This is to certify that the
thesis entitled

EFFECTIVE ALGORITHMS FOR MINIMIZING WEIGHTED AND UNWEIGHTED FLOW TIME IN SINGLE AND MULTIPROCESSOR ENVIRONMENTS

presented by

CARL BUSSEMA III

has been accepted towards fulfillment
of the requirements for the

M.S. degree in Computer Science

[illegible]
Major Professor's Signature

8/14/07

Date

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.
MAY BE RECALLED with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE
FEB 2 1 2009		
08 31 09		

6/07 p:/CIRC/DateDue.indd-p.1

EFFECTIVE ALGORITHMS FOR MINIMIZING WEIGHTED AND UNWEIGHTED FLOW TIME IN SINGLE AND MULTIPROCESSOR ENVIRONMENTS

By

Carl Bussema III

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Computer Science

2007

ABSTRACT

EFFECTIVE ALGORITHMS FOR MINIMIZING WEIGHTED AND UNWEIGHTED FLOW TIME IN SINGLE AND MULTIPROCESSOR ENVIRONMENTS

By

Carl Bussema III

We consider two problems of online scheduling of jobs to minimize different metrics in different environments. First, we attempt to minimize the L_p norms of flow time and stretch on m identical machines. This is a hard problem for which no constant competitive online algorithms are known. We consider the case where the online algorithm is given slightly faster machines or slightly more machines than the optimal offline algorithm. Using resource augmentation, we show that the greedy algorithm Shortest Job First (SJF) is $(1 + \epsilon)$-speed $O(1)$-competitive for minimizing the unweighted L_p norms of flow time and stretch on m identical machines. We also extend this result to apply to the Shortest Remaining Processing Time (SRPT) algorithm and for the case where SJF and SRPT are given extra machines. From that result, we derive that the greedy Highest Density First (HDF) algorithm is $(1 + \epsilon)$-speed $O(1)$-competitive for minimizing the weighted L_p flow time on m identical machines. This is the first result which proves a constant factor competitive ratio for this problem with arbitrarily small resource augmentation. Second, we also analyze the related algorithm, Highest Remaining Density First (HRDF) in a uniprocessor environment, and show that it is $\Omega(P)$-competitive for weighted flow time without augmentation.

TABLE OF CONTENTS

Chapter 1
Introduction to Scheduling

In this chapter, I will give a broad overview of the problem of scheduling, explain and define notation and terminology for the parameters that influence particular problems, define the specific problems I will cover in this thesis, summarize prior work in this field, and finally give the results that will be explored in depth in the remaining chapters.

1.1 Scheduling Overview

Scheduling is a fundamental problem that is both theoretical and practical. It is studied in many fields: computer science as it is here, math for the algebra and problem solving aspects, business for the application of managing human or machine resources efficiently, and logistics for operational environments and applications. Each field takes a slightly different approach and emphasizes a different aspect of the problem, but at the core, all study the same problems.

A schedule involves a set of jobs, or work to be done, one or more machines or processors, and an assignment for each machine of which job to run at any given time. In the theoretical model for studying scheduling problems, time may be abstracted to a series of unit-length intervals, during which each machine may work on only one job. The minimum time required to finish any given job is some number of such intervals. The speed of a machine, typically abstracted to "1", determines how much work each machine accomplishes during each time interval.

The most important variable in a problem is whether your schedule is "offline," where you have full knowledge of all jobs that will be released in the future before starting, or "online," where a schedule can only use information about jobs released at or before the current time step, and has no knowledge of jobs in the future. It is easy to see that, all else being equal, an offline algorithm, having more information, would be able to do at least as well as an online algorithm for a given problem. However, the offline version of many scheduling problems, including the ones we will study in this paper, are NP-hard.

Some online algorithms operate under the assumption that the time it will take to complete a job is known to the algorithm as soon as the job is released, a concept known as "clairvoyance," while a non-clairvoyant algorithm does not know how much time is required to complete a job. The clairvoyant setting is similar to an environment found in many applications: a file server could consider the size of the file to be delivered and the speed of the network and disks; a database server could estimate the size of the query by looking at the tables involved; a web server with dynamic pages might estimate based on a sum of a file transfer calculation and a database calculation. In all cases, online is vastly more appropriate than offline, since rarely do servers operate by taking all requests at once and then closing to new requests.

All algorithms we study in this paper will be online and clairvoyant. More formal definitions can be found in Section 1.4.

1.2 Analysis Techniques

Schedules can be evaluated in many different ways, but these all divide into metrics that are machine-centric and seek to optimize the server resources, or metrics that are client-centric and seek to optimize the experience for clients requesting the service. Historically, machine-centric criteria like makespan (the total time required between the start of the first job and the end of the last job) were more commonly studied, but recently client-centric evaluations have been increasingly presented as alternatives. Among client-centric criteria, one of the most obvious ones to use is the **total flow time** metric, also called **wait time** or **response time**, where the flow time of a job is its completion time minus its release time. However, algorithms that are optimal for minimizing total flow time often starve individual jobs, which is undesirable. To alleviate that problem, some recent work has focused on the L_p norms of flow time, and so we also examine this metric in more detail later. Related to flow time is the metric of stretch, which is the flow time of a job divided by a job's processing time (the actual time required to complete the job). Another client-centric metric is the weighted flow time, where a job's flow time is multiplied by the weight, or priority, of a job. As with unweighted flow time, recent work has begun to study the L_p norms of weighted flow time as well. In this paper, we will study only client-centric scheduling criteria.

The flow time may be intuitively considered the time a job spends waiting between release and completion. A large job necessarily has a large flow time, since it must wait at least as long as it would take to run the job completely. For that reason, the stretch is also considered, which may be thought of as the perceived wait time of a job: a user would expect their long job to take a long time; if that expectation is directly proportional to the length of the job, then stretch adequately encapsulates that perception. Weighted flow time measures the severity of the delay by allowing a low-priority job to be delayed a long time with minimal penalty, while a high-priority job must be run quickly to avoid falling behind.

In order to fairly evaluate any online algorithm, theorists measure an algorithm's performance against OPT, a theoretical optimal offline algorithm. OPT is assumed to have perfect knowledge of all jobs beforehand; i.e., OPT may act as if it were offline and clairvoyant, although it still may not schedule a job before its release time or use more machines than the algorithm being measured. We may not know exactly what OPT is or how it runs, but we can often determine properties about how it must behave in a particular situation, or at least determine certain minimums that it must meet. An algorithm's score for a metric is divided by OPT's score for the same metric, and this ratio is said to be the competitive ratio of the algorithm. For example, an algorithm that could always earn a flow time score no more than 10 times what OPT earns would be 10-competitive for flow time. These ratios are usually simplified using asymptotic notation.

