Multi-processor Scheduling to Minimize Flow Time with ϵ Resource Augmentation

Chandra Chekuri
Lucent Bell Labs
600 Mountain Avenue
Murray Hill, NJ 07974.
chekuri@research.bell-labs.com

Ashish Goel *
Dept. of Mgmt. Sci. & Engg.
Stanford University
Stanford, CA 94305
ashishg@stanford.edu

Sanjeev Khanna†
Dept. of Comp. & Inf. Sci.
University of Pennsylvania
Philadelphia, PA 19104
sanjeev@cis.upenn.edu

Amit Kumar‡
Dept. of Comp. Sci. & Engg.
Indian Inst. of Technology
New Delhi 110016
amitk@cse.iitd.ernet.in

ABSTRACT

We investigate the problem of online scheduling of jobs to minimize flow time and stretch on m identical machines. We consider the case where the algorithm is given either $(1+\epsilon)m$ machines or m machines of speed $(1+\epsilon)$, for arbitrarily small $\epsilon > 0$. We show that simple randomized and deterministic load balancing algorithms, coupled with simple single machine scheduling strategies such as SRPT (shortest remaining processing time) and SJF (shortest job first), are $O(\text{poly}(1/\epsilon))$-competitive for both flow time and stretch. These are the first results which prove constant factor competitive ratios for flow time or stretch with arbitrarily small resource augmentation. Both the randomized and the deterministic load balancing algorithms are non-migratory and do immediate dispatch of jobs.

The randomized algorithm just allocates each incoming job to a random machine. Hence this algorithm is non-clairvoyant, and coupled with SETF (shortest elapsed time first), yields the first non-clairvoyant algorithm which is constant competitive for minimizing flow time with arbitrarily small resource augmentation.

The deterministic algorithm that we analyze is due to Avrahami and Azar. For this algorithm, we show $O(1/\epsilon)$-competitiveness for total flow time and stretch, and also for their L_p norms, for any fixed $p \geq 1$.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.1 [**Discrete Mathematics**]: Combinatorics.

General Terms

Algorithms, Theory.

Keywords

Flow Time, Stretch, Load Balancing, Multi-processor Scheduling, Online Algorithms, Resource Augmentation.

1. INTRODUCTION

We address the problem of minimizing the average response time of jobs in the multi-processor setting. We are given m identical machines and n jobs. Job i has an arrival time of r_i and a processing requirement of p_i. Jobs queue up at the machines as they wait to be serviced, and we are interested in minimizing the overall time jobs spend in the system (referred to as flow time or response time). This simple scenario is of great interest in both theory and practice. Applications range from scheduling of jobs by operating systems, serving HTTP requests at web servers, parallel computing, and many others. We work in the model where jobs can be *preempted* and rescheduled: this is necessary in almost all cases where resources are shared by many independent competing requests. Both *clairvoyant* (the processing time of a job is known when it arrives) and *non-clairvoyant* (the processing time of a job in known only when it completes) are of interest. For example, when scheduling static web page requests at a web server, the size of the page gives a very good estimate of the time to serve that page. On the other hand while scheduling processes in an operating system, there is often little or no knowledge of the characteristics of the jobs.

*Dept. of Management Science & Engg. and (by courtesy) Computer Science. Research supported in part by NSF CAREER grant CCR-0339262.

†Supported in part by an Alfred P. Sloan Research Fellowship and by an NSF Career Award CCR-0093117.

‡Part of this work was done while the author was at Lucent Bell Labs.

STOC'04, June 13–15, 2004, Chicago, Illinois, USA.

In this paper we assume that the arrival times r_i are not known in advance and use the framework of *competitive analysis* to study the performance of our algorithms. However, all our algorithms can also be looked upon as approximation algorithms since they run in polynomial time. We present simple randomized and deterministic algorithms and prove that these algorithms are constant-competitive given an arbitrarily small amount of resource augmentation. All our algorithms are non-migratory, immediately dispatch a job to a machine upon arrival, and can simultaneously provide good approximation guarantees for all L_p norms of flow times. Further, our randomized algorithm also leads to a non-clairvoyant algorithm for flowtime. We believe that the simplicity of our algorithms (for example, our randomized algorithm merely assigns an incoming job to a machine chosen uniformly at random) makes them particularly useful in practical settings. Prior to our work, no algorithms were known to be constant competitive for flow time in the multiprocessor setting with arbitrarily small resource augmentation. Before stating our results more precisely, we describe some relevant related research to motivate our problem and to put our work in context.

