ since (4.12) $= 0$ when $k = 1$. By considering whether or not $V^{O(t)}_{>d_i} \leq \frac{\epsilon}{k}\Big(\epsilon(t-r_i) + V^{A(t)}_{>d_i} + \tau\Big)$,

and via simple algebra, we have

$$\begin{aligned}
&(4.12)\\
&\leq 2\epsilon(s-\epsilon)\sum_{i\in A_x(t)} d_i \int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau\\
&\quad +(1+\frac{k}{\epsilon})^{k-1}(s-\epsilon)k\sum_{i\in A_x(t)} d_i p_i^A(t)(V_{>d_i}^{O(t)})^{k-1}
\end{aligned}$$

By Lemma 31, the fact $s\leq 2$, and a simple algebra it follows that

$$\begin{aligned}
&\int_{t=0}^{\infty}(4.12)\mathrm{dt}\\
&\leq 5\epsilon(s-\epsilon)\int_{t=0}^{\infty}\sum_{i\in A_x(t)} d_i \int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau\mathrm{dt}\\
&\quad +(\frac{2k}{\epsilon})^{k+1}\mathrm{OPT}_x
\end{aligned}\tag{4.14}$$

We now bound (4.13). By considering whether $V_{>d_{a(t)}}^{O(t)}\leq\frac{\epsilon}{k}(\epsilon(t-r_a))$ or not, we have

$$(4.13)\leq 2\epsilon s d_{a(t)}\Big(\epsilon(t-r_{a(t)})\Big)^k+s(1+\frac{k}{\epsilon})^k d_{a(t)}(V_{>d_{a(t)}}^{O(t)})^k.$$

Thus, by Corollary 2,

$$\int_{t=0}^{\infty}(4.13)\mathrm{dt}\leq 2\epsilon s d_{a(t)}\int_{t=0}^{\infty}\Big(\epsilon(t-r_{a(t)})\Big)^k\mathrm{dt}+2(\frac{2k}{\epsilon})^k\mathrm{OPT}_x\tag{4.15}$$

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

$$\begin{aligned}
&\sum_{\substack{i\in A(t)\\ d_i<d_{b(t)}}} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+V_{>d_i}^{O(t)}+\tau\Big)^{k-1}\mathrm{d}\tau \qquad (4.16)\\
\leq\ & (1+\frac{\epsilon}{k})^{k-1}\sum_{\substack{i\in A(t)\\ d_i<d_{b(t)}}} d_i\int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i)+V_{>d_i}^{A(t)}+\tau\Big)^{k-1}\mathrm{d}\tau\\
&+(1+\frac{k}{\epsilon})^{k-1}\sum_{\substack{i\in A(t)\\ d_i<d_{b(t)}}} k d_i p_i^A(t)(V_{>d_i}^{O(t)})^{k-1}
\end{aligned}$$

The inequality easily follows by considering whether $V^{O(t)}_{>d_i} \leq \frac{\epsilon}{k}\Big(\epsilon(t-r_i) + V^{A(t)}_{>d_i} + \tau\Big)$ or not. And by Lemma 31 and simple algebra, we obtain

$$\begin{aligned}
&\int_{t=0}^{\infty} (4.16)\mathrm{dt} \\
&\leq (1+2\epsilon)\int_{t=0}^{\infty} \sum_{\substack{i \in A(t) \\ d_i < d_{b(t)}}} d_i \int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i) + V^{A(t)}_{>d_i} + \tau\Big)^{k-1} \mathrm{d}\tau\mathrm{dt} \qquad (4.17)\\
&+3\epsilon\int_{t=0}^{\infty} \sum_{i\in A(t)} d_i \int_{\tau=0}^{p_i^A(t)} k\Big(V^{A(t)}_{>d_i} + \tau\Big)^{k-1} \mathrm{d}\tau\mathrm{dt} + (\frac{2k}{\epsilon})^{k+1}\mathrm{OPT}_x \qquad (4.18)
\end{aligned}$$

Here (4.18) was obtained assuming $k \geq 2$. When $k = 1$, $\int_{t=0}^{\infty}(4.16)\mathrm{dt} \leq (4.17)$. Thus the above upper bound holds for all $k \geq 1$.

4.6 Upperbound: Final Analysis

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

$$\begin{aligned}
&\int_{t=0}^{\infty} \frac{\mathrm{d}}{\mathrm{dt}}\Phi_x(t)\mathrm{dt} \\
= &\int_{t=0}^{\infty} [\frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,1}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,2}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi_{x,3}(t)]\mathrm{dt} \\
\leq &\int_{t=0}^{\infty} (4.4)\mathrm{dt} + (4.10) + (4.11) + (4.14) + (4.15) + (4.17) + (4.18) \\
\leq &((1+2\epsilon) - (1-17\epsilon)(s-\epsilon)) \sum_{i\in A_x(t)} d_{ix} \int_{\tau=0}^{p_i^A(t)} k\Big(\epsilon(t-r_i) + V^{A(t)}_{>d_{ix}} + \tau\Big)^{k-1} \mathrm{d}\tau \\
&-s(1-2\epsilon)\epsilon^k \int_{t=0}^{\infty} d_{a(x,t),x}(t - r_{a(x,t)})^k \mathrm{dt} \\
&+8(\frac{4k}{\epsilon})^{K+1}\mathrm{OPT}_x \\
\leq &-s(1-2\epsilon)\epsilon^k A_x + 8(\frac{4k}{\epsilon})^{k+1}\mathrm{OPT}_x
\end{aligned}$$

The second inequality can be obtained by combining (4.11) with (4.17). The last inequality holds when $1 + 30\epsilon \leq s \leq 2$, and $0 < \epsilon \leq \frac{1}{50}$. Since Φ is 0 before no jobs are released and also after all jobs are completed by A and OPT, the total change of Φ is 0. Recall that the sum of non-continuous changes is at most 0. Thus the total continuous change of Φ over time is non-negative. Hence from the above inequality for each machine x, we have $0 \leq \sum_{x\in[m]} \int_{t=0}^{\infty} \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t)_x\mathrm{dt} \leq -s(1-2\epsilon)\epsilon^k A + 8(\frac{4k}{\epsilon})^{k+1}\mathrm{OPT}$.

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

Theorem 7. *For any integer $k \geq 1$ and for any $0 < \epsilon \leq 1$, there exists a $(1+\epsilon)$-speed $O(\frac{k}{\epsilon^{2+1/k}})$-competitive algorithm for minimizing weighted ℓ_k-norms of fractional flowtime on unrelated machines. In particular, the algorithm is immediate dispatch and non-migratory.*

Using the relation between integral kth power flow time and fractional kth power flow time discussed in Section 1.5.3, we can show Theorem 5.

4.7 Lowerbound

In this section we prove Theorem 6.

Proof of [Theorem 6] Suppose that we have $m = 2^k$ machines. We create the following adversarial instance $\mathcal{I}$. It has k groups of jobs: G_α, $\alpha \in [k]$. Jobs in each group (set) G_α can be assigned to only a subset of machines $\mathcal{M}_\alpha$ where $|G_\alpha| = |\mathcal{M}_\alpha| = 2^{k+1-\alpha}$. All jobs have uniform size. For simplicity, we assume that all jobs are released at time 0, but the algorithm is given jobs to schedule one by one; this can be simulated by letting jobs have sufficiently large size and arrive at distinct integer times during $[0, 2m]$. We will assume that the jobs in G_i arrives before G_j if $i < j$.

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

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

4.8 Concluding Remarks

In this chapter we introduced a scalable algorithm for the weighted ℓ_k norm of flow time in the unrelated machines model for any fixed $k > 0$. It is important

to note that our algorithm must know the speed (ϵ) to be scalable. That is the algorithm uses ϵ to determine the machine assignment of jobs. Knowing the speed the algorithm is given has recently been shown to be useful in scheduling analysis [65, 42, 33]. One possible candidate algorithm that does not depend on ϵ is the algorithm that assigns a job to the machines which increases the (fractional) kth norm of flow time the least. We currently do not know if this algorithm is scalable or not. We were able only to show that it is $O(1)$-competitive with speed $k + \epsilon$. Recently, Anand et al. gave another scalable algorithm based on a dual fitting [3]. Their algorithm is slightly simpler than ours, but still requires the knowledge of the speed it is given.

Open Problem 3. *Consider any integer $k \geq 2$. For the problem of minimizing the weighted ℓ_k norm of flow time in the unrelated machines model, give a scalable algorithm that does not depend on ϵ, or show that no such algorithm exists.*

Our lower bound is restricted to immediate-dispatch and non-migratory algorithms. Without these restrictions, there may exist a scalable algorithm whose competitive ratio does not depend on k.

Open Problem 4. *Consider any integer $k \geq 2$. For the problem of minimizing the weighted ℓ_k norm of flow time in the unrelated machines model, give a scalable algorithm whose competitive ratio does not depend on k, or show that no such algorithm exists.*

In the offline setting, there exist only a few positive results without speed augmentation [52, 53, 93]. All these results can handle only special cases. For example, [52] shows an $O(\log P)$-approximation and a nearly matching lowerbound when each job can go to a specific subset of machines, and has the same size on all machines in the subset; here P is the ratio of the maximum size of a job to the minimum size of a job.