Recently, theorists have begun studying the concept of "resource augmentation," whereby the algorithm being evaluated is allowed to use extra speed or extra machines compared to OPT. Typically, this is measured by some small augmentation factor, $\epsilon < 1$, and the algorithm is then said to be, for example, $1 + \epsilon$ speed. The advantage of studying this concept is threefold. First, there may be no constant-competitive algorithm without augmentation, but with just ϵ extra speed, we can reach a bound of $O(1/\epsilon)$, which is constant, so we now have an idea of a good algorithm. Second, not all algorithms are capable of utilizing the extra speed effectively: two algorithms with identical unaugmented performance may be differentiated by augmentation; perhaps only one of the two algorithms can reach constant-competitive with augmentation. Third, it may be realistic to operate under the assumption that a server does have extra speed available, since most servers in deployment are not constantly 100% utilizied, they have extra speed available, in effect.

1.3 Definitions and Notation

We will use the following definitions and notation throughout the paper. An input instance I consists of n jobs where job J_i is characterized by its release time, r_i, its weight or priority w_i, and its size or work p_i. The density of a job $d_i = w_i/p_i$. The jobs will be scheduled on m identical machines where no job can be assigned to more than one machine at any time. We assume that jobs can be preempted and migrated from one machine to another with no penalty. Finally, the remaining size of a job at any time t is its initial size minus the amount of work done on that job up to time t.

Given a scheduling algorithm A, we denote the completion time of job J_i as $c_i(A)$. The flow time of job J_i, denoted $F_i(A)$, is $c_i(A) - r_i$. The stretch of job J_i, denoted

$S_i(A)$, is $F_i(A)/p_i$. We then define $F^p(A,I) = \sum_i (F_i(A))^p$, $S^p(A,I) = \sum_i (S_i(A))^p$, and $WF^p(A,I) = \sum_i w_i(F_i(A))^p$. The L_p norms of flow time, stretch, and weighted flow time are then $(F^p(A,I))^{1/p}$, $(S^p(A,I))^{1/p}$, and $(WF^p(A,I))^{1/p}$, respectively. To simplify notation, we will often drop the A and I when the algorithm or input instance is clear from context.

Let $M(A(I))$ denote the cost of algorithm A when applied to input instance I when we focus on metric M where M may be the L_p norm of flow time, stretch, or weighted flow time. An online algorithm A is said to be *c-competitive* for some metric M if $M(A(I)) \leq c \cdot M(OPT(I))$ for any input instance I. An algorithm A using *resource augmentation* in the form of faster processors is said to be s-speed c-competitive if the competitive ratio c can be maintained when A's processors are s times as fast as OPT's (note that c may be defined in terms of s and is not necessarily constant). Analogously, an algorithm A is said to be s-machine c-competitive if the competitive ratio c can be maintained when A has s times as many processors as OPT.

1.4 Definition of Algorithms and Problems

Shortest Job First [SJF], Shortest Remaining Processing Time [SRPT], Highest Density First [HDF], and Highest Remaining Density First [HRDF] are greedy, preemptive, migratory algorithms that always schedule the m unfinished jobs with highest priority where the priority metric differs for each algorithm. For SJF, the priority is the initial size of the job, p_i; for SRPT, the priority is the remaining size of the job; for HDF, the priority is the density of the job, d_i; and for HRDF, the priority is the remaining density of the job, which is the weight, w_i divided by the remaining size of the job. Remaining density strictly increases as a job is processed. In all algorithms, ties are broken first in favor of a running job, then arbitrarily. Furthermore, jobs are held in a central queue until assigned to a machine, and a preempted job is returned to the queue.

We study three minimization problems: first, the unweighted L_p norm of flow time and Stretch on multiple machines, using SJF and SRPT with ϵ resource augmentation; second, the weighted L_p norm of flow time, using HDF with ϵ resource augmentation; and third, the weighted (unnormalized) flow time on a single machine, using HRDF without augmentation.

We study these problems because many currently-used scheduling algorithms are used because they perform well on server-centric criteria. While server-centric algorithms may utilize server resources well, clients may receive poor performance. By focusing on a client-centric criterion like flow time, we change the emphasis from simply "use resources efficiently" to "use resources fairly while being as efficient as possible." Adding the L_p norms to the analysis further improves the fairness to clients, since no individual job may be sacrificed for the sake of all other jobs. A multiprocessor environment is becoming increasingly common, even in home desktop systems that now feature dual- or quad-core CPUs. Finally, studying the weighted flow time case allows for a more interesting environment as might befit a corporation that may have work assigned to various tiers of priorities.

1.5 Previous Results and Related Work

In [5], Bansal and Pruhs persuasively argue that the conflicting criteria of minimizing the total wait time of an algorithm without starving individual jobs can be formalized as the weighted L_p norms of flow time. However, they consider only the uniprocessor server scheduling problem. The multiprocessor setting is important as enterprise web and file servers often are composed of multiple machines in order to handle the large volume of requests they need to serve. In this paper, we show that HDF is an effective algorithm for the multiprocessor server scheduling problem. A key step in our proof is showing that the SJF and SRPT algorithms are effective at minimizing the unweighted L_p norms of flow time and stretch on multiple machines.

Bansal and Pruhs showed that no online algorithm can be constant competitive for the problem of minimizing the L_p norms of flow time or stretch for $2 \leq p < \infty$, even in the uniprocessor environment [4]. Thus, to achieve constant competitive ratios for the problem of minimizing the weighted L_p norm of flow time, we are forced to consider the resource augmentation technique popularized by Kalyanasundaram and Pruhs [10]. Bansal and Pruhs show that several algorithms including SJF and SRPT are $(1+\epsilon)$-speed $O(1)$-competitive algorithms for minimizing the L_p norms of flow time and stretch on a single machine [4]. Chekuri *et al.* then show that simple uniprocessor scheduling algorithms such as SRPT combined with simple load balancing algorithms are also $(1+\epsilon)$-speed $O(1)$-competitive for minimizing the L_p norms of flow time and stretch on m identical machines [8]. In particular, they consider an algorithm that immediately dispatches jobs to machines and allows no migrations introduced by Avrahami and Azar [1]. Bansal and Pruhs then show that HDF and a nonclairvoyant algorithm are $(1+\epsilon)$-speed $O(1)$-competitive for the problem of minimizing weighted L_p norms of flow time on a single machine [5].

If we ignore the L_p norms and focus only on total flow time, stretch, or weighted flow time, more results are known. On a single machine, Bansal and Dhamdhere give an $O(W)$ algorithm for minimizing weighted flow time where W is the ratio of the maximum job weight divided by the minimum job weight [3]. Note that this algorithm is constant competitive if there are only a constant number of distinct job weights. Similarly, Chekuri *et al.* described an algorithm that we have named HRDF and gave a lower bound of $\sqrt{P}$, where P is the ratio of the largest job size to the smallest job size [9]. In the same paper, they proved that if P is known *a priori* to a *biased greedy* algorithm, it can achieve a $O(\log^2 P)$-competitive ratio.