Related Work and Motivation: In both theory and practice, the problem of minimizing flowtime has been much better understood in the single machine case. The preemptive algorithm SRPT (shortest remaining processing time first) is known to be optimal for minimizing flow time on a single machine in the clairvoyant case. Contrary to folklore belief, SRPT has also been shown to have good fairness characteristics in many settings [15, 22, 6, 7]. For the non-clairvoyant case, a strong lower bound of $\Omega(\sqrt{n})$ is known for deterministic algorithms [19]. In practice, the algorithm most commonly used is the multi-level feedback queue algorithm (MLF). MLF uses quanta of time units to minimize preemptions and SETF (shortest elapsed time first) is the theoretical equivalent of MLF. Motivated by MLF's good performance, a randomized variant of MLF has been analyzed and it is known be $\Theta(\log n)$-competitive for a single machine where n is the number of jobs in the sequence [9]. In the dynamic setting where the system essentially runs forever, a competitive ratio that depends on the length of the sequence of inputs is not attractive. However, there exists a lower bound of $\Omega(\log n)$ on randomized competitiveness [19] for the non-clairvoyant case. In situations such as this where strong lower bounds exist for the worst case, it is common and useful to weaken the power of the adversary. In the context of scheduling, Kalyanasundaram and Pruhs [17] proposed *resource augmentation* wherein we assume that the input is still chosen by the adversary in a worst-case manner but the online algorithm is given extra power. They proposed giving the algorithm a machine with a speed $(1+\epsilon)$ times the speed of the adversary's machine. With this assumption they showed that MLF is $O(1/\epsilon)$-competitive for non-clairvoyant scheduling on a single machine. Our work can be viewed as the first multi-processor generalization of this result.

The multi-processor setting is of much current relevance with processors and machines getting cheaper. Enterprise web and file servers often consist of banks of machines to enable processing of the large number of requests they need to handle. Thus it is an important question as to how to schedule on multiple processors to minimize average response time. There are a variety of constraints that are imposed in this situation. It is often impractical or expensive to move a job from one machine to another once it is assigned to a machine. Thus jobs cannot be *migrated*. In addition we prefer algorithms that assign a job as soon as it arrives into the system. Centralized queuing systems are undesirable for a variety of reasons including memory limitations and fault-tolerance. Thus we prefer *immediate-dispatching*. In practice it is common to use load balancing of some kind to assign jobs to machines and then use the known single machine algorithms such as MLF, FIFO, or SRPT. Our work justifies the performance of such algorithms.

Even though it was folklore that SRPT is optimal on a single machine, the analysis of its performance on multiple machines was done fairly recently by Leonardi and Raz [18] who showed that it is $O(\min(\log P, \log n/m))$-competitive where P is the ratio of the largest job size to the smallest job size. They also showed a matching lower bound on the competitive ratio of any randomized algorithm in the oblivious model. Note that SRPT migrates jobs between machines. Awerbuch et al. [5] subsequently gave a non-migratory algorithm with a similar competitive ratio but it did not immediately dispatch the jobs. More recently Avrahami and Azar [4] gave an algorithm that is both non-migratory and immediate-dispatching. In the resource augmentation model SRPT is known to be optimal if it is given 2-speed machines [21]. For the non-clairvoyant case Becchetti and Leonardi [9] showed that a randomized variant of MLF has a competitive ratio of $O(\min(\log n \log P, \log n \log n/m))$. These results do not completely explain the good performance of algorithms used in practice and further, the algorithms do not satisfy all the desired properties such as being non-migratory. There is no constant-competitive non-clairvoyant algorithm known even with a constant factor speedup.

Before we go on to describe our results, we discuss two related measures of performance of online algorithms. In the context of minimizing the average response time, the measure of *stretch* was introduced by Bender et al. [12]. The stretch of a job is the time it spends in the system divided by its processing time. For minimizing average stretch SRPT is shown to be 2-competitive on a single machine and $O(1)$-competitive on multiple machines [20].