Open Problem 5. *Consider the problem of minimizing average flow time offline in the unrelated machines model. Let n be the number of jobs and P be the ratio of the maximum size of a job on any machine to the minimum size of a job on any machine. For the problem, give an approximation algorithm whose approximation factor is poly-logarithmic in n and P, or show that no such algorithm exists.*

Chapter 5

Non-clairvoyant Scheduling on Related Machines

5.1 Introduction

Around 2002, the consequences of Moore's law finally impacted computer processor designers as they hit a "thermal wall", where it was no longer economically viable to cool the ever-hotter traditional uniprocessor architectures. One technology adopted to surmount this thermal wall is multiprocessor chips. The common rule of thumb is that the power used by a processor is roughly cubic in the speed of the processor. In theory m processors with speed s/m could potentially handle the same load as one speed-s processor, but with a factor of $1/m^2$ less power (assuming that the power consumption of each processor grows in proportion to s^3). Moore's gap, which is the difference in the achievable performance predicted by Moore's law and the actual performance of commercial processors, is largely explained by difficulty of getting many slower processors to approximate the performance of one fast processor in practice.

Current commercial chip architectures mostly commonly consist of a homogeneous collection of identical processors. However, many computer architects believe that architectures consisting of *heterogeneous* processors/cores will be the dominant architectural design in the future [23, 74, 75, 80, 81]. The main advantage of a heterogeneous architecture over a homogeneous architecture is that it allows for the inclusion of processors whose design is specialized for particular types of jobs, with the intent that jobs be assigned to a processor best suited for that job. Most notably, it is envisioned that these heterogeneous architectures will consist of a small number of high-power high-performance processors for critical jobs, and a larger number of lower-power lower-performance processors for less critical jobs. Naturally, the lower-power processors would be more energy efficient in terms of the computation performed per unit of energy expended, and would generate less heat per unit of computation. An example of a such a heterogeneous multiprocessor is the STI Cell chip. For a given area and power budget, heterogeneous multiprocessor architectures can give a order of magnitude better performance than homogeneous multiprocessor architectures [63]. This makes research into scheduling policies for heterogeneous processors of fundamental importance (see the position paper [23] for further arguments about the importance of this research direction).

Currently the most pervasive technology for achieving power heterogeneity is that of *speed-scalable* processors. Speed-scalable processors have a collection of available speeds, and the power consumed at the various speeds is a convex function of the speed. The speed of the processors can be dynamically scaled over time. A system that sits atop a speed-scalable processor needs not only an online scheduling policy to determine which job to run on which processor, but also a speed scaling policy for setting the speed of these processors. In the homogeneous setting, each processor runs at the same speed when using a particular power setting while in the heterogeneous setting the speed for a given power depends on the specific processor being considered.

Following the line of research in [56, 57], we investigate worst-case performance guarantees (or competitive ratios) achievable by algorithms for scheduling heterogeneous multiprocessor architectures. Throughout this chapter we focus on a type of heterogeneous multiprocessor scheduling which is best described as *related heterogeneous* multiprocessors and is defined as follows. We adopt the following formal model as in [56]. We are given a collection of m processors/machines, with processor i having a *speed function* Q_i: for every value P, $Q_i(P)$ is the speed obtained when the processor is run at a power of P. Notice that the speed depends on the processor being considered. One can assume without loss of generality that Q_i is concave, continuous, and $Q_i(0) = 0$ [13]. If processors are not running a job then they can be shut down, and consume no power. We note that an important special case of this model is the *related machines* model, where each processor $i$can only run at a single fixed speed s_i and each processor consumes no power (We say that a processor runs at a *fixed speed* if there is only one possible speed for the processor and it consumes no energy).

Jobs arrive in an online fashion over time, with job J_j arriving in the system at its release/arrival time r_j. The job has a positive *size* p_j, and a positive *importance/weight* w_j. Each job can be scheduled on only one processor at each time and can be preempted. The goal is to devise a scheduling policy and an associated speed scaling policy to minimize some weighted combination of the average (weighted) flow time $\sum_j w_j F_j$ of the jobs and the total energy consumed. Here, the flow time F_j of a job J_j is the difference between its completion time and its release time. We can assume without loss of generality that the objective is total (or equivalently average) weighted flow time plus total energy consumption by scaling the speed functions or job weights. This objective makes a balance between the system performance and power consumption; we defer more discussion about this objective to Section 5.2.

In particular, we will be interested in obtaining *non-clairvoyant* algorithms. A non-clairvoyant algorithm has to make scheduling decisions without knowing the actual job size until completing each job. Studying the performance of non-clairvoyant algorithms is of particular importance because schedulers in general purpose systems generally do not know the size of the job upon its arrival. To recap, we study non-clairvoyant scheduling algorithms for mini-

mizing total (weighted) flow time plus total energy consumption on (related) heterogeneous machines.

When there is only a single processor, i.e. $m = 1$, the problem is fairly well understood. Even for a single processor of fixed speed, it is known that any randomized algorithm has a competitive ratio of $\Omega(\log n)$ without speed augmentation [83]. Also the non-clairvoyant algorithms SRPT (Shortest Elapsed Time First) and LAPS are known to be scalable [1] . For the definition of these algorithms, we refer the reader to Section 1.4. Also the weighted version of these algorithms (WSETF and WLAPS, reps.) are also scalable for weighted average flow time [9, 15]. For a single processor that is speed-scalable, LAPS (WLAPS) is known to be scalable for total (weighted) flow time plus energy [27].

When all m processors are homogeneous, [76] considered the special case where the allowable speeds are bounded and the power that each processor consumes is polynomial in the speed it is run. They considered a non-clairvoyant algorithm that combines a varian of RR (Round Robin) and the speed scaling policy from [14], showed it is scalable.

The heterogeneous machines setting seems much more challenging. Most of the analysis techniques for scheduling algorithms in the homogeneous multiprocessor fixed speed setting do not extend to the heterogeneous fixed speed multiprocessor setting for one or both of the following reasons. Firstly, contrary to conventional intuition, priority algorithms such as SRPT and SJF are not locally competitive[2] (even with any constant factor speed augmentation) as they are on a homogeneous fixed speed multiprocessor [84]. (See Section 5.4.4.) Secondly, unlike the homogeneous case, it is difficult to establish lower bounds on when the optimal solution completes these jobs. E.g., the total work of a set of jobs divided by the total speed of the processors is not useful: even though a set of processors may have large aggregate speed, each individual processor may be very slow. For the same reason, the number of processors is not a useful quantity.

In fact, even when in the heterogeneous machines setting where each machine has a fixed speed, the only algorithm, either clairvoyant or non-clairvoyant, that is known to be scalable is the one given by Chadha et al. [24]. Unfortunately, this algorithm is far from being non-clairvoyant. When the online scheduler is non-clairvoyant, the only non-trivial result known in the heterogeneous machines setting was given by Gupta et al. [56]. They showed RR (Round Robin) is $(2+\epsilon)$-speed $O(1)$-competitive with an appropriate speed scaling policy.

[1] A scheduling algorithm is scalable if it possess a constant competitive ratio when provided a processor that is a factor $(1+\epsilon)$ faster than optimal solution [69, 42]. See Section 1.3.3 for more details of resource augmentation.

[2] An algorithm is *locally competitive* if at all times the increase in the algorithm's objective is within a constant factor of the increase in the optimal solution's objective. For weighted flow time this means that the total weight of unsatisfied jobs in the algorithm's schedule is within a constant factor of the total weight of unsatisfied jobs in the optimal solution's schedule at all times.

5.1.1 Our Results

We give the first *non-clairvoyant* scalable algorithm for the objective of minimizing total (*unweighted*) flow time plus energy when machines are related. That is, we show that the following algorithm, which combines the LAPS [42] (Latest Arrival Processor Sharing) policy with a non-trivial speed-scaling policy is scalable for the objective of total flow plus energy on a heterogeneous multiprocessor. The speed-scaling policy uses the so-called Speed Abstraction Problem that determines the maximum aggregate speed that is achievable subject to (i) the number of machines, and (ii) and total power used, both being at most the number of unfinished jobs given in [56]. This improves upon the result of [56] which shows that the scheduling algorithm Equipartition is $(2+\epsilon)$-speed $O(1)$-competitive for the same objective. We note that this is the *first* example of a scalable non-clairvoyant algorithm for speed-scalable heterogeneous processors, or even fixed-speed related machines. We note that there is a strong lower bound on the competitive ratio without resource augmentation [83].

At a high level, the main technical difficulty in showing LAPS is scalable is the following. Consider the related machines model where machines have a fixed speed and do not consume any energy. In this case, typical arguments for LAPS on homogeneous multiprocessors proceed by (i) showing that we can treat m identical machines as just a single processor of speed m as long as we restrict each job to not run at more than unit speed at any time, and (ii) on this one machine instance, showing that we can just distribute the speed of the system among the ϵn most recently arriving jobs, and still make enough progress on the overall objective. However, we run into trouble in both steps for heterogeneous systems. For (i) it is not clear what the single machine instance should be, because the machines could have vastly different speed profiles, and we can't therefore place such natural restrictions on jobs to capture the fact that a job can run only on one machine. So sticking with multiprocessors, the problem then with (ii) is that ϵn could always be smaller than m, the number of machines. In this case it is not possible to fully utilize the resources of m machines without running a job simultaneously on two machines, which is an infeasible schedule. However, we show that the algorithm which shares the ϵn fastest machines between the ϵn latest arriving jobs is scalable. We use this as a starting point for our general algorithm in the speed-scaling case. Because of the issues discussed above, our analysis is also forced to reason directly about a heterogeneous multiprocessor system.