With resource augmentation, Becchetti *et al.* show that HDF is $(1+\epsilon)$-speed $O(1+1/\epsilon)$-competitive in the uniprocessor setting and $(2+\epsilon)$-speed $O(1+1/\epsilon)$-competitive in the multiprocessor setting [7]. Phillips *et al.* show that an algorithm they named Preemptively-Schedule-By-Halves is a 2-speed 1-competitive algorithm on a single processor [14], and Becchetti *et al.* observe that this analysis also applies to HDF [7]. Bansal and Dhamdere then show that a nonclairvoyant algorithm is also $(1+\epsilon)$-speed $O(1+1/\epsilon)$-competitive on a single machine matching HDF's performance [3].

If we consider total flow time, it is a well-known result that SRPT is an optimal algorithm in the uniprocessor setting, and Leonardi and Raz showed that SRPT is $O(\min\{\log n/m, \log P\})$-competitive in the multiprocessor setting where P is the ratio of the maximum processing time of any job divided by the minimum processing time of any job [11]. They further show that this is optimal within constant factors for any online algorithm, randomized or deterministic. Muthukrishnan *et al.* show that SRPT is constant competitive for minimizing stretch for the multiprocessor environment [13]. Many algorithms without migration and then with immediate dispatch were developed that had essentially the same competitive ratios within constant factors [1, 2, 6, 9].

With resource augmentation, Phillips *et al.* first showed that SRPT is a $(2-1/m)$-speed 1-competitive for minimizing total flow time on m identical machines [14], and McCullough and Torng improved this result to show that SRPT is in fact an s-speed $1/s$-competitive algorithm for minimizing total flow time on m identical machines when $s \geq 2-1/m$ [12].

1.6 Our Results

We first extend the work of Bansal and Pruhs [4] and Chekuri *et al.* [8] by showing that SJF and SRPT are $(1+\epsilon)$-speed $O(1+1/\epsilon)$-competitive for minimizing the L_p norms of flow time and stretch on m identical machines. These results extend to extra machines. Furthermore, the constant hidden in the Big Oh notation is somewhat better than the constant for the immediate dispatch algorithm as analyzed by Chekuri *et al.* We then use our SJF extra speed result for flow time to extend the result of

Bansal and Pruhs [5] by showing that HDF is a $(1+\epsilon)$-speed $O(1+1/\epsilon^2)$-competitive for minimizing the weighted L_p norms of flow time on m identical machines. For the most part, our proofs are simple extensions of those found in [4, 8, 5].

Lastly, we present a lower bound of $\Omega(P)$ for HRDF on a single machine without augmentation, to show that the weighted flow time problem is significantly more difficult than the unweighted problem. This improves the bound of $\Omega(\sqrt{P})$ presented in [9].

Chapter 2

Multiprocessor algorithms with resource augmentation

In section 2.1, we will explore the popular online algorithms SJF and SRPT as augmented by faster processor or more machines, proving that each of the four algorithms is $O(\frac{1}{\epsilon})$-competitive for the L_p norms of unweighted flow time and stretch. All these proofs will use a common framework that we outline in subsections 2.1.2 and 2.1.3. Following that, section 2.2 will extend the results for SJF to show that HDF is $O\left(\frac{1}{\epsilon}\right)^2$-competitive for weighted L_p norms of flow time when augmented with faster processors.

2.1 Unweighted L_p norms of flow time and stretch

2.1.1 Notation

We first define the following notation. Let A denote any algorithm and t denote any time. Let $U(A,t)$ be the set of unfinished jobs for algorithm A at time t, and the number of such jobs. Which meaning of $U(A,t)$ will be clear from context. The total flow time of a schedule can be computed as $\int U(A,t)\,dt$. For any job J_i in $U(A,t)$, let $\text{Age}_i(t) = t - r_i$ and $\text{SAge}_i(t) = \text{Age}_i(t)/p_i$; informally this is a "stretched age." Let $\text{Age}^p(A,t)$ denote the sum over all jobs $J_i \in U(A,t)$ of $\text{Age}_i(t)^{p-1}$, and let $\text{SAge}^p(A,t)$ denote the sum over all jobs $J_i \in U(A,t)$ of $\text{SAge}_i(t)^{p-1}$. Bansal and Pruhs observed that $F^p(A) = \int \text{Age}^p(A,t)\,dt$ and that $S^p(A) = \int \text{SAge}^p(A,t)\,dt$ [4].

We use $V_{\leq k}(t)$ to denote the sum of remaining sizes, or Volume, of all unfinished jobs of initial size at most k for SJF or SRPT at time t released prior to time t. We similarly define $V^*_{\leq k}(t)$ for OPT. We use $P_{\leq k}(t)$, to denote the sum of initial sizes, or Processing Times, of all unfinished jobs of initial size at most k for SJF or SRPT at time t released prior to time t. We use $P_{\leq k}(t_1, t_2)$ to denote the sum of initial sizes of all jobs of initial size at most k released during interval $[t_1, t_2)$.

2.1.2 Proof Structure

Our strategy for proving competitive ratios for SJF and SRPT will follow that of [8] which is a modification of the uniprocessor strategy of [4]. We focus on the L_p norms of flow time as the proof for stretch is essentially identical.

Given that $F^p(A) = \int \mathrm{Age}^p(A,t)\, dt$, we would prove our result if we can bound $\mathrm{Age}^p(A,t)$ relative to $\mathrm{Age}^p(OPT,t)$ for all times t. Unfortunately, as observed in [8], this is not possible as online algorithms can "fall behind" OPT so that there can be times where $\mathrm{Age}^p(A,t) > 0$ while $\mathrm{Age}^p(OPT,t) = 0$. To compensate for this problem, at any time t, we restrict our attention to unfinished jobs that are sufficiently old as was done in [8].

More formally, we define $U^\alpha(A,t) = \{J_i \in U(A,t) \mid \mathrm{Age}_i(t) \geq \alpha p_i\}$ where $\alpha > 0$ is a constant to be defined later. α serves as a delaying factor; a job that is not at least α times as old as its processing time is uninteresting to the schedule, since it is not yet old enough to contribute significantly to the flow time. We then define $\mathrm{Age}^p(A,t,\alpha)$, the p^{th} norm of Age with respect to α, to be the sum over all jobs $J_i \in U^\alpha(A,t)$ of $\mathrm{Age}_i(t)^{p-1}$ and the p^{th} norm of flow time with respect to α, $F^p(A,\alpha) = \int Age^p(A,t,\alpha)\, dt$. Finally, we define $DIF(A) = F^p(A) - F^p(A,\alpha)$, or the difference between the actual flow time of A and the flow time with respect to α. We can then bound $F^p(A)$ relative to $F^p(OPT)$ if we can individually bound both $F^p(A,\alpha)$ and $DIF(A)$ relative to $F^p(OPT)$.