In many online situations it is of concern that algorithms that try to optimize the average performance might perform quite poorly on some jobs, for the benefit of the many. However, it is usually difficult to optimize the worst case performance on all jobs (say the maximum flow time or the maximum stretch). In such situations it is natural to consider compromise measures such as L_p norms of performance for some $p \geq 1$. In particular the L_2 norm is commonly used. For flow time and stretch on a single machine Bansal and Pruhs [7] showed that with $(1+\epsilon)$-extra speed, well known algorithms such as SRPT, SJF, and SETF not only have good competitive ratios for flow time and stretch but also for their L_p norms. In our results we also consider these stronger measures.

In the context of multi-processors there are two types of resource augmentation. The algorithm can either use m machines with speed $(1+\epsilon)$ each, which we refer to as the extra-speed model, or can use $(1+\epsilon)m$ machines with speed 1, which we refer to as the extra-machine model.

Our Results: We show that with arbitrarily small resource augmentation in the form of extra speed or extra machines we can obtain simple constant-competitive ran-

Problem	One Machine			m Machines		
	Ratio	Res. Aug.	Ref	Ratio	Res. Aug.	Ref
C-Flowtime	1			$O(\log P)$		[18]
				$O(\log n/m)$		[18]
				1	2-speed	[21]
				$O(1/\epsilon)$	$(1+\epsilon)$-speed	this paper
				$O(1/\epsilon)$	$(1+\epsilon)$-machine	this paper
(L_p norm)	$O(1/\epsilon)$	$(1+\epsilon)$-speed	[7]	$O(1/\epsilon)$	$(1+\epsilon)$-speed	this paper
				$O(1/\epsilon)$	$(1+\epsilon)$-machine	this paper
NC-Flowtime	$O(\log n)$		[9]	$O(\log n \log P)$		[9]
				$O(\log n \log n/m)$		[9]
	$O(1/\epsilon)$	$(1+\epsilon)$-speed	[17]	$O(\frac{\log(1/\epsilon)}{\epsilon^7})$	$(1+\epsilon)$-speed	this paper
(L_p norm)	$O(1/\epsilon^{2+2/p})$	$(1+\epsilon)$-speed	[7]			
C-Stretch	2		[20]	$O(1)$		[20, 10, 14]
(L_p norm)	$O(1/\epsilon)$	$(1+\epsilon)$-speed	[7]	$O(1/\epsilon)$	$(1+\epsilon)$-speed	this paper
				$O(1/\epsilon)$	$(1+\epsilon)$-machine	this paper

Figure 1: Known Results on Online Algorithms for Flow Time and Stretch. C and NC refer to clairvoyant and non-clairvoyant setting respectively.

domized and deterministic algorithms for minimizing flow time and stretch. The randomized algorithm is the following: when a new jobs arrives it is assigned immediately to a machine chosen uniformly at random. The deterministic assignment algorithm we analyze is that of Avrahami and Azar [4] which assigns a new job based on the prior load that it had assigned to the machines. Once the jobs are assigned to the machines, in the clairvoyant case we use either SRPT, SJF or SCF (smallest class first, to be defined later), and in the non-clairvoyant case we use SETF. Our results are summarized below. All our results hold for resource augmentation for arbitrarily small $\epsilon > 0$.

- The randomized algorithm is $O(\frac{\log(1/\epsilon)}{\epsilon^3})$-competitive for both flow time and stretch in the clairvoyant setting in both models of resource augmentation. It is $O(\frac{\log(1/\epsilon)}{\epsilon^7})$-competitive in the non-clairvoyant setting in the extra speed model. The competitive ratio holds even against an *adaptive offline* adversary. We note that in the non-clairvoyant setting, for any $\alpha > 0$, no deterministic immediate-dispatch algorithm can be $o(m/\alpha)$-competitive even with α-speed machines.
- The deterministic algorithm is $O(1/\epsilon)$-competitive for both flow time and stretch in the clairvoyant setting. It is also $O(1/\epsilon)$-competitive for minimizing L_p norms of flow time and stretch for any $p \geq 1$. These results hold in both models of resource augmentation.
- The randomized algorithm is competitive for minimizing L_p norms of flow time and stretch, even in the non-clairvoyant setting. We omit the details of this result from this version of the paper.