If we were to consider weights, the situation however becomes more challenging. We show in Section 5.4 that the standard extension of priority algorithms [3] for the *weighted* flow objective, namely HDF, WSETF, and WLAPS (Weighted

[3] We say a scheduling algorithm is a priority algorithm if the jobs are assigned a single parameter (which can possibly change with time) called its priority, and the scheduling decision is based solely on each job's priority. For example, in SRPT, the priority of each job is simply the remaining processing time of the job.

Latest Arrival Processor Sharing), are all not $O(1)$-speed $O(1)$-competitive, even for the related machines setting when machines have different but fixed speeds and consume no energy. Note that as mentioned above, these algorithms *are scalable* for the homogeneous case when all processors have the same speed [42, 47]. Intuitively, perhaps the underlying reason is that when we have both related machines and weighted jobs there is an extra dimension to the problem over both the cases of weighted jobs on homogeneous machines and unweighted jobs on heterogeneous machines. The natural extensions of priority algorithms fail to capture the interplay between these dimensions. We believe that this justifies the analysis of non-standard algorithms in [24, 57].

On the whole, our results suggest that scheduling heterogeneous multiprocessors may be inherently more difficult than scheduling homogeneous multiprocessors, or at the very least, require substantially different algorithms.

5.1.2 Related Work

In this section, we summarize both clairvoyant and non-clairvoyant algorithms that are rested to our problems. For a single processor of fixed speed, the well-known priority algorithms [4] covered in standard introductory operating systems texts are known to be scalable (i.e. possess a constant competitive ratio when provided a processor that is a factor $(1+\epsilon)$ faster than the optimal solution) for the unweighted case—these include SRPT (Shortest Remaining Processing Time), SJF (Shortest Job First), SETF (Shortest Elapsed Time First), and their weighted versions are known to be scalable for the weighted case [69, 19, 42, 8, 15]. (See Section 1.4 for definitions of these algorithms.) Likewise, for a single processor that is speed-scalable, we can obtain near-optimal algorithms in the weighted or unweighted settings by combining the standard priority scheduling algorithms with a natural speed-scaling policy where the power is set to be a small multiple of the total weight of the unsatisfied jobs [14, 13, 4, 25, 27]. It is easy to see that such a speed-scaling policy is natural (for the objective of weighted flow times plus energy) because it balances the increase in the weighted flow time objective with the increase in the energy objective.

Many of these standard priority scheduling algorithms are known to be scalable for the problem of homogeneous fixed-speed *multi*processors [97, 47, 42]. As mentioned previously, [76] gave a scalable algorithm in the homogenous processors setting for some special cases of speed functions. [26] gave a logarithmic competitive algorithm when jobs can be processed by multiple processors.

For heterogeneous multiprocessors however, the landscape is not so well-charted. Scalable clairvoyant algorithms are known for weighted flow on fixed speed processors [24], and for weighted flow plus energy on speed-scalable

[4] We say a scheduling algorithm is a priority algorithm if the jobs are assigned a single parameter (which can possibly change with time) called its priority, and the scheduling decision is based solely on each job's priority. For example, in SRPT, the priority of each job is simply the remaining processing time of the job.

	Fixed Speed Processor	
	Clairvoyant	Non-Clairvoyant
Unweighted	SRPT optimal SJF scalable [19]	SETF scalable [69] LAPS scalable [42]
Weighted	HDF scalable [19]	WSETF scalable [8]
	Speed-Scalable Processor	
	Clairvoyant	Non-Clairvoyant
Unweighted	SRPT competitive [14, 13, 4] SJF scalable [13]	LAPS scalable [25]
Weighted		WLAPS scalable [15]

Table 5.1: Guarantees for the standard scheduling algorithms on a single processor

	Fixed Speed Processors	
	Clairvoyant	Non-Clairvoyant
Unweighted	SRPT scalable [97, 47] SJF scalable [97]	LAPS scalable [42]
Weighted	HDF scalable [97]	WLAPS scalable [15]

Table 5.2: Guarantees for the standard scheduling algorithms on a homogeneous multiprocessor

	Fixed Speed Processors	
	Clairvoyant	Non-Clairvoyant
Weighted	HDF not scalable [*] Scalable Algorithm [24]	WSETF not scalable [*] WLAPS not scalable [*]
	Speed-Scalable Processors	
	Clairvoyant	Non-Clairvoyant
Unweighted		PS $(2+\epsilon)$-speed $O(1)$-competitive [56] LAPS Variant scalable [*]
Weighted	Scalable Algorithm [57]	

Table 5.3: Guarantees for the standard scheduling algorithms on a heterogeneous multiprocessor

processors [56]. These algorithms are quite different, and more complicated than the standard priority algorithms. It is also known that the non-clairvoyant scheduling algorithm RR (Round Robin) is $(2+\epsilon)$-speed $O(1)$-competitive for the objective of unweighted flow plus energy on speed-scalable processors [56].

We summarize previously known results together with our results in Table 5.1, 5.2 and 5.3. Our results are marked by [*]. See the following summary to see what scheduling algorithm each short name refers to. Any algorithm that starts with "W" implies the weighted version of its unweighted counterpart. Note that showing an upper bound for the non-clairvoyant setting subsumes the clairvoyant setting. Likewise, an upper bound for the speed scaling setting subsumes the fixed processor setting. A lower bound for an algorithm in the fixed processor setting then implies a lower bound for the algorithm in the speed scaling setting.

5.2 Formal Problem Statement and Notation

In the (related) heterogeneous processors setting, we are given m processors/machines, with processor i having a *speed function* Q_i. For every value $P \geq 0$, $Q_i(P)$ is the speed obtained when the processor is run at a power of P. The achieved speed depends on the processor being considered. One can assume without loss of generality that for all $i \in [m]$, tQ_i is concave, continuous, and $Q_i(0) = 0$ [13]. If a processor has no job to run, it can be shut down, and consume no power. We note that an important special case of this model is the *related machines* model, where each processor $i$can only run at a single fixed speed s_i and each processor consumes no power (We say that a processor runs at a *fixed speed* if there is only one possible speed for the processor and it consumes no energy).

Each job J_j arrives online in the system at its release/arrival time r_j with a size $p_j > 0$ and possibly with weight $w_i > 0$. Each job can be scheduled on only one processor at each time and can be preempted. The flow F_j of a job J_j is its completion time C_j minus its release time r_j. This is the amount of time that the job waits to be satisfied. The weighted flow for a job J_j is $w_j F_j$, and the weighted flow for a schedule is $\sum_j w_j F_j$.

Our goal is to minimize the total weighted flow time plus the total energy used. The intuitive rationale for the objective of weighted flow plus energy can be understood as follows: assume that the possibility exists to invest E units of energy to decrease the flow of jobs $J_1, \ldots, J_k$ by $x_1, \ldots, x_k$ respectively for some $k > 0$. An optimal scheduler for this objective would make such an investment if and only if $\sum_{i=1}^{k} w_i x_i \geq E$. So the importance w_j of job J_j can be viewed as specifying an upper bound on the amount of energy that the system is allowed to invest to reduce J_j's flow time by one unit of time. Hence jobs with higher weight are more important, since higher investments of energy are permissible to justify a fixed reduction in the job's flow time.

Particularly, we require our algorithm to be non-clairvoyant. That is, our algorithm must make scheduling decisions without knowing the actual jobs sizes until completing them respectively.

We give the following summary of terminology to clarify the scheduling problems that are mentioned in this chapter.

- **Speed-Scalable Processors vs. Fixed-Speed Processors:** A fixed speed processor has only one allowable speed, the power used can be assumed to be zero without loss of generality. A speed-scalable processor can change its speed over time and the energy consumed depends on the speed used.
- **Homogeneous Multiprocessor vs. Heterogeneous Multiprocessor:** In the homogeneous setting, the speed function of every processor is the same. That is, each processor runs at the same speed for a given amount of power. However, at any given time, the processors can run at different speeds by using different powers. In the heterogeneous setting, each processor has its own specified speed function.

- **Unweighted vs. Weighted Jobs:** In the unweighted setting each job is of equal importance, i.e., all weights are assumed to be one. In the weighted case, jobs have varying importance/weights associated.
- **Clairvoyant vs. Non-Clairvoyant Scheduler:** A clairvoyant scheduler learns the job size when it is released. A non-clairvoyant scheduler does not learn a job's size and must make scheduling decisions without this information.

5.3 Latest Arrival Processor Sharing (LAPS) on a Heterogeneous Multiprocessor for Flow Plus Energy

In this section we show that a natural extension of the LAPS algorithm is scalable for the objective of minimizing the total flow time plus energy on a heterogeneous multiprocessor. Recall that in this model, each processor i is speed-scalable with an independent speed function Q_i, and the scheduler at each time must decide on the speed-scaling policy and the job assignment policy. We begin by describing these policies for our algorithm LAPS.

LAPS Speed Scaling Policy. At each time t, a collection of processors and associated speed settings are selected to maximize the aggregate speed extracted, subject to the constraints that **(i)** the number of processors selected is at most $\lceil \epsilon |A(t)| \rceil$, and **(ii)** the aggregate power used is at most $\lceil \epsilon |A(t)| \rceil$ where $A(t)$ is the set of unfinished jobs for the online algorithm. More formally, the total speed extracted is given by the algorithm $\mathsf{GreedySS}(\epsilon |A(t)|)$ defined below. Note that if there are more machines than $\lceil \epsilon |A(t)| \rceil$ being used then our algorithm idles some of the processors even though there are jobs that could be scheduled.