For $DIF(A)$, we first observe that $F^p(OPT) \geq (\sum p_i^p)^{1/p}$ as each job j_i requires at least p_i time to finish. On the other hand, $DIF(A) \leq (\sum(\alpha p_i)^p)^{1/p}$ as $F^p(A,\alpha)$ accounts for the entire time that each job is in the system except for at most the first αp_i time units. Combining these inequalities and simplifying, we observe that $DIF(A) \leq \alpha F^p(OPT)$.

2.1.3 An Association Scheme for Bounding $F^p(A, \alpha)$

For $F^p(A, \alpha)$, we observe that for any time t, jobs that are in both $U^\alpha(A, t)$ and $U(OPT, t)$ contribute equally to $\text{Age}^p(A, t, \alpha)$ and $\text{Age}^p(OPT, t)$. Thus, the reason that $\text{Age}^p(A, t, \alpha)$ may greatly exceed $\text{Age}^p(OPT, t)$ are the jobs that are in $U^\alpha(A, t)$ but *not in* $U(OPT, t)$. We define the *difference set*, $D(t)$ as $U^\alpha(A, t) - U(OPT, t)$ where the specific algorithm A will always be clear from context. $D(t)$ represents the old jobs that A has left to finish at time t but which OPT has already completed. To simplify notation, we will often use D to represent $D(t)$. We then define the p^{th} norm of the Age of the difference set, $\text{Age}^p(D, t)$ to be the sum over all jobs $J_i \in D(t)$ of $\text{Age}_i(t)^{p-1}$ and observe that $\text{Age}^p(A, t, \alpha) \leq \text{Age}^p(OPT, t) + \text{Age}^p(D, t)$.

We now define an association scheme S_β similar to that of [4] to show that $\text{Age}^p(D, t)$ is bounded relative to $\text{Age}^p(OPT, t)$. Given $0 < \beta < 1$, for each job J_j in $U(OPT, t)$, we define a set of jobs $S_\beta(j)$ with the following properties:

1. For all $J_i \in S_\beta(j)$, $\text{Age}_i(t) \leq 1/\beta \, \text{Age}_j(t)$.

2. For all $J_i \in S_\beta(j)$, $p_j \leq p_i$. With this constraint, the argument easily extends to L_p norms of stretch as well, and we will not cover those details here.

3. $|S_\beta(j)| \leq 1/\beta$.

4. $D(t) = \bigcup_{J_j \in U(OPT, t)} S_\beta(j)$

This association requires that every job in the difference set is associated with at least one job that OPT has not finished (condition 4), and since not more than $1/\beta$ jobs from D can be associated with the same $j \in U(OPT, t)$ (condition 3), the existence of such a scheme will bound the number of jobs in D. Further, since the jobs in D are within a $1/\beta$ factor of the age the jobs they are associated with in $U(OPT, t)$, we can bound the total Age of D, as follows: If we can create such an association scheme S_β for all times t, then $\text{Age}^p(D, t) \leq (1/\beta)^p \, \text{Age}^p(OPT, t)$ and thus $F^p(A,\alpha) \leq (1 + 1/\beta)F^p(OPT)$. Combining this with the earlier bound for $DIF(A)$, we get that $F^p(A) \leq (1 + \alpha + 1/\beta)F^p(OPT)$.

We now prove that an association scheme S_β exists for all times t and β sufficiently large relative to ϵ. For any time t, index the jobs in $D(t)$ in increasing order of size p_i. Given time t and $i \leq |D(t)|$, define $t(i) = t - \beta(t - r_i)$. This time index, $t(i)$, allows us to find jobs that have an Age at least β times the age of the corresponding job j_i. To prove that an association scheme S_β exists, it sufficient to prove that we can always allocate to $J_i \in D(t)$ a βp_i amount of work from $V^*_{\leq p_i}(t)$ from jobs with release time at most $t(i)$ that was not already allocated to a lower indexed job in $D(t)$.

We use $AV(t,i)$, or *Available Volume*, to denote the amount of work from $V^*(t)$ that can be assigned to the first i jobs in $D(t)$.

Lemma 1. *For any time t and any $i \leq |D(t)|$,*

$$AV(t,i) \geq V^*_{\leq p_i}(r_i) + P_{\leq p_i}(r_i, t(i)) - m(t - r_i).$$

Proof. OPT's unfinished work at time t for jobs of length at most p_i released before $t(i)$ is at least: OPT's unfinished work of jobs of this length as of r_i plus the total work to be done for jobs of this length arriving in $[r_i, t(i))$ minus the maximal work done by OPT during $[r_i, t)$. □

Define $W(t,i)$ to be the amount of work that must be allocated to the first i jobs in $D(t)$.

Lemma 2. *For any time t and any $i \leq |D(t)|$,*

$$W(t,i) \leq \beta(m(t - r_i) + P_{\leq p_i}(r_i))$$

Proof. Consider the first i jobs in $D(t)$. Some of these jobs are released at time r_i or later. By definition of $D(t)$, OPT must finish these jobs by time t, so their total size can be at most $m(t - r_i)$. The other jobs must be released before time r_i and are unfinished by A at time r_i. Thus we can bound their processing times by $P_{\leq p_i}(r_i)$ and the result follows. $\square$

It suffices to show that the amount of work required for the allocation is no greater than the amount of work available. Our goal is to show that for any time t, any $i \leq |D(t)|$ and an appropriate value of β that

$$W(i,t) \leq \beta(m(t-r_i) + P_{\leq p_i}(r_i)) \leq V^*_{\leq p_i}(r_i) + P_{\leq p_i}(r_i, t(i)) - m(t - r_i) \leq AV(t,i)$$

We restate our goal as follows.