Our results validate the use of simple load balancing strategies combined with well understood single machine algorithms to optimize the throughput and fairness of multiprocessor server systems. Figure 1 summarizes our results in the context of known results. We also believe that the randomized algorithm together with HDF (highest density first) is competitive for weighted flow time.

Organization: Section 2 formally defines the problem and describes various algorithms that we study. We study the randomized load balancing algorithm for minimizing flow time and stretch in both models of resource augmentation in Section 3. We also show that this simple load balancing scheme yields an $O(1)$-competitive non-clairvoyant algorithm in the extra-speed model when combined with the single machine non-clairvoyant algorithm SETF. In Section 4, we analyze the deterministic algorithm of Avrahami and Azar [4], and show that it is $O(1)$-competitive for minimizing L_p norms of flow time and stretch on both models of resource augmentation.

2. PRELIMINARIES

We consider the online problem of scheduling jobs in a multiprocessor environment. We assume that jobs can be *preempted*. A job's existence is known only upon its arrival. In the clairvoyant setting we assume that the job's processing time is known on arrival. In the non-clairvoyant setting the processing time is known only when the job completes its processing time and leaves the system. We denote by p_j the processing time of job j. The flow time F_j of a job $j \in J$ is defined as the time spent by the job in the system before its processing is completed; equivalently $F_j = C_j - r_j$ where r_j is the release time of j and C_j is the completion time of j. The stretch S_j of a job j is defined to be F_j/p_j. For any algorithm A, let $F_p(A) = (\sum_j F_j^p)$ and $S_p(A) = (\sum_j (S_j)^p)$ be the sums of the pth powers of the flow time and stretch of the jobs. We consider the objectives of minimizing the L_p norms of the flow time and stretch, $(F_p(A))^{1/p}$ and $(S_p(A))^{1/p}$ respectively. An algorithm is *migratory* if it processes a job on more than one machine, otherwise it is *non-migratory.*

We use resource augmentation to analyze our algorithm. We assume that the optimal offline algorithm has m machines of speed 1 to schedule the jobs. We analyze the performance of our algorithm under two different scenarios: a) when the algorithm is given m machines of speed $(1+\epsilon)$, and b) when the algorithm is given $\lceil(1+\epsilon)m\rceil$ machines of speed 1. In both cases we assume that the optimal algorithm can migrate jobs while our algorithm does not.

We assume without loss of generality that the processing time of any job is at least one. We say that a job j is in class k if its processing time p_j lies in the interval $[2^k, 2^{k+1})$. We denote the class of a job j by $\textsc{Class}(j)$ and its arrival time by r_j. We consider non-migratory immediate-dispatch algorithms. A job is assigned to a machine as soon as it arrives and it is processed only on the machine it is assigned to. In this setting, if job sizes are known upon arrival (the clairvoyant case) we note that it is optimal to process the jobs on each machine by SRPT. However it is easier to analyze the following sub-optimal algorithm: at each time, process the job in the queue from the lowest class and break ties within a class in favor of the job that has arrived earlier. This priority rule was used in [14] and subsequently in [4] to simplify the analysis of algorithms for multiple machines. We refer to this algorithm as SCF (shortest class first). In the non-clairvoyant case when the job sizes are not known, we use the well known algorithm SETF (shortest elapsed time first).

We analyze a randomized and a deterministic strategy to assign jobs to machines when the arrive. The randomized strategy is simple: assign a new job to a machine chosen uniformly at random from the set of available machines. The deterministic strategy is the balancing algorithm of Avrahami and Azar [4]. The algorithm maintains for each machine i and class k the total processing time of all jobs that have been assigned to i in class k by time t. Let $Z_k^i(t)$ denote this quantity. The algorithm assigns a new job that arrives at time t from class k to machine j where $j = \text{argmin}_\ell Z_k^\ell(t)$. Note that the randomized strategy is *oblivious* to the processing time of the arriving job and is also *stateless* while the deterministic strategy is neither.

3. RANDOMIZED LOAD BALANCING

We will assume that no two jobs have the same arrival time; since job arrivals are real numbers, this does not result in any loss of generality. We define a total order $\preceq$ on all jobs as follows: $i \preceq j$ iff $\textsc{Class}(i) < \textsc{Class}(j)$ OR ($\textsc{Class}(i) = \textsc{Class}(j)$ AND $r_i \leq r_j$). A job is *active* if it has arrived but has not yet been completed. We will use $C_i^{(R)}$ to denote *completion time* of job i in $\textsc{Rand}$.