LAPS Job Selection Policy. The extracted speed is evenly shared among the $\lceil \epsilon |A(t)| \rceil$ jobs that arrived the most recently. Such a distribution is possible because the number of machines running at non-zero speed in the speed scaling policy defined by GreedySS is at most $\lceil \epsilon |A(t)| \rceil$, and in this case, it is easy to have the algorithm cycle through different permutations to share the $\lceil \epsilon |A(t)| \rceil$ jobs on the chosen machines.

The Speed Abstraction Problem and the GreedySS Algorithm. We now define the speed extraction problem and define an optimal greedy algorithm GreedySS for this problem. The definition of the algorithm and proof of Lemma 32 appears in [56]. We re-state it for completeness.

Speed Extraction Problem. Given an integer power budget W, assign an integer power budget of E_i to each processor i so as to maximize the total extracted speed $\sum_i Q_i(E_i)$ subject to the constraints that E_i is a non-negative integer, and $\sum_i E_i \leq W$.

Algorithm GreedySS**.** Intuitively the algorithm partitions the power budget into units, and assigns each unit to the machine which offers the best increase to the total speed that can be extracted. Note that we only constrain all feasible solutions for the above speed extraction problem to set integral values for the E_i's, and make no such assumption about the different power settings of the machines in general. We now give the pseudo-code of GreedySS for completeness:

- Initially set $E_i := 0$ for all processors i. E_i will eventually be the power used by processor i.
- For $j = 1$ to W do
  - Let $k = \arg\max_i Q_i(E_i + 1) - Q_i(E_i)$
  - Increment E_k to $E_k + 1$
- Set the speed s_i of each processor i to be $Q_i(E_i)$

Lemma 32. *[56] The greedy algorithm* GreedySS *optimally solves the speed extraction problem.*

5.3.1 Simplifying Assumptions

In order to convey the main idea of our analysis more clearly, we make the following simplifying assumptions. These assumptions will affect the resulting competitive ratio by a factor of at most $O_\epsilon(1)$.

(A): We assume that OPT is the GKP algorithm [56] which is a clairvoyant online algorithm that is $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive (by doing this, we only lose an additional factor of $O(1/\epsilon)$ in the competitive ratio). In particular, we crucially use the following property of the GKP algorithm: if GKP has $|O(t)|$ jobs unsatisfied at any time t, then the most speed it can use (in total over all machines) at this time is GreedySS$(|O(t)|)$. This follows from the fact that the GKP algorithm always runs any machine at a power that is at most the number of unfinished jobs assigned to the machine; this gives a valid solution for the Speed Extraction problem, and the quantity GreedySS$(|O(t)|)$ can only be larger by its optimality.

(B): We assume that the arrival times of jobs are distinct to simplify the analysis—we can handle identical arrivals by making infinitesimally small perturbations in the arrival times.

(C): We assume that LAPS is given $(1 + 10\epsilon)$ speed-up for some given parameter $0 < \epsilon < 1/10$.

5.3.2 Potential Function Analysis

In this section we define and analyze a potential function to bound the competitiveness of LAPS. A tutorial on the use of potential functions to analyze

scheduling problems can be found in [67]. Before we define the potential function, we introduce some notation. Denote the completion time of job J_i as C_i^A (and C_i^O) for the online algorithm (and optimal schedule respectively). At any time t, let $A(t)$ denote the set of unsatisfied jobs in the algorithm's schedule, and likewise $O(t)$ is the set of unsatisfied jobs in OPT's schedule. We also let $p_i^A(t)$ denote the remaining work at time t for job J_i in the algorithm's schedule, and $p_i^O(t)$ is the remaining work at time t for job J_i in OPT's schedule. Also define $z_i(t) = \max\{p_i^A(t) - p_i^O(t), 0\}$. For a job J_i, let $\text{rank}(i,t) := \sum_{j_{i'} \in A(t), r_{i'} \leq r_i} 1$ denote the number of unfinished jobs that arrived earlier. For any integer value W, let $Q(W) := \mathsf{GreedySS}(W)$ denote the value of the optimal solution to the Speed Extraction problem with budget W. Our potential function is defined as follows.

$$\Phi(t) = \frac{2}{\epsilon^2} \sum_{J_i \in A(t)} \frac{\text{rank}(i,t) z_i(t)}{Q(\text{rank}(i,t))}$$

Now we bound the changes in the potential function. When bounding the changes, the following lemma will be useful. The proof of the following lemma is straightforward given the definition of Q.

Lemma 33. *For any integers A and B such that $B \geq A$, we have that $Q(A) \geq \frac{A}{B} Q(B)$.*

Proof. Consider the run of the algorithm $\mathsf{GreedySS}(B)$, and consider the B increments that it made. By definition of $\mathsf{GreedySS}$, $Q(B)$ is the sum of the incremental speeds we obtained at each step, and these values are monotonically non-increasing. As a result, if we only consider the first A of these increments, we get a feasible solution to the Speed Extraction Problem on input A, and this has value at least $(A/B)Q(B)$. □

The next two corollaries follow immediately from the above Lemma.

Corollary 4. *For any integer $i \geq 2$, we have that $\frac{i-1}{Q(i-1)} \leq \frac{i}{Q(i)}$*

Corollary 5. *For any integer n and $0 < \epsilon \leq 1$, $Q\big(\lceil \epsilon n \rceil\big) \geq \epsilon Q(n)$.*

We are now ready to proceed with an amortized analysis. Let $\lambda > 0$ be some constant. Our aim is to show the following equation holds at all times t:

$$2|A(t)| + \mathtt{dt}\Phi(t) \leq 2\lambda |O(t)|. \tag{5.1}$$

We will also show that $\Phi(0) = \Phi(\infty) = 0$, and Φ does not experience any increase at discontinuities. By integrating over time, we can then conclude that the total cost of the online algorithm (flow time plus energy) is at most λ times that of the optimal algorithm. We now consider various cases:

Job Arrival: Consider when job J_i arrives. This job has the largest rank out of all the jobs in $A(t)$. When J_i arrives the rank of every other job remains the same and the terms in Φ corresponding to other jobs do not change. There is a

new term added to the potential function corresponding to job J_i, but we know that $z_i(r_i) = 0$. Hence there is no overall change to the potential function value.
Job Completion: Consider a time t when job J_i completes in the online algorithm. The term in the potential function corresponding to J_i must be 0 since $z_i(C_i) = 0$ by definition. This term drops out of the potential function, causing no change in the potential value. The ranks of all the other jobs which arrive after J_i will decrease by 1, but by Corollary 4, the net change for these terms is negative. Therefore the completion of a job J_i may cause a discontinuity at $\Phi(t)$, but we have ensured that Φ does not increase. Further, it can be seen that when OPT completes a job there is no effect on the potential function.
Job Processing: Here we consider the change in Φ due to the processing of jobs by the algorithm and the optimal solution in an infinitesimally small time interval $[t, t + \mathrm{dt})$ when there are no job arrivals or completions. We will break the analysis into two cases.
`Case (a):` $|O(t)| \geq \epsilon^2 |A(t)|$. In this case, we ignore the change in Φ due to the algorithm's processing. This can be justified since the algorithm's processing can only decrease Φ. We will charge the algorithm's flow time and any increase in the potential function directly to the optimal solution. We first upper bound the increase in Φ. To this end, recall that the most speed OPT uses is $Q(|O(t)|)$ because assumption (A) states that OPTis the GKP algorithm. By Corollary 4, the adversary can increase Φ the most by working on the job with the highest rank. Let $|O(t)| = c|A(t)|$ where $c \geq \epsilon^2$. We obtain the following upper bound on the increase in Φ due to OPT's processing:

$$\frac{2}{\epsilon^2} \frac{|A(t)|}{Q(|A(t)|)} Q(|O(t)|) = \frac{2}{\epsilon^2} |A(t)| \frac{Q(c|A(t)|)}{Q(|A(t)|)}$$

There are two cases. If $c \geq 1$ then we appeal to Lemma 33 and infer that $\frac{2}{\epsilon^2}|A(t)|\frac{Q(c|A(t)|)}{Q(|A(t)|)} \leq \frac{2}{\epsilon^2}|A(t)|c = \frac{2}{\epsilon^2}|O(t)|$. Otherwise, $\frac{2}{\epsilon^2}|A(t)|\frac{Q(c|A(t)|)}{Q(|A(t)|)} \leq \frac{2}{\epsilon^2}|A(t)| \leq \frac{2}{\epsilon^4}|O(t)|$ since Q is non-decreasing and $|A(t)| \leq \frac{1}{\epsilon^2}|O(t)|$. Thus the increase is at most a constant times the optimal solution's current cost. This bound combined with the fact that the algorithm's cost is at most $2|A(t)| \leq \frac{2}{\epsilon^2}|O(t)|$, we get that the term $2|A(t)| + \mathtt{dt}\Phi(t)$ is at most $\frac{4}{\epsilon^4}|O(t)|$. Thus, setting the constant λ from above to be $2/\epsilon^4$ suffices.
`Case (b):` $|O(t)| \leq \epsilon^2 |A(t)|$. In this case, we need to use the potential function to pay for the increase in the algorithm's objective. First consider the change in Φ due to the adversary's processing of jobs. Again by assumption (A), the most speed OPT can use at time t is $Q(|O(t)|)$. By Corollary 4, the largest increase in the potential function would occur when OPT uses all of the power invested on the job with the highest rank. Therefore the largest increase in the potential due to OPT's processing is upper bounded by:

$$\frac{2}{\epsilon^2}\frac{|A(t)|Q(|O(t)|)}{Q(|A(t)|)} \leq \frac{2}{\epsilon^2}\frac{|A(t)|Q(\lceil\epsilon|A(t)|\rceil)}{Q(|A(t)|)}$$
[By definition of Q and $|O(t)| \leq \epsilon^2|A(t)| \leq \lceil\epsilon|A(t)|\rceil$]