Theorem 1. *Consider the online algorithms SJF and SRPT augmented with either $(1+\epsilon)$-speed processors or $(1+\epsilon)m$ 1-speed processors. For any time t and any $1 \leq i \leq |D(t)|$, there exist values of β and α such that:*

$$(1+\beta)m(t-r_i) + \beta P_{\leq p_i}(r_i) \leq V^*_{\leq p_i}(r_i) + P_{\leq p_i}(r_i, t(i)) \qquad (2.1.1)$$

Corollary 1. *SJF and SRPT are $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive and $(1+\epsilon)m$-machine $O(1/\epsilon)$-competitive.*

Proof. Our proofs in the next sections show that the following values for β and α make Theorem 1 true. For SJF or SRPT with $1+\epsilon$-speed processors or $(1+\epsilon)m$ machines, $\beta = \epsilon/(6+6\epsilon)$. α varies as follows: for SJF augmented with $(1+\epsilon)$-speed processors, $\alpha = 4/\epsilon + 4$; for SRPT with $(1+\epsilon)$-speed, $\alpha = 6/\epsilon + 6$; for SJF with $(1+\epsilon)m$ machines, $\alpha = 4/\epsilon + 8 + 6\epsilon$; and for SRPT with $(1+\epsilon)m$ machines, $\alpha = 6/\epsilon + 12 + 6\epsilon$. $\square$

We prove Theorem 1 in the next sections, first considering SJF with faster machines.

2.1.4 SJF

We seek to prove Theorem 1 for SJF and faster machines. The first step in the proof is to take advantage of our augmentation by deriving a bound on the volume of remaining work for OPT at any time u, $V^*_k(u)$. This derivation requires new arguments than those from [4, 8].

Lemma 3. *When SJF is given $(1+\epsilon)$-speed processors, for any time u and any job size k,*

$$V^*_{\leq k}(u) \geq \frac{\epsilon}{1+\epsilon} P_{\leq k}(u) + \frac{1}{1+\epsilon} V_{\leq k}(u) - mk \tag{2.1.2}$$

Proof. We first focus on SJF's work. Define time $u' < u$ as the most recent time (possibly 0) such that for some positive interval of time just prior to u', SJF had an idle machine or was running a job with initial size $> k$. By definition of u', SJF has at most $m-1$ unfinished jobs of initial size at most k at time u' that were released prior to time u'. Thus, $V_{\leq k}(u') \leq (m-1)k < mk$.

Now, by the definition of u', every machine must remain busy until u working on jobs with initial size at most k. Thus, SJF does at least $(1+\epsilon)m$ work on jobs of initial size at most k during the interval $[u', u)$. Finally, the amount of work of initial size at most k that arrives during the interval $[u', u)$ is $P_{\leq k}(u', u)$. This leads to the following observation:

Fact 1. *At any time u, for u' relative to u, and any job size k, the amount of unfinished work left for SJF is bounded:*

$$V_{\leq k}(u) \leq mk + P_{\leq k}(u', u) - (1+\epsilon)m(u-u') \tag{2.1.3}$$

We now consider OPT. OPT receives $P_{\leq k}(u', u)$ work from jobs of size at most k in interval $[u', u)$ and can do at most $m(u-u')$ work during interval that interval, which gives the following:

Fact 2. *At any time u, for u' relative to u, and any job size k, the amount of unfinished work left for OPT is bounded:*

$$V^*_{\leq k}(u) \geq P_{\leq k}(u', u) - m(u - u') \tag{2.1.4}$$

Rewriting (2.1.3) as $-m(u-u') > \frac{1}{1+\epsilon}(V_{\leq k}(u) - mk - P_{\leq k}(u', u))$ and substituting into (2.1.4) yields $V^*_{\leq k}(u) \geq \frac{\epsilon}{1+\epsilon}P_{\leq k}(u', u) + \frac{1}{1+\epsilon}V_{\leq k}(u) - \frac{1}{1+\epsilon}mk$. To complete the proof, we then observe that $P_{\leq k}(u', u) \geq P_{\leq k}(u) - mk$, as at most mk of the unfinished work for SJF at time u could have come prior to time u', by its definition. □

From here, we use (2.1.2) to substitute into (2.1.1), changing variables ($u = r_i$ and $k = p_i$). We then replace $P_{\leq p_i}(r_i)$ with $V_{\leq p_i}(r_i)$ which is obviously smaller. Combining like terms and restricting $\beta \leq \frac{\epsilon}{1+\epsilon}$ gives the following goal.

$$(1 - \beta)V_{\leq p_i}(r_i) + P_{\leq p_i}(r_i, t(i)) \geq (1 + \beta)m(t - r_i) + mp_i \tag{2.1.5}$$

We now derive a lower bound on $P_{\leq p_i}(r_i, t(i)) + V_{\leq p_i}(r_i)$ by deriving a lower bound on the work done by SJF during $[r_i, t(i))$ on jobs of length at most p_i.

Lemma 4. *For any job $J_i \in D(t)$, we derive the following bound:*

$$V_{\leq p_i}(r_i) + P_{\leq p_i}(r_i, t(i)) \geq (1 + \epsilon)m(t(i) - r_i - p_i/(1 + \epsilon)) \tag{2.1.6}$$

Proof. We first observe that if SJF works on job J_i for $p_i/(1+\epsilon)$ time in this interval, it would finish job J_i by time $t(i)$ and thus J_i would not be in $D(t)$. Thus SJF must be working on other jobs of size at most p_i for at least $t(i) - r_i - \frac{p_i}{1+\epsilon}$ time and the result follows. □

Using (2.1.6) to substitute into (2.1.5) (noting that the $(1-\beta)$ coefficient was also applied to the P term before substituting and that $t(i) - r_i = (1-\beta)(t - r_i))$, we get the following goal:

$$(1-\beta)\left((1+\epsilon)(1-\beta)m(t-r_i) - mp_i\right) \geq (1+\beta)m(t-r_i) + mp_i \qquad (2.1.7)$$

To complete the proof, we need to select values for β and α. Intuitively, returning to the association scheme S_β, we are trying to associate unfinished jobs from OPT with jobs in $D(t)$. Increasing α helps by restricting the membership of $D(t)$. Decreasing β helps as each job in OPT can be associated with more jobs in $D(t)$. We now derive specific values of β and α that imply (2.1.7).

Since we have previously restricted $\beta \leq \frac{\epsilon}{1+\epsilon}$, we will start by selecting $\beta = \frac{\epsilon}{1+\epsilon}$, where c is a constant ≥ 1 to be determined later. Substituting this into the above, we get the following sequence of goals.

$$\begin{aligned}
\frac{(c+c\epsilon-\epsilon)^2}{c^2(1+\epsilon)}(t-r_i) - \frac{c+c\epsilon+\epsilon}{c(1+\epsilon)}(t-r_i) &\geq (2-\beta)p_i \\
\left(\frac{c^2\epsilon - 3c\epsilon + c^2\epsilon^2 - 2c\epsilon^2 + \epsilon^2}{c^2}\right)(t-r_i) &\geq 2(1+\epsilon)p_i \\
\left(\frac{c^2\epsilon - 3c\epsilon}{c^2}\right)(t-r_i) &\geq 2(1+\epsilon)p_i \\
(t-r_i) &\geq \left(\frac{2c+2c\epsilon}{(c-3)\epsilon}\right)p_i
\end{aligned}$$

Note that the 3rd goal implies the second goal when $c \geq 2$ as $c^2\epsilon^2 - 2c\epsilon^2 + \epsilon^2 \geq 0$ when $c \geq 2$. We can satisfy the last goal as long as we choose $\alpha = (2c + 2c\epsilon)/((c-3)\epsilon)$. We now choose a value of c that minimizes $(1 + 1/\beta + \alpha)$. For SJF, selecting $c = 6$ yields $\beta = \frac{\epsilon}{6(1+\epsilon)}$ and $\alpha = \frac{4}{\epsilon} + 4$ giving a total competitive ratio of $\frac{10}{\epsilon} + 11 = O\left(\frac{1}{\epsilon}\right)$.