For a given job i that is alive at time t, let $S_i^{(*)}(t)$ be the total remaining processing time of all active jobs $j \preceq i$ at time t, divided by m. Let $V_{\leq h}^{(*)}(t)$ denote the volume of jobs in OPT's queue in classes 1 to h at time t. Note that $S_i^{(*)}(t) \leq V_{\leq k}^{(*)}(t)/m$ where $k = \textsc{Class}(i)$. Define $S_i^{(R)}(t)$ to be the total remaining processing time of all jobs $j \preceq i$ that are active at time t and which are assigned to the machine that job i is assigned to in $\textsc{Rand}$. Define $A_i^{(*)}(t_1, t_2)$ to be the total processing time of all jobs $j \preceq i, r_i \in (t_1, t_2]$, divided by m. The quantities $S_i^{(*)}(t)$ and $A_i^{(*)}(t_1, t_2)$ measure the average remaining volume and average total arrivals, respectively, on a machine in OPT. Define $A_i^{(R)}(t_1, t_2)$ to be the total processing time of all jobs $j \preceq i, r_i \in (t_1, t_2]$ which are assigned to the machine that job i is assigned to in $\textsc{Rand}$.

We say that a job i that is in $\textsc{Rand}$'s queue at time t is δ-*bad* at t if $S_i^{(R)}(t) > (1+\delta)S_i^{(*)}(t)$; it is δ-*good* otherwise. We define the δ-*overhead*, denoted $T_\delta(i)$, of job i to be the total length of all intervals during which job i is δ-bad. Observe that this is the total length of the interval during which we cannot "charge" this job to a job in OPT's queue. The following lemma relates the total overhead to the total flow time.

Lemma 3.1 *The total flow time of* $\textsc{Rand}$ *is at most* $O((1+\delta))\frac{m'}{m}\text{OPT} + \sum_i T_\delta(i)$ *where* m' *is the number of machines in* $\textsc{Rand}$.

Proof. We set up a charging scheme that charges all δ-good jobs at t to the payment available in jobs from OPT's queue. Let $n_h^*(t)$ be the number of jobs in class h that are alive at t in OPT. We set up payment from OPT's queue to different classes as follows. Class h receives a payment from classes 1 to $h-1$, say $\alpha(t)$. Class h retains $(\alpha(t) + n_h^*(t))/2$ and passes on $(\alpha(t)+n_h^*(t))/2$ to class $h+1$. Note that each job pays for one unit.

From the payment scheme, the payment available to class h is $n_1^*(t)/2^h + n_2^*(t)/2^{h-1} + \ldots + n_h^*(t)/2$ which is $(n_1^*(t) + 2n_2^*(t) + \ldots + 2^{h-1}n_h^*(t))/2^h$. Since each job in class i has processing time in $[2^i, 2^{i+1})$, it follows that $(n_1^*(t)+2n_2^*(t)+\ldots+2^{h-1}n_h^*(t))/2^h \geq (V_{=1}^*(t)+V_{=2}^*(t)+\ldots+V_{=h}^*(t))/2^{h+2} = V_{\leq h}^*(t)/2^{h+2}$.

Thus, the total payment available to jobs in class h is $\Omega(V_{\leq h}^{(*)}(t)/2^h)$. We partition this payment to each of the m' machines equally, hence the payment available to class h on each machine is $\Omega(V_{\leq h}^{(*)}(t)/2^h)\frac{m}{m'}$. We distribute this payment to jobs in class h in $\textsc{Rand}$'s queue in the order defined by $\preceq$. We give each job a payment of $\frac{m}{m'}\frac{1}{1+\delta}$. From simple calculations, we can pay for a job i if it is δ-good. The bound on $\textsc{Rand}$'s flow time follows. $\square$

We now need to bound the expected δ-overhead of a job in terms of its processing time. Theorems 3.2 and 3.3 are proved in sections 3.1 and 3.2, respectively.

Theorem 3.2 *In the extra machines model, for any job* i, $\mathbf{E}\left[T_\epsilon(i)\right] = O\left(\frac{p_i}{\epsilon^3}\log\frac{1}{\epsilon}\right)$.