Now consider the change in the potential function due to the algorithm's processing. Again, by the definition of our algorithm, we know that the algorithm round robins the $\lceil\epsilon|A(t)|\rceil$ latest arriving jobs on at most $\lceil\epsilon|A(t)|\rceil$ machines whose total speed extracted is $Q(\lceil\epsilon|A(t)|\rceil)$. Let $A'(t)$ be the set of jobs that the algorithm processes. For any job J_i which the algorithm processes, $\text{rank}(i,t) \geq (1-\epsilon)|A(t)|$ and $Q(\text{rank}(i,t)) \leq Q(|A(t)|)$. Further, we know that the z variables decrease for at least $\lceil\epsilon|A(t)|\rceil - \epsilon^2|A(t)|$ jobs since the optimal solution has at most $\epsilon^2|A(t)|$ jobs in its queue by assumption. For these jobs z_i decreases at a rate of $\frac{1}{\epsilon|A(t)|}Q(\lceil\epsilon|A(t)|\rceil)(1+10\epsilon)$ using the fact that the algorithm is given $(1+10\epsilon)$ resource augmentation and the definition of the algorithm. Thus we have that the change in Φ due to the algorithm's processing is at most the following

$$\begin{aligned}
& -\frac{2}{\epsilon^2}\sum_{J_i\in A'(t)\setminus O(t)}\frac{\text{rank}(i,t)}{Q(\text{rank}(i,t))}\cdot\frac{1}{\lceil\epsilon|A(t)|\rceil}Q(\lceil\epsilon|A(t)|\rceil)(1+10\epsilon)\\
\leq\ & -\frac{2}{\epsilon^2}\sum_{J_i\in A'(t)\setminus O(t)}\frac{(1-\epsilon)|A(t)|}{Q(\text{rank}(i,t))}\cdot\frac{1}{\lceil\epsilon|A(t)|\rceil}Q(\lceil\epsilon|A(t)|\rceil)(1+10\epsilon)\\
& \qquad [\text{Since } \text{rank}(i,t) \geq (1-\epsilon)|A(t)| \text{ for } J_i \in A'(t)]\\
\leq\ & -\frac{2}{\epsilon^2}\cdot\frac{(1-\epsilon)|A(t)|}{Q(|A(t)|)}\cdot\frac{1}{\lceil\epsilon|A(t)|\rceil}\sum_{J_i\in A'(t)\setminus O(t)}Q(\lceil\epsilon|A(t)|\rceil)(1+10\epsilon)\\
& \qquad [\text{Since } Q(|A(t)|) \geq Q(\text{rank}(i,t)) \text{ for all } J_i \in A(t)]\\
\leq\ & -\frac{2}{\epsilon^2}\cdot\frac{(1-\epsilon)|A(t)|}{Q(|A(t)|)}\cdot\frac{1}{\lceil\epsilon|A(t)|\rceil}\cdot\left(\lceil\epsilon|A(t)|\rceil - \epsilon^2|A(t)|\right)Q(\lceil\epsilon|A(t)|\rceil)(1+10\epsilon)\\
& \qquad [\text{Since } |A'(t)| = \lceil\epsilon|A(t)|\rceil]\\
\leq\ & -\frac{2}{\epsilon^2}\cdot\frac{(1-\epsilon)^2|A(t)|}{Q(|A(t)|)}\cdot Q(\lceil\epsilon|A(t)|\rceil)(1+10\epsilon)
\end{aligned}$$

Thus the total net change in the potential function is at most,

$$\begin{aligned}
& \frac{2}{\epsilon^2}\frac{|A(t)|Q(\lceil\epsilon|A(t)|\rceil)}{Q(|A(t)|)} - \frac{2}{\epsilon^2}\frac{(1-\epsilon)^2|A(t)|Q(\lceil\epsilon|A(t)|\rceil)(1+10\epsilon)}{Q(|A(t)|)}\\
\leq\ & -\frac{2}{\epsilon}|A(t)|\frac{Q(\lceil\epsilon|A(t)|\rceil)}{Q(|A(t)|)} \qquad [\text{Since } \epsilon \leq 1/10]\\
\leq\ & -2|A(t)| \qquad [\text{By Corollary 5}]
\end{aligned}$$

Thus, the net change in the potential function plus the increase in the algorithm's objective is non-positive. So this gives us the restriction that $\lambda \geq 0$. Therefore, we get that $\lambda = \frac{2}{\epsilon^4}$ suffices in all cases.

For the final analysis, we add the upper bound on the change for each of the cases we studied above. Let $\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t)$ denote the change (rate) of $\Phi(t)$, $\mathtt{dt}\mathrm{LAPS}(t)$ denote the change of our algorithm's objective and $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$ denote the change in the optimal solution's objective. We have that $\mathtt{dt}\mathrm{LAPS}(t) + \mathtt{dt}\Phi(t) \leq \frac{2}{\epsilon^4}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$ by the previous arguments. Thus,

$$\begin{aligned}
\mathrm{LAPS} &= \int_0^\infty \Big(\mathtt{dt}\mathrm{LAPS}(t)\Big)\mathrm{dt} \\
&= \int_0^\infty \Big(\mathtt{dt}\mathrm{LAPS}(t) + \mathtt{dt}\Phi(t)\Big)\mathrm{dt} \\
&\qquad [\text{Since } \Phi(0) = \Phi(\infty) = 0] \\
&\leq \int_0^\infty \Big(\frac{2}{\epsilon^4}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)\Big)\mathrm{dt} = \frac{2}{\epsilon^4}\mathrm{OPT}
\end{aligned}$$

However, since we assumed that OPT runs the GKP algorithm (which is in itself $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive from [56]), we get that the overall competitive ratio of our non-clairvoyant algorithm is $O(1/\epsilon^5)$. We stress that we have not tried to optimize the competitive ratio but rather to show that the related machines setting admits a non-clairvoyant scalable algorithm.

Theorem 8. *The algorithm LAPS is $(1+\epsilon)$-speed $O(1/\epsilon^5)$-competitive for the problem of flow time plus energy on related machines.*

5.4 Lower Bounds on Weighted Flow Time on Related Machines

In this section we show that the standard priority algorithms for the weighted flow objective, namely HDF, WSETF, and the most natural adaptation of WLAPS (*weighted latest arrival processor sharing*), are not $O(1)$-speed $O(1)$-competitive for total weighted flow time on uniformly related machines. This is the heterogeneous processor setting where each machine runs at a fixed speed and consumes no power. When each machine consumes no power, the objective just simplifies to minimizing the total weighted flow time. As previously stated, this is a special case of the speed scaling heterogeneous processor setting with the objective of total weighted flow time plus energy. In each of the following subsections, we first explain how these priority algorithms generalize to heterogeneous machines, and then provide the lower bound examples. Finally, in Section 5.4.4, in the heterogeneous processors setting, the performance of SRPT and SJF cannot be shown via the popular local competitiveness argument as was the case in the homogeneous processors setting.

5.4.1 Lower Bound for Highest Density First (HDF)

In HDF, the priority of a job is its density, i.e., job J_i has a priority equal to its weight divided by its size $\frac{w_i}{p_i}$. The algorithm on a single processor, always

schedules the highest density job. This naturally extends to related machines by scheduling the job of the k^{th} highest density on the k^{th} fastest machine at all times. The work of [19] shows that this algorithm is $O(1)$-speed $O(1)$-competitive when all machines have the same speed. However, when the speeds can be different, the following example shows that HDF has unbounded competitive ratio on related machines, even when provided any constant speed augmentation.

Theorem 9. *For any constants $\alpha, B > 0$, there exists an instance $\mathcal{I}(\alpha, B)$ of related machines scheduling for which HDF is not B-competitive for the objective of weighted flow with a speed augmentation of α.*

Proof. For this proof, let cost(A) denote the total weighted flow time of an algorithm A. The instance $\mathcal{I}$ is defined in the following manner. There is a "fast" machine of speed S, and infinitely many "slow" machines of speed 1. At time $t = 0$, a "heavy" job of weight W and length L arrives. Then, at each time $\frac{i}{\alpha S}$, for integer $0 \leq i < SL$, a "small" job of weight $w = 4W/L$ and length 1 arrives (all the parameters S, W, L will be set appropriately when required).

Note that each small job has a density w, which is greater than the density of the heavy job, W/L. Hence as long as there is a small job that is unfinished, the heavy job will not run on the fast machine in HDF's schedule. Now, since each small job completes on the fast machine after a time of $\frac{1}{\alpha S}$, the next small job arrives as soon as its preceding small job is finished by HDF by the way we have set up the instance. This is repeated until all small jobs complete and takes exactly $SL\frac{1}{\alpha S} = \frac{L}{\alpha}$ units of time. This implies that the heavy job is processed entirely on slow machines by HDF, and as a result, HDF incurs a weighted flow time of at least cost(HDF) $\geq W\frac{L}{\alpha}$.