We now compare our bounds for SJF with bounds for IMD. In [8], β was chosen to be $\frac{\epsilon}{4(1+\epsilon)}$, giving an α of $\frac{96}{\epsilon}$ for IMD with extra speed. A better bound can be achieved by selecting $\beta = \frac{\epsilon}{27(1+\epsilon)}$ to yield a competitive ratio of $\frac{54}{\epsilon} + \frac{110}{3}$. While neither bound is necessarily tight, this comparison gives us some insight into the power of migration for this problem.

2.1.5 SRPT

We now prove Theorem 1 for SRPT and faster machines. This proof is very similar to that for SJF and faster machines with only a few differences. We first observe that we can prove the same result as Lemma 3 for SRPT, but the proof is different because of the differences in the algorithms.

Lemma 5. *When SRPT is given* $(1+\epsilon)$*-speed processors, for any time* u *and job size* k*,*

$$V^*_{\le k}(u) \ge \frac{\epsilon}{1+\epsilon} P_{\le k}(u) + \frac{1}{1+\epsilon} V_{\le k}(u) - mk \qquad (2.1.8)$$

Proof. We first focus on SRPT and define time $u' < u$ similar to before: the most recent time (possibly 0) such that for some positive interval of time ending at u', SRPT had an idle machine or was running a job with *remaining* size $> k$. From this, there can be at most m unfinished jobs released prior to time u' with remaining size at most k at time u'. For the sake of argument, say m_1 of these jobs have $p_i > k$ while the other $m_2 \le m - m_1$ jobs have $p_i \le k$, making the total work left in the system for short jobs $V_{\le k}(u') \le m_2 k$.

Now, by the definition of u', every machine in the interval $[u', u)$ must be busy processing jobs of remaining size at most k. In particular, any jobs that arrive in the interval $[u', u)$ that are processed must have initial size at most k. Thus, the only work done during the interval $[u', u)$ on jobs with initial size larger than k is any work done on the m_1 unfinished jobs at time u'. SRPT can do at most $m_1 k$ work on these jobs during $[u', u)$. Thus, SRPT does at least $(1+\epsilon)m - m_1 k$ work on jobs of initial size at most k during $[u', u)$. The amount of work for jobs of initial size at most k

left at time u can be no more than the work left at u', plus the work for new jobs arriving during this interval, minus the work done during the same interval.

Fact 3. *At any time u, for u' relative to u, and any job size k, the amount of unfinished work left for SRPT is bounded:*

$$\begin{aligned} V_{\leq k}(u) &\leq m_2 k + P_{\leq k}(u', u) - ((1+\epsilon)m(u-u') - m_1 k) \\ &\leq mk + P_{\leq k}(u', u) - (1+\epsilon)m(u-u') \end{aligned}$$

The amount of work OPT has left is unchanged, since its solution is independent of the algorithm being tested. We use the above inequality, Fact 2, and the same observation that $P_{\leq k}(u', u) \geq P_{\leq k}(u) - mk$ to arrive at (2.1.8). □

As before, we bound the work accomplished by SRPT during $[r_i, t(i))$ for jobs of initial size p_i.

Lemma 6. *For any job $J_i \in D(t)$, we derive the following bound:*

$$V_{\leq k}(r_i) + P_{\leq k}(r_i, t(i)) \leq (1+\epsilon)m\left(t(i) - r_i - \frac{p_i}{1+\epsilon}\right) - mp_i$$

Proof. The maximal amount of work done is $(1+\epsilon)m(t(i) - r_i)$. From this amount, deduct the potential work applied to large jobs with short remaining times and the maximum potential idle time. □

Finishing the proof the same way as for SJF, we select $\beta = \frac{\epsilon}{6(1+\epsilon)}$ and $\alpha = \frac{6}{\epsilon} + 6$, for a ratio of $\frac{12}{\epsilon} + 13 = O\left(\frac{1}{\epsilon}\right)$. Note again that this bound is significantly better than the bound for IMD.

2.1.6 Extra machines

We prove Theorem 1 for SJF and SRPT with extra machines. In this scenario, SJF-M or SRPT-M uses $(1+\epsilon)m$ 1-speed machines while OPT uses m 1-speed machines.

The proof structure is the same as before. Two facts change: the work SJF-M and SRPT-M have left at any time u is increased slightly, and the minimum work either algorithm does in any interval from $[r_i, t(i))$ is slightly diminished. Modified versions of Facts 1 and 3 (combined here) and Lemmas 4 and 6 are presented below.

Fact 4. *For any time u, with regard to jobs of initial size at most k, when SJF-M or SRPT-M has $(1+\epsilon)m$ 1-speed processors:*

$$V_{\leq k}(u) \leq (1+\epsilon)mk + P_{\leq k}(u', u) - (1+\epsilon)m(u-u')$$

Proofsketch: Since we have $(1+\epsilon)m$ machines instead of m machines, when we define u' as before, we get $V_{\leq k}(u') \leq (1+\epsilon)mk$ instead of $V_{\leq k}(u') \leq mk$. The rest follows as before.

Lemma 7. *Consider SJF-M. For any time t and any job $J_i \in D(t)$, we derive the following bound:*

$$V_{\leq p_i}(r_i) + P_{\leq p_i}(r_i, t(i)) \leq (1+\epsilon)m(t(i) - r_i - p_i)$$

Proof. SJF-M could not have worked on job J_i during $[r_i, t(i))$ for p_i time units or else J_i would be completed at time t and thus not part of $D(t)$. Thus, SJF-M must devote all $(1+\epsilon)m$ machines for at least $t(i) - r_i - p_i$ time units to other jobs of size at most p_i during the interval $[r_i, t(i))$. □

Lemma 8. *Consider SRPT-M. For any time t and any job $J_i \in D(t)$, we derive the following bound:*

$$V_{\leq k}(r_i) + P_{\leq k}(r_i, t(i)) \leq (1+\epsilon)m(t(i) - r_i - 2p_i)$$