Theorem 3.3 *In the extra speed model, for any job* i, $\mathbf{E}\left[T_{\epsilon/2}(i)\right] = O\left(\frac{p_i}{\epsilon^3}\log\frac{1}{\epsilon}\right)$.

Lemma 3.1 and theorems 3.2 and 3.3 immediately yield:

Theorem 3.4 *The expected flow time of* $\textsc{Rand}$ *is* $O(1+\epsilon)\text{OPT} + O(\frac{\log 1/\epsilon}{\epsilon^3})\sum_i p_i$ *in both the extra machine and extra speed models. The expected stretch of* $\textsc{Rand}$ *is* $O(1+\epsilon)\text{OPT} + O(\frac{\log 1/\epsilon}{\epsilon^3})n$ *in both the extra machine and extra speed models. These bounds hold against both oblivious and adaptive adversaries.*

Corollary 3.5 $\textsc{Rand}$ *is* $O(\frac{\log 1/\epsilon}{\epsilon^3})$*-competitive for both flow time and stretch in the extra machine and extra speed models with parameter* ϵ.

Since the randomized algorithm is non-clairvoyant, we can couple it with SETF to obtain an online non-clairvoyant algorithm. The following theorem analyzes this algorithm and also presents a lower bound on deterministic algorithms.

Theorem 3.6 $\textsc{Rand}$ *with* SETF *(*$\textsc{RandSETF}$*) is a* $(1+\epsilon)$*-speed* $O(\frac{\log(1/\epsilon)}{\epsilon^7})$*-competitive* non-clairvoyant *and* non-migratory *algorithm to minimize average flow time. No deterministic non-clairvoyant algorithm with immediate dispatch can be* $o(m/(1+\epsilon))$*-competitive for minimizing average flow time even with* $(1+\epsilon)$*-speed machines.*

PROOF. Fix any instance I. Let I_j be the (random) instance presented to machine M_j by RAND. By Theorem 3.4,

$$\sum_{j=1}^{m} \mathbf{E}[\mathrm{OPT}_{(1+\epsilon)}(I_j)] \leq \sum_{j=1}^{m} \mathbf{E}[\mathrm{SCF}_{(1+\epsilon)}(I_j)] \leq O(\frac{1}{\epsilon^3 \log 1/\epsilon})\mathrm{OPT}(I) .$$

Also, from [7], on a single machine, for any instance H, the *non-clairvoyant* algorithm SETF satisfies $\mathrm{SETF}_{1+\epsilon}(H) \leq O(\frac{1}{\epsilon^4})\mathrm{OPT}(H)$. Combining this with the above equation, we get

$$\begin{aligned}\mathbf{E}[\mathrm{RANDSETF}_{(1+\epsilon)^2}(I)] &= \sum_{j=1}^{m} \mathbf{E}[\mathrm{SETF}_{(1+\epsilon)^2}(I_j)] \\ &\leq O(\frac{1}{\epsilon^4})\sum_{j=1}^{m} \mathbf{E}[\mathrm{OPT}_{(1+\epsilon)}(I_j)] \\ &\leq O(\frac{\log 1/\epsilon}{\epsilon^7})\mathrm{OPT}(I) .\end{aligned}$$

To see the lower bound, consider the instance in which m^2 jobs arrive at time 0. Any algorithm with immediate dispatch must assign at least m jobs to one of the machines, say M_1, immediately. Fix the processing time of the first m jobs assigned to M_1 to be 1 each and all other processing times are 0. Clearly, the flow time for this algorithm is $\Omega(m^2/(1+\epsilon))$. The optimal value, on the other hand, is m, obtained by assigning each unit length job to a distinct machine. □

3.1 Bounding the overhead in the extra machine model

In the extra machine model RAND uses $m' = (1+\epsilon)m$ machines of speed 1. We focus on some fixed job i and analyze $\mathbf{E}[T_\epsilon(i)]$. We assume without loss of generality that job i is assigned by RAND to machine M_1.