The optimal solution, however, will run the heavy job on the fast machine until completion, and run each small job on a *dedicated* unit speed machine. Recall that there are enough slow machines to run all small jobs simultaneously. The cost of the optimal solution is cost(OPT) $= W\frac{L}{S} + wSL = W\frac{L}{S} + 4WS$. We set the length of the heavy job so that $W\frac{L}{S} = 4SW$, i.e., $L = 4S^2$. This implies a lower bound on the competitive ratio of $W\frac{L}{\alpha}\Big/2W\frac{L}{S} = \frac{S}{2\alpha}$. To complete the proof, we set $S = 4\alpha B$. □

5.4.2 A Lower Bound for Weighted Shortest Elapsed Time First (WSETF)

In this section we show a lower bound on the well-known algorithm WSETF (*weighted shortest elapsed first*) in the related machines setting for total weighted flow time. We begin by describing the algorithm: at any time t, let $q_j(t)$ denote the the amount of work that job J_j has been processed by. For any unfinished job J_j, define its priority at time t to be $w_j/q_j(t)$. Then WSETF assigns the job with the i^{th} highest priority on the i^{th} fastest machine. We remark that this algorithm is scalable on a single processor [8]. We now show that an instance quite similar to the bad example for HDF is also bad for WSETF.

Theorem 10. *For any constants $\alpha, B > 0$, there exist related machine instances $\mathcal{I}(\alpha, B)$ where WSETF is not B-competitive for weighted flow with a speed augmentation of α.*

Proof. For this proof, let cost(A) denote the total flow time of an algorithm A. The instance $\mathcal{I}$ is defined in the following manner and is similar to the lower bound on HDF. There is a "fast" machine of speed S, and infinitely many "slow" machines of speed 1. At time $t = 0$, a "heavy" job of weight W and (unknown) length L arrives. Then, at each time $\frac{L}{2\alpha S} + \frac{i}{\alpha S}$, for integer $0 \leq i < \frac{SL}{2}$, a "small" job of weight w and length 1 arrives (all the parameters S, W, L, w will be set appropriately when required). For notational convenience, we will assume that $\frac{SL}{2}$ is an integer.

At time $t = \frac{L}{2\alpha S}$, the priority of the heavy job is $\frac{W}{L/2} = \frac{2W}{L}$, since it has run on the fast machine and there is α speed augmentation for WSETF. Note that by definition, the priority of any job can only decrease over time. We now set the value of w such that the worst-case priority of a small job (i.e., when it completes) is larger than this quantity. This implies that as long as a small job is unfinished, the heavy job can not run on the fast machine. The condition required for this is $\frac{w}{1} \geq \frac{2W}{L}$. We therefore set $w = \frac{2W}{L}$. Since each small job completes on the fast machine in $\frac{1}{\alpha S}$ time steps, a small job arrives as soon as its preceding small job is finished by WSETF. This will be repeated until all small jobs complete. This takes exactly $\frac{SL}{2}\frac{1}{\alpha S} = \frac{L}{2\alpha}$ units of time. This implies that the entire second half of the heavy job is processed on slow machines by WSETF. Thus, WSETF incurs a weighted flow time of at least cost(WSETF) $\geq W\frac{L}{2\alpha}$.

The optimal solution, however, will run the heavy job on the fast machine until completion, and run each small job on a *dedicated* unit speed machine. Recall that there are enough slow machines to accommodate all small jobs simultaneously. The cost of the optimal solution is then cost(OPT) $= W\frac{L}{S} + \frac{SL}{2}w = W\frac{L}{S} + SW$. We set the length of the heavy job so that $W\frac{L}{S} = SW$, i.e., $L = S^2$. This implies a lower bound on the competitive ratio of $W\frac{L}{2\alpha}\Big/2W\frac{L}{S} = \frac{S}{4\alpha}$. To complete the proof, we simply set $S = 8\alpha B$. □

5.4.3 A Lower Bound for Weighted Latest Arrival Processor Sharing (WLAPS)

Finally, in this section we show a lower bound on WLAPS. In order to simplify the presentation, we describe a lower bound instance for the Weighted Processor Sharing (WPS) algorithm (or Equipartition or Round-Robin) for total weighted flow time on related machines. Subsequently, we explain how this also translates to a lower bound for WLAPS. This is because WLAPS can be shown to always be dominated by WPS when WPS is given a constant amount of resource augmentation over WLAPS by definition of the algorithms. As usual, we begin with the algorithm description.

On a single fixed-speed machine, at any time, the algorithm WPS works on a job J_j with weight w_j at a speed of its "fair share", i.e., a fraction $\frac{w_j}{W}$ of the speed where W is the total weight of unfinished jobs. How do we generalize this to multiple related machines? Ideally, we would like to process job J_j at a rate of $\frac{w_j}{W}S$, where S is the total speed of the fastest n machines where n is the number of unfinished jobs. However, this may not always be achievable. For example, if there is a single job J_j with very large weight and $n-1$ jobs of negligible weight then job J_j can only be processed at the speed of the fastest processor because a job can only be processed by a single processor at any point in time. This is much less than its fair share of the fastest n processors. As a result, the most natural extension of WPS to the setting of heterogeneous processors, is to assume that each job J_j is given $\frac{w_j}{W}$ share of the $\lfloor \frac{W}{w_j} \rfloor$ fastest processors. Therefore, job J_j is processed with a total speed of $\sum_{i=1}^{\lfloor W/w_j \rfloor} \frac{w_j\, s_i}{W}$, where the machines are ordered in decreasing order of speed. It is not difficult to see that this scheduling policy can be achieved without scheduling a job on more than one machine at the same time. (Essentially, every job is scheduled to an extent of 1 across machines, and every machine has utilization of at most 1. Then we can decompose this fractional assignment into a convex combination of integer schedules, and then preemptively follow this combination). One can define the WLAPS algorithm in a similar fashion: WLAPS uses the WPS algorithm assuming that the latest ϵ fraction of the unsatisfied jobs that arrived the latest are the only jobs in the queue where $0 \leq \epsilon \leq 1$ is a constant that parametrizes WLAPS.

Theorem 11. *For any constants $\alpha, B > 0$, there exist related machine instances $\mathcal{I}(\alpha, B)$ where* WPS *is not B-competitive for weighted flow with any constant speed augmentation α.*

Proof. Consider the following instance. There are n jobs, with job J_j having a weight of $w_j = 1/j$ and length l_j which will be determined later. There are n machines with machine i with speed $1/\sqrt{i}$. We will set the lengths of the jobs in such a way that all jobs complete at the same time in WPS.

We can bound the cost of WPS as follows. For such an instance, it is easy to see that the total speed at which job J_j is processed by the WPS algorithm is exactly $\alpha \sum_{i=1}^{jH_n} \frac{1}{jH_n} \frac{1}{\sqrt{i}}$ where α is the speed augmentation WPS has over the optimal solution. Set l_j to be precisely the above sum so that WPS completes all the jobs at exactly $t = 1$. Thus, WPS incurring a weighted flow time of $\sum_j w_j = H_n = \Omega(\log n)$.

Now we bound the cost of the optimal solution. Consider the following alternate schedule which simply schedules job J_j on machine j. Noticing that $l_j = \alpha \frac{1}{jH_n} \sum_{i=1}^{jH_n} \frac{1}{\sqrt{i}} \leq O(\frac{\alpha}{\sqrt{jH_n}})$, we get that the weighted flow time for this schedule is at most

$$\sum_j \frac{w_j l_j}{s_j} = \sum_j \frac{1}{\sqrt{j}} l_j \leq \frac{O(\alpha)}{\sqrt{H_n}} \sum_j \frac{1}{j} \leq O(\alpha\sqrt{\log n})$$

which gives the $\Omega(\sqrt{\log n})$ bound on the competitive ratio of WPS for any speed augmentation of α. □

The careful reader might observe that the above algorithm does not utilize every machine to an extent of 1. Indeed, consider the example of one heavy job and a large number of very small jobs. Then the utilization of machines $3, 4, \ldots$ is negligibly small because each tiny job only occupies w_j/W of these machines while much of the total weight in W comes from the heavy job (even though the heavy job is never going to be scheduled on these slower machines). A better algorithm with possibly better performance is one where we re-weight the jobs' fair shares on each machine depending on which jobs have not yet been scheduled to full utilization. However, it can be shown that the above example has an unbounded competitive ratio even for this modified algorithm.

Now we show how the previous lemma extends to lower bound the performance of WLAPS.