Proof. SRPT-M could not have worked on job J_i during $[r_i, t(i))$ for p_i time units or else J_i would be completed at time t and thus not part of $D(t)$. Thus, SRPT-M must devote all $(1+\epsilon)m$ machines for at least $t(i) - r_i - p_i$ time units to other jobs of remaining size at most p_i during the interval $[r_i, t(i))$. Of these jobs, at most $(1+\epsilon)mp_i$ of this time could be devoted to jobs of initial size $> p_i$ and the result follows. □

Putting everything back together, for SJF-M, we select $\beta = \frac{\epsilon}{6(1+\epsilon)}$ and $\alpha = \frac{4}{\epsilon} + 8 + 6\epsilon$, so the total ratio is $\frac{10}{\epsilon} + 15 + 6\epsilon = O\left(\frac{1}{\epsilon}\right)$. For SRPT-M, we select $\beta = \frac{\epsilon}{6(1+\epsilon)}$ and $\alpha = \frac{6}{\epsilon} + 12 + 6\epsilon$, giving a ratio of $\frac{12}{\epsilon} + 19 + 6\epsilon = O\left(\frac{1}{\epsilon}\right)$. These compare to the best values for IMD with extra machines in [8] of $\beta = \frac{\epsilon}{27(1+\epsilon)}$ and $\alpha = \frac{27}{\epsilon} + \frac{80}{3}$, for a total ratio of $\frac{54}{\epsilon} + \frac{164}{3}$.

2.2 Weighted L_p norms of flow time

We now consider the weighted L_p norms of flow time. Recall that each job j_i is now said to have a positive weight w_i, where a higher weight represents a higher priority, and the weighted flow time of a job is $w_i f_i$, where f_i is the job's unweighted flow time. We follow the proof technique used in [5] for a single machine, extending it now for multiple machines. We adopt their notations of $OPT(M, \mathcal{I}, S)$ as meaning the value of the optimal solution for metric M on input instance $\mathcal{I}$ given S-speed processors, and $M(A, \mathcal{I}, S)$ as meaning the worst-case solution for metric M under algorithm A on input instance $\mathcal{I}$ with S-speed processors.

Given an input instance $\mathcal{I}$, we define $\mathcal{I}'$ as follows: for each $J_i \in \mathcal{I}$, create w_i identical jobs in $\mathcal{I}'$, each of size p_i/w_i, weight 1, and release time r_i. Denote these as $J'_{i_1}, J'_{i_2}, \ldots J'_{iw_i}$ and let $X_i = \{J'_{i_1} \ldots J'_{iw_i}\}$.

Lemma 9. *Given an input instance $\mathcal{I}$ and related instance $\mathcal{I}'$ as defined above,*

$$OPT(F^p, \mathcal{I}', 1) \leq OPT(WF^p, \mathcal{I}, 1)$$

Proof. Let S be a schedule which minimizes the L_p norm of weighted flow time for

$\mathcal{I}$. Given S we schedule $\mathcal{I}'$ as follows: for any time t, for a given machine a' under $OPT(\mathcal{I}')$, work on a job in X_i if and only if J_i is being processed on the corresponding machine a under S at t.

Thus all jobs in X_i are finished when J_i finishes, so no job in X_i has a higher unweighted flow time than J_i, which by definition is $w_i f_i^p$. The w_i jobs in X_i each contribute a maximum of f_i^p to flow time, for a maximum total of $w_i f_i^p$, and OPT can do no worse. □

From Theorem 1, we know that SJF is $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive for L_p norms of unweighted flow time on multiple machines. Thus,

Fact 5.

$$F^p(SFJ, \mathcal{I}', 1+\epsilon) = O\left(\frac{1}{\epsilon}\right) OPT(F^p, \mathcal{I}', 1)$$

Now we relate HDF on $\mathcal{I}$ with $(1+\epsilon)$ speed to SJF with 1 speed.

Lemma 10. *For every job $J_i \in \mathcal{I}$ and time t, if J_i is alive under HDF with $(1+\epsilon)$-speed, at least $\frac{w_i\epsilon}{1+\epsilon}$ jobs in $X_i \in \mathcal{I}'$ are alive at t under SJF with 1 speed.*

Proof. Suppose t is the earliest time when J_i is alive under HDF and there are less than $\frac{w_i\epsilon}{1+\epsilon}$ jobs from X_i alive under SJF. Since J_i is alive under HDF with $(1+\epsilon)$-speed processors, HDF has spent less than $\frac{p_i}{1+\epsilon}$ time on J_i, whereas SJF has spent more than that amount of time on jobs in X_i.

So there must exist some time $t' \in [r_i, t)$ when HDF was not running J_i on any machine and SJF was processing at least one job in X_i on some machine. Therefore, there exist m jobs $J_{j_1}...J_{j_m}$ with strictly higher density than J_i being processed at t'. Thus, the $X_{j_1}...X_{j_m}$ jobs corresponding to J_j would all have smaller sizes than those in X_i. Since SJF works on X_i at time t', it must have finished all the jobs in $X_{j_1}...X_{j_m}$ by t', and since $J_{j_1}...J_{j_m}$ are alive under HDF at t', this contradicts the assumption of the minimality of t. □

Corollary 2. *HDF is a $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm for minimizing the weighted L_p norms of flow time on m identical machines.*

Proof. We get $WF^p(HDF, \mathcal{I}, 1+\epsilon) \leq (1+1/\epsilon)^p F^p(SJF, \mathcal{I}', 1)$ from Lemma 10. From Fact 5, we see that $WF^p(HDF, \mathcal{I}, 1+\epsilon) = O\left(\frac{1}{\epsilon^2}\right)^p OPT(WF^p, \mathcal{I}, 1)$, and the result follows. □

2.3 Conclusions

Our results indicate that SJF, SRPT and HDF are reasonable algorithms for multiprocessor server scheduling applications given that they are $(1+\epsilon)$-speed $O(1)$-competitive algorithms for the scheduling problems considered. Furthermore, combining our results with those from [8] provides some insight into the power of migration for the unweighted L_p norms of flow time and stretch. However, several open questions remain. For example, can an algorithm that does not use migrations (and with immediate dispatch) be effective for the weighted L_p norms of flow time? Also, can new proof techniques prove a tighter bound on the competitive ratio for HDF? In particular, is HDF $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive for minimizing the weighted L_p norms of flow time?

Chapter 3

Single machine without resource augmentation

3.1 Introduction

The problem of optimizing weighted flow time is in many ways related to the problem of optimizing unweighted flow time. In the latter problem, SRPT and SJF are similar algorithms, but SRPT is optimal for minizing unweighted flow on a single machine and SJF is not. Our goal was to prove that a natural analogue of SRPT, namely HRDF, or Highest Remaining Density First, is superior to a natural analogue of SJF, namely, HDF, or Highest Density First for minimizing weighted flow on a single machine.