For $\alpha > 0$, we will say that an infinite sequence $\langle x\rangle = x_1, x_2, \ldots$ is α-bounded if $0 < x_i \leq \alpha$ for all $i \geq 1$. Let $z_i(\theta)$ denote a Bernoulli 0,1 variable such that $\mathbf{Pr}[z_i(\theta) = 1] = \theta$. We will assume that the variables $z_1(\theta), z_2(\theta), \ldots$ are independent, and may omit the argument θ where its value is obvious from the context. If $\langle x\rangle$ represents a sequence of arrivals to the system, then the random variable

$$\max\left\{0, \sum_{j\leq i} x_i z_i \left(\frac{1}{m'}\right) - \frac{1}{m}\sum_{j\leq i} x_i\right\}$$

represents the excess total processing time assigned to machine M_1 by RAND, compared to the average total processing time assigned to a machine by OPT, after i arrivals. The next lemma bounds the maximum value of this excess load for any α-bounded arrival sequence.

Lemma 3.7 *For $0 < \delta < 1/4$, and for an arbitrary α-bounded sequence $\langle x\rangle$,*

$$\mathbf{E}\left[\max\left\{0, \max_{i\geq 1}\left\{\sum_{j\leq i} x_i z_i \left(\frac{1}{m(1+\delta)}\right) - \frac{1}{m}\sum_{j\leq i} x_i\right\}\right\}\right] = O\left(\frac{\alpha}{\delta^2}\log\frac{1}{\delta}\right).$$

We now analyze the total volume of jobs that must arrive before RAND can compensate for an initial excess processing time of y.

Lemma 3.8 *Let $\langle x\rangle$ be an arbitrary α-bounded sequence satisfying $\sum_{j\leq i} x_i \to \infty$ as $i \to \infty$. For $0 < \delta < 1/4$, and $y \geq 0$, let $k^{(c)}(y)$ denote the largest value of k such that $y + \sum_{i=1}^{k} x_i z_i \left(\frac{1}{m(1+\delta)}\right) \geq \sum_{i=1}^{k} x_i/m$ and let $X^{(c)}(y)$ denote $\sum_{i=1}^{1+k^{(c)}(y)} x_i/m$. Then*

$$\mathbf{E}\left[X^{(c)}(y)\right] = O\left(\frac{y}{\delta} + \frac{\alpha}{\delta^3}\log\frac{1}{\delta}\right).$$

The proofs of Lemmas 3.7 and 3.8 are technical and can be found in the appendix. With these technical lemmas in place, we are ready to bound the expected ϵ-overhead of a job in the extra machines model.

PROOF OF THEOREM 3.2. Assume wlog that job i goes to machine M_1. Let $\gamma = \max\{0, S_i^{(R)}(r_i) - S_i^{(*)}(r_i)\}$. Observe that $\gamma \leq \max\{0, \max_{t<r_i}\{A_i^{(R)}(t, r_i) - A_i^{(*)}(t, r_i)\}\}$. If we ignore jobs j which do not satisfy $j \preceq i$ and look at time in reverse, then the set of arrivals in $(-\infty, r_i)$ forms a $2p_i$-bounded sequence. Now we can invoke Lemma 3.7 with $\delta = \epsilon$ to obtain

$$\mathbf{E}[\gamma] = O\left(\frac{p_i}{\epsilon^2}\log\frac{1}{\epsilon}\right). \quad (1)$$

If job i is never ϵ-bad, then we are done. Else, let t_1 denote the first time instant in $[r_i, C_i^{(R)})$ such that $S_i^{(R)}(t_1) \geq (1+\epsilon)S_i^{(*)}(t_1)$. Let $\beta = \max\{0, A_i^{(R)}(r_i, t_1) - A_i^{(*)}(r_i, t_1)\}$. Observe that β is stochastically dominated by the quantity $\max\{0, \max_{t>r_i}\{A_i^{(R)}(r_i, t) - A_i^{(*)}(r_i, t)\}\}$. We now invoke Lemma 3.7 to obtain

$$\mathbf{E}[\beta] = O\left(\frac{p_i}{\epsilon^2}\log\frac{1}{\epsilon}\right). \quad (2)$$

Now, $S_i^{(R)}(t_1) \leq S_i^{(*)}(t_1) + \gamma + \beta$, and since job i is ϵ-bad at time t_1, we also have $S_i^{(R)}(t_1) \geq (1+\epsilon)S_i^{(*)}(t_1)$. Thus, $S_i^{(*)}(t_1) \leq (\gamma+\beta)/\epsilon$ and

$$S_i^{(R)}(t_1) \leq (1+\epsilon)(\gamma+\beta)/\epsilon. \quad (3)$$

Now consider the sequence $\langle x\rangle_{(t_1,\infty)}$ of processing times[1] of jobs $j \preceq i$ that arrive after time t_1 and let $X^{(c)}(y)$ be as defined in the statement of Lemma 3.8. Let $t_2 = t_1 + S_i^{(R)}(t_1) + X^{(c)}(\gamma+\beta)$.