Corollary 6. *For any constants $\alpha, B > 0$, there exist related machine instances $\mathcal{I}(\alpha, B)$ where WLAPS is not B-competitive for weighted flow with any constant speed augmentation α.*

Proof. Note that by definition of WPS and WLAPS, when WPS is given a constant factor greater resource augmentation WPS schedule can only be better for total flow time than WLAPS. In particular, this holds when WPS is given more than a $\frac{1}{\epsilon}$ factor greater resource augmentation over WLAPS where ϵ is the constant that parametrizes WLAPS. Thus a lower bound of c on the competitive ratio of WPS for any constant resource augmentation this implies a lower bound of c on the competitive ratio of WLAPS for any constant resource augmentation. □

5.4.4 Local Competitiveness Lower Bounds

A scheduling algorithm A is said to be locally competitive if the number (or total weight) of unfinished jobs at any time t under A's schedule is comparable to the number (or total weight) of unfinished jobs in the optimal schedule. Local competitiveness implies that the algorithm's competitive ratio can be bounded for (weighted) flow time because the (weight) number of the unsatisfied jobs in a schedule at some time is the instantaneous increase in the objective at that time. Many scheduling algorithms have been proved to have bounded competitiveness using a local competitiveness argument. In particular, SRPT and SJF can be shown to be scalable on identical parallel machines via a local competitiveness argument. We however show that these algorithms are not locally competitive on related machines even with any constant speedup. Recall that in the related machines setting machines/processors have different fixed speeds and consume not power. We show that local competitiveness cannot be shown even in the unweighted setting.

Theorem 12. *For any $s \geq 1$, assume that SRPT or SJF is given s-speed augmentation. Then there exists a schedule and time t such that the schedule finished all jobs at time t while SRPT or SJF has unsatisfied jobs.*

Proof. We first describe the instance. There are $k+1$ groups of machines, $\mathcal{M}_0, \mathcal{M}_1, ..., \mathcal{M}_k$. Group $\mathcal{M}_i$ has $h^{2(k-i)}$ machines of speed h^i where h is a sufficiently large constant. There are $k+1$ groups of jobs, $\mathcal{J}_0, \mathcal{J}_1, ..., \mathcal{J}_k$. Group $\mathcal{J}_i$ has $h^{2(k-i)}$ jobs of size h^i. For notational convenience, we will use subscript to denote a subset of groups. For example, $\mathcal{M}_{\geq i} = \bigcup_{i' \geq i} \mathcal{M}_{i'}$.

Note that one can finish all jobs by time 1 by scheduling each job in $\mathcal{J}_i$ on one machine in $\mathcal{M}_i$. We will show that SRPT cannot finish all jobs by time $\frac{k}{s}$. Then the theorem follows by setting $k = s+1$ and $t = 1$. At time 0, SRPT fills all machines in $\mathcal{M}_{\geq 1}$ with the jobs in $\mathcal{J}_0$. This is because jobs in $\mathcal{J}_0$ are the shortest jobs and $|\mathcal{J}_0| = h^{2k} > \sum_{i=1}^{k} h^{2(k-i)} = |\mathcal{M}_{\geq 1}|$. Until time $\frac{1}{2s}$, no job in $\mathcal{J}_0$ can be finished by SRPT unless it is processed on one of the machines in $\mathcal{M}_{\geq 1}$. The total volume of jobs in $\mathcal{J}_0$ that can be processed on $\mathcal{M}_{\geq 1}$ for $\frac{1}{2s}$ time units by s speed resource augmented machines is at most $\frac{1}{2}\sum_{i=1}^{k} h^{2(k-i)}h^i = \frac{1}{2}\sum_{i=1}^{k} h^{2k-i} \leq h^{2k-1}$. The last inequality holds for sufficiently large h. Further, machines in $\mathcal{M}_0$ can process at most $h^{2k}/2$ volume of jobs in $\mathcal{J}_0$ during $[0, \frac{1}{2s}]$. Hence we have a lower bound $h^{2k} - h^{2k}/2 - h^{2k-1} = h^{2k}/2 - h^{2k-1}$ on the total remaining volume of the unfinished jobs in $\mathcal{J}_0$ at time $\frac{1}{2s}$. This is because the total volume of jobs in $\mathcal{J}_0$ is h^{2k} and a total volume of at most $h^{2k}/2 + h^{2k-1}$ can be processed during $[0, \frac{1}{2s}]$ by SRPT with s resource augmentation. Since this lower bound is larger than $|\mathcal{M}_{\geq 1}|$, we know that during $[0, \frac{1}{2s}]$, all jobs in $\mathcal{J}_{\geq 1}$ were scheduled only on machines in $\mathcal{M}_0$. It is easy to see that each job in $\mathcal{J}_{\geq 1}$ has been processed by a fraction of at most $\frac{1}{h}$.

The remaining proof can be completed by repeating this argument. Formally, one can show the following: At time $\frac{\ell}{2s}$ for integer $1 \leq \ell \leq k$, each job in $\mathcal{J}_{\geq \ell}$ has remaining size that is at least $(1-1/h)^\ell$ times its initial size. The proof for SJF is the same, since SJF and SRPT produce the same schedule on any instance where all jobs arrive at the same time by definition of the algorithms. □

5.5 Concluding Remarks

In this chapter, we gave the first non-clairvoyant scalable algorithm for minimizing total (unweighted) flow time plus total energy consumed on (related) heterogeneous machines. In contrast, for the weighted objective, we do not have any algorithm that is $O(1)$-speed $O(1)$-competitive algorithm even when machines have fixed speeds. We have already shown that natural/popular algorithms such as HDF, WSETF or WLAPS do not work. The only candidate algorithm we have is a variant of the algorithm by Chadha et al. [24]. Roughly speaking, their clairvoyant algorithm assigns each incoming job to the machine that gives the minimum increase of the total fractional weight flow time. One

obvious way of converting this algorithm into a non-clairvoyant one is to guess job sizes: Initially assume each job has a unit size, and double the size whenever necessary, i.e., assume that the size is 2^{l+1} if it is not finished after processing it for 2^l time units. However, this completely destroys the analysis framework used in [24]. The recent paper [3] that gives an alternate proof based on dual fitting seems to have a similar problem.

Open Problem 6. *For the problem of minimizing weighted flow time on related machines (when machines have fixed speeds), find an $O(1)$-speed $O(1)$-competitive algorithm, or show that no such algorithm exists.*

Another interesting problem is to investigate the performance of some fixed priority algorithms such as SRPT or SJF for the unweighted objective. We believe that they are scalable at least when the machines have fixed speeds. The main difficulty in the analysis comes from the fact that we do not have a clean mathematical expression for total flow time even when all jobs arrive at time zero.

Conjecture 3. *For the problem of minimizing weighted flow time on related machines (when machines have fixed speeds), the algorithms SRPT and SJF are scalable.*

Chapter 6

Future Research Directions

Since the celebrated resource augmentation analysis model was introduced [69], a large amount of work has been done in this model. Another important milestone in the history of scheduling research was the development of potential functions for online scheduling [11, 67]. Potential functions enabled a large number of analyses to be performed in online scheduling where local competitiveness cannot be used. This dissertation aligns with this research direction and presents several scalable algorithms in various settings. Most of our results are based on potential function arguments.

Although most of potential functions for online scheduling seem to have a certain standard form (see Section 1.5.2), it would be fair to say that our understanding of potential functions is limited. For example, consider the problem in Chapter 3 of minimizing the ℓ_k norms of flow time for jobs of different parallelizability. It is not clear that the potential function used is the "right" one in capturing the discrepancy between the online algorithm's status and the optimal scheduler's status, although it yields the analysis of a scalable algorithm. Particularly, we do not know if the competitive ratio should exponentially grow with k. Or it may be the case that our competitive ratio is the best that can be shown via the "standard" potential functions.

We believe that our understanding of potential functions could substantially improve by bridging the techniques in offline and online settings, which at first sight seems somewhat irrelevant. In the online setting, the popular techniques are local competitiveness argument and potential functions. In the offline setting, more diverse techniques have been used such as linear programming relaxation, rounding, and dynamic programming. Two recent works by Anand et al. [3] and Günther et al. [54] show how online scheduling can benefit from the algorithmic ideas and analysis tools in the offline setting.

Anand et al. [3] used linear program and dual fitting to give a fairly clean analysis of several interesting results. Very interestingly, using dual fitting, they gave an alternative proof of the breakthrough result in [24]. Based on a novel potential function, [24] showed that a natural algorithm is scalable for minimizing average flow time in the unrelated machines setting (see Chapter 4). In contrast to its conciseness and elegance, the potential function gave little insight on why such a simple algorithm works. Dual fitting in [3] seems to give

a more natural explanation for this. This primal-dual type analysis for online scheduling was recently further explored in [58].

Another good example is the recent work by Günther et al. [54]. Let us focus on one of the problems they consider, namely minimizing total weighted completion time, i.e. $\sum_j w_j C_j$, on a single machine where preemption is allowed. Recall that w_j, r_j and C_j denote job j's weight, release time and completion time, respectively. This problem admits a constant completive algorithm, and there is a constant lower bound known [45, 92]. There was a gap between the upper and lower bounds before [54]. To remove this gap, [54] exploits a variety of techniques that were developed in the offline setting. This was enabled by their novel idea that simplifies online instances so that only a finite set of future jobs need to be considered. As a result, they were able to obtain an online algorithm whose competitive ratio is arbitrarily close to the optimal competitive ratio. It would be interesting if one can extend their approach to more difficult objectives such as minimizing total weighted flow time, i.e., $\sum_j w_j(C_j - r_j)$.

Finally, we believe that studying the power of knowing the speed that the algorithm is given will be an important research direction. As mentioned earlier, some of recently found scalable algorithms require the knowledge of the speed and are parameterized by the speed [42, 66, 33, 44, 59]. Those algorithms are not as natural as other algorithms that do not change with the speed. In fact, all the scalable algorithms we develop and present in this dissertation are as such. We do not know how crucial the knowledge of the speed is in obtaining a scalable algorithm. Edmonds conjectures that any deterministic algorithm must know the speed for minimizing average flow time for jobs of different parallelizability, and makes an attempt to prove his conjecture [39].