As a reminder, HRDF is a greedy algorithm that always runs the job with the highest remaining density (weight divided by remaining processing time) that has been released at or before the current time step. Ties among jobs are broken arbitrarily. In contrast, HDF does not consider the remaining time of a job, and always runs the job with the highest density.

We define W as the ratio of the highest weight among all jobs to the lowest weight among all jobs; likewise we define P as the ratio of lengths between the longest job and the shortest job. It is known that HDF is $\Omega(P)$-competitive and HRDF is $\Omega(\sqrt{P})$-competitive for minimizing weighted flow on a single machine [9], so we sought to prove that HRDF is actually $O(\sqrt{P})$-competitive, but found that HRDF is actually $\Omega(P)$-competitive instead.

3.2 Overview

In broad terms, HRDF can be forced to repeatedly make bad decisions about which job to run when given a choice. This proof has similarities to the proof that SRPT can be arbitrarily bad for flow time on multiple machines without augmentation. The idea will be to release 2 jobs with nearly-identical densities, one "short" job and one "long" job with twice the length and a slightly higher density than the short one. HRDF will prioritize the long job, but OPT can finish the short job and be ready to start another job when HRDF is only half done with the long job. This new job can have a density that is nearly double the initial density of the initial "long" job and HRDF will still keep running the initial job. As soon as OPT finishes this new job, another job can be released with even higher density, and so on. If all the jobs except the initial "short" job are the same length, HRDF will always be about half a job behind OPT. Eventually, a stream of extremely dense jobs can be released when OPT finishes a high-density job, leaving HRDF to carry a very high weight job while OPT only carries the initial low weight long job.

3.3 Proof

At time 0, release two jobs, j_0 with length $P/2$ and weight 1 and j_1 with length P and weight $2+\epsilon$, where ϵ is an arbitrarily small positive constant. Since HRDF will work on j_1, at time $P/2$, j_1 will have a remaining density of $(4+2\epsilon)/P$. At this point then, another job (j_2) of size P and weight $4+\epsilon$ (density $(4+\epsilon)/P$) can be released. OPT, which ran j_0 to start, is immediately ready to run j_2. HRDF continues the higher remaining density job, j_1, finishing at time P and begins j_2. At time $3P/2$, OPT finishes j_2, so we can release another job. Again we make this job denser than before, since HRDF is half done with j_2, this new job may be $8+\epsilon/P$ dense. This pattern can repeat as long as there are higher-weight jobs available, i.e., until a job j_Y with weight $W = 2^Y + \epsilon$ is released. The release schedule for the first phase is summarized in Table 3.1 on the next page.

At time $t = PY - P/2$, OPT will only have job j_1 with weight 2 unfinished (we drop the ϵ terms here as insignificant). Meanwhile, HRDF will have j_0 with weight

job	time	size	weight
j_0	0	$P/2$	1
j_1	0	P	$2+\epsilon$
j_2	$P/2$	P	$4+\epsilon$
j_i	$P/2+P(i-2)$	P	$2^i+\epsilon$
j_Y	$P/2+P(Y-2)$	P	$W=2^Y+\epsilon$

Table 3.1: Summary of release schedule for phase 1

1 and job j_Y with weight W unfinished. Job j_0 will be unstarted while j_Y will be half-finished.

Starting at time t, a stream of very dense jobs with weight $2W/P+\epsilon$ and length 1 are released at unit time steps. Each job is immediately run by both algorithms. This stream continues for X time units, where X is a positive constant.

With an appropriate choice of Y and X, it is easy to see that the dominant term for both algorithms will be their flow time from t to $t+X$. During this period, HRDF always has W weight in unfinished jobs, while OPT always has $2W/P$ weight in the same. Thus it follows that HRDF is $\Omega(P)$-competitive for weighted flow.

3.4 Conclusions

Since we have shown HRDF to be at best $\Omega(P)$-competitive, a ratio matched by a simple greedy algorithm that always runs the heaviest job without regard to size [9], we begin to see that it is significantly harder to come up with a simple greedy algorithm when there are multiple criteria influencing the evaluation metric. Finding a good upper bound for HRDF, possibly with resource augmentation, remains an open problem. As SJF and SRPT are both $O\left(\frac{1}{\epsilon}\right)$-competitive for unweighted flow time on a single machine, and HDF matches that for weighted flow time on one machine, might HRDF also be proved to be $O\left(\frac{1}{\epsilon}\right)$-competitive on a single machine? Without augmentation, it is easy to show that SRPT outperforms SJF, but is the same true for HRDF with respect to HDF?

Bibliography

[1] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *Proc. 15th Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 11–18. ACM, 2003.

[2] Baruch Awerbuch, Yossi Azar, Stefano Leonardi, and Oded Regev. Minimizing the flow time without migration. In *Proc. 31st Symp. Theory of Computing (STOC)*, pages 198–205. ACM, 1999.

[3] Nikhil Bansal and K. Dhamdhere. Minimizing weighted flow time. In *Proc. 14th Symp. on Discrete Algorithms (SODA)*, pages 508–516. ACM/SIAM, 2003.

[4] Nikhil Bansal and Kirk Pruhs. Server scheduling in the L_p norm: A rising tide lifts all boats. In *Proc. 35th Symp. Theory of Computing (STOC)*, pages 242–250. ACM, 2003.

[5] Nikhil Bansal and Kirk Pruhs. Server scheduling in the weighted ℓ_p norm. In *Proc. 6th Theoretical Informatics, Latin American Symposium (LATIN)*, pages 434–443, 2004.

[6] L. Becchetti, S. Leonardi, and S. Muthukrishnan. Scheduling to minimize average stretch without migration. In *Proc. 11th Symp. on Discrete Algorithms (SODA)*, pages 548–557. ACM/SIAM, 2000.

[7] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. In *RANDOM-APPROX*, volume 2129 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2001.

[8] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multiprocessor scheduling to minimize flow time with ϵ resource augmentation. In *Proc. 36th Symp. Theory of Computing (STOC)*, pages 363–372. ACM, 2004.

[9] Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proc. 33rd Symp. Theory of Computing (STOC)*, pages 84–93. ACM, 2001.

[10] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47:214–221, 2000.

[11] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *Proc. 29th Symp. Theory of Computing (STOC)*, pages 110–119. ACM, 1997.

[12] Jason McCullough and Eric Torng. SRPT optimally uses faster machines to minimize flow time. In *Proc. 15th Symp. on Discrete Algorithms (SODA)*, pages 343–351. ACM/SIAM, 2004.

[13] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. E. Gehrke. Online scheduling to minimize avarage strech. In *Proc. 40th Symp. Foundations of Computer Science (FOCS)*, pages 433–443. IEEE, 1999.

[14] Cynthia Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, pages 163–200, 2002.