Bibliography

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

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

[3] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA '12: Proceedings of the Twenty-thrid Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1228–1241, 2012.

[4] Lachlan L. H. Andrew, Minghong Lin, and Adam Wierman. Optimality, fairness, and robustness in speed scaling designs. In *SIGMETRICS '10: Proceedings of the 2010 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 37–48, 2010.

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

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

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

[8] Nikhil Bansal and Kirk Pruhs. Server scheduling in the L_p norm: a rising tide lifts all boat. In *STOC '03: Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, pages 242–250, 2003.

[9] Nikhil Bansal and Kirk Pruhs. Server scheduling in the weighted l_p norm. In *LATIN '04: Proceedings of the Sixth Latin American Theoretical Informatics*, pages 434–443, 2004.

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

[11] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007.

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

[13] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *SODA '09: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 693–701, 2009.

[14] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM J. Comput.*, 39(4):1294–1308, 2009.

[15] Nikhil Bansal, Ravishankar Krishnaswamy, and Viswanath Nagarajan. Better scalable algorithms for broadcast scheduling. In *ICALP '10: Proceedings of the Thirty-seventh International Colloquium on Automata, Languages and Programming*, pages 324–335. Springer, 2010.

[16] 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.

[17] Yair Bartal and S.Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *SODA '00: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 558–559. SIAM, 2000. ISBN 0-89871-453-2.

[18] 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.

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

[20] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *SODA '98: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–279, 1998.

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

[22] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[23] F.A. Bower, D.J. Sorin, and L.P. Cox. The impact of dynamically heterogeneous multicore processors on thread scheduling. *Micro, IEEE*, 28(3):17 –25, may-june 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 *STOC '09: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, pages 679–684, 2009.

[25] Ho-Leung Chan, Jeff Edmonds, Tak Wah Lam, Lap-Kei Lee, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, pages 255–264, 2009.

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

[27] Sze-Hang Chan, Tak Wah Lam, and Lap-Kei Lee. Non-clairvoyant speed scaling for weighted flow time. In *ESA '10: Proceedings of the Eighteenth Annual European Symposium on Algorithms*, pages 23–35, 2010.

[28] 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 *COCOON '04: Proceedings of the Tenth Annual International Computing and Combinatorics Conference*, pages 210–218, 2004.

[29] Jessica Chang, Thomas Erlebach, Renars Gailis, and Samir Khuller. Broadcast scheduling: algorithms and complexity. In *SODA '08: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 473–482, 2008.

[30] Moses Charikar and Samir Khuller. A robust maximum completion time measure for scheduling. In *SODA '06: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, pages 324–333, 2006.

[31] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *STOC '04: Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, pages 363–372, 2004.

[32] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Longest wait first for broadcast scheduling [extended abstract]. In *WAOA '09: Proceedings of the Seventh Workshop on Approximation and Online Algorithms*, pages 62–74. Springer, 2009.

[33] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Minimizing maximum response time and delay factor in broadcast scheduling. In *ESA '09: Proceedings of the Seventeenth Annual European Symposium on Algorithms*, pages 444–455. Springer, 2009. The journal version will appear in Theory of Computing, Special Issue in Honor of Rajeev Motwani.

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

[35] Marek Chrobak, Christoph Dürr, Wojciech Jawor, Lukasz Kowalik, and Maciej Kurowski. A note on scheduling equal-length jobs to maximize throughput. *J. of Scheduling*, 9(1), 2006.

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

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

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

[39] Jeff Edmonds. Every deterministic nonclairvoyant scheduler has a suboptimal load threshold. Manuscript, 2010.

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

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

[42] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 685–692, 2009.

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

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

[45] Leah Epstein and Rob van Stee. Lower bounds for on-line single-machine scheduling. *Theor. Comput. Sci.*, 1-3(299):439–450, 2003.

[46] 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.

[47] Kyle Fox and Benjamin Moseley. Online scheduling on identical machines using SRPT. In *SODA '11: Proceedings of the Twenty-first Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 120–128, 2011.

[48] 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.

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

[50] Naveen Garg and Amit Kumar. Minimizing average flow time on related machines. In *STOC '08: Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, pages 730–738, 2006.

[51] Naveen Garg and Amit Kumar. Better algorithms for minimizing average flow-time on related machines. In *Proceedings of the Thirty-thrid International Colloquium on Automata, Languages and Programming*, pages 181–190, 2006.

[52] Naveen Garg and Amit Kumar. Minimizing average flow-time : Upper and lower bounds. In *FOCS' 07: Proceedings of the Forty-eighth Annual IEEE Symposium on Foundations of Computer Science*, pages 603–613, 2007.

[53] Naveen Garg, Amit Kumar, and V. N. Muralidhara. Minimizing total flow-time: The unrelated case. In *ISAAC '08: Proceedings of the Nineteenth International Symposium on Algorithms and Computation*, pages 424–435, 2008.

[54] Elisabeth Günther, Olaf Maurer, Nicole Megow, and Andreas Wiese. A new approach to online scheduling: Approximating the optimal competitive ratio. *CoRR*, abs/1204.0897, 2012.

[55] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling jobs with varying parallelizability to reduce variance. In *SPAA '10: Proceedings of the Twenty-second ACM Symposium on Parallelism in Algorithms and Architectures*, pages 11–20, 2010.

[56] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Nonclairvoyantly scheduling power-heterogeneous processors. In *Green Computing Conference*, pages 165–173, 2010.

[57] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Scalably scheduling power-heterogeneous processors. In *ICALP '10: Proceedings of the Thirty-seventh International Colloquium on Automata, Languages and Programming*, pages 312–323, 2010.

[58] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. *CoRR*, abs/1109.5931, 2011.

[59] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *SODA '12: Proceedings of the Twenty-thrid Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1242–1253, 2012.

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

[61] Leslie A. Hall. Approximation algorithms for np-hard problems. chapter Approximation algorithms for scheduling, pages 1–45. PWS Publishing Co., Boston, MA, USA, 1997. ISBN 0-534-94968-1.

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

[63] Nikos Hardavellas. Exploiting dark silicon for energy efficiency, 2011. Article to appear in IEEE Micro.

[64] Sungjin Im and Benjamin Moseley. An improved analysis of longest wait first. Manuscript, 2009.

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

[66] Sungjin Im and Benjamin Moseley. Online scalable algorithm for minimizing ℓ_k-norms of weighted flow time on unrelated machines. In *SODA '11: Proceedings of the Twenty-first Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 95–108, 2011.

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

[68] Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

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

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

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

[72] David Karger, C. Stein, and Joel Wein. Scheduling algorithms. In M.J. Atallah, editor, *Handbook on Algorithms and Theory of Computation*, chapter 34. 1999.

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

[74] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi, and K.I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *ISCA '04: Proceedings of the Thirty-first Annual International Symposium on Computer Architecture*, pages 64–75, 2004.

[75] Rakesh Kumar, Dean M. Tullsen, and Norman P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *PACT '06: Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, pages 23–32, New York, NY, USA, 2006.

[76] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Competitive non-migratory scheduling for flow time and energy. In *SPAA '08: Proceedings of the Twentieth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 256–264, 2008.

[77] Eugene L. Lawler, Jan Karel Lenstra, Alexander H.G. Rinnooy Kan, and David B. Shmoys. Chapter 9 sequencing and scheduling: Algorithms and complexity. In A.H.G. Rinnooy Kan S.C Graves and P.H. Zipkin, editors, *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445 – 522. Elsevier, 1993.

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

[79] Joseph Y-T. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Boca Raton, FL, USA, 2004.

[80] R. Merritt. CPU designers debate multi-core future. EE Times, Feb. 2010.

[81] Tomer Y. Morad, Uri C. Weiser, Avinoam Kolodny, Mateo Valero, and Eduard Ayguade. Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors. *IEEE Comput. Archit. Lett.*, 5:14–17, 2006.

[82] Benjamin Moseley. Scheduling to minimize energy and flow time in broadcast scheduling. *CoRR*, abs/1007.3747, 2010.

[83] Rajeev Motwani, Steven Phillips, and Eric Torng. Non-clairvoyant scheduling. *Theor. Comput. Sci.*, 130(1):17–47, 1994.

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

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

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

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

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

[89] Julien Robert and Nicolas Schabanel. Non-clairvoyant batch sets scheduling: Fairness is fair enough. In *ESA '07: Proceedings of the Fifteenth Annual European Symposium on Algorithms*, pages 741–753, 2007.

[90] Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *SODA '08: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 491–500, 2008.

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

[92] René Sitters. Competitive analysis of preemptive single-machine scheduling. *Oper. Res. Lett.*, 38(6):585–588, 2010.

[93] René A. Sitters. Approximation and online algorithms. chapter Minimizing Average Flow Time on Unrelated Machines, pages 67–77. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-540-93979-5.

[94] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[95] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[96] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2007. ISBN 9780136006633.

[97] Eric Torng and Jason McCullough. SRPT optimally utilizes faster machines to minimize flow time. *ACM Trans. Algorithms*, 5:1:1–1:25, December 2008.

[98] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.

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

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

[101] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

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

[103] 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 *COCOON '06: Proceedings of the Twelfth Annual International Computing and Combinatorics Conference*, pages 320–329. Springer, 2006.