Our goal is to now prove that job i cannot be ϵ-bad at any time $t > t_2$. If job i finishes before time t_2, then we are

[1]To invoke Lemma 3.8, we need the technical condition that $\sum_{k\leq j} x_j \to \infty$ as $j \to \infty$. If the sequence $\langle x\rangle_{(t_1,\infty)}$ does not satisfy this property, then we can introduce fake jobs of size p_i at the end of this sequence without affecting the proof. We omit the details.

done. Let t' be the time when $A_i^{(*)}(t_1, t') = X^{(c)}(\gamma+\beta)$. We will now consider two cases:

Case 1: $\mathbf{t_2 \le t'}$. By the definition of $X^{(c)}$, $A_i^{(R)}(t_1, t') \le A_i^{(*)}(t_1, t') - \gamma - \beta$. Hence,

$$\begin{aligned} A_i^{(R)}(t_1, t_2) &\le A_i^{(*)}(t_1, t') - \gamma - \beta \\ &\le X^{(c)}(\gamma+\beta) - \gamma - \beta \\ &\le X^{(c)}(\gamma+\beta) \\ &\le t_2 - t_1 - S_i^{(R)}(t_1). \end{aligned}$$

As long as job i is on machine M_1, that machine only executes jobs $j \preceq i$. Hence the last inequality implies that job i finishes by time t_2.

Case 2: $\mathbf{t_2 > t'}$. Consider any time $t \in [r_i, C^{(R)}]$, $t > t_2$. Machine M_1 must be busy executing jobs $j \preceq i$ between times t_1 and t. Hence, by Lemma 3.8,

$$\begin{aligned} S_i^{(R)}(t) &= S_i^{(R)}(t_1) + A_i^{(R)}(t_1, t) - (t - t_1) \\ &\le (S_i^{(*)}(t_1) + \gamma + \beta) + \\ &\qquad (A_i^{(*)}(t_1, t) - \gamma - \beta) - (t - t_1) \\ &\le S_i^{(*)}(t), \end{aligned}$$

and hence job i cannot be ϵ-bad at time t.

Thus, job i can be ϵ-bad only during the interval $[t_1, t_2]$ and the total ϵ-overhead $T_\epsilon(i)$ of this job is at most $t_2 - t_1 = (1+\epsilon)(\gamma+\beta)/\epsilon + X^{(c)}(\gamma+\beta)$. Consider $X = X^{(c)}(\gamma+\beta)$. Since Lemma 3.8 applies to an *arbitrary* α-bounded arrival sequence, we can ignore any dependence of X on $\langle x \rangle_{(t_1,\infty)}$ to obtain

$$\mathbf{E}[X|\gamma, \beta] \le \rho\left(\frac{\gamma+\beta}{\epsilon} + \frac{1}{\epsilon^3}\log\frac{1}{\epsilon}\right)$$

for some suitably large constant ρ. Removing the conditioning on γ, β yields

$$\begin{aligned} \mathbf{E}[X] &\le \rho\left(\frac{\mathbf{E}[\gamma+\beta]}{\epsilon} + \frac{1}{\epsilon^3}\log\frac{1}{\epsilon}\right), \text{ and hence,} \\ \mathbf{E}[T_\epsilon(i)] &\le (1+\epsilon)\mathbf{E}[\gamma+\beta]/\epsilon + \rho\left(\frac{\mathbf{E}[\gamma+\beta]}{\epsilon} + \frac{1}{\epsilon^3}\log\frac{1}{\epsilon}\right). \end{aligned}$$

Substituting equations 1 and 2 in the above expression yields the desired bound on the ϵ-overhead. □

3.2 Bounding the overhead in the extra speed model

In the extra speed model RAND uses m machines of speed $1+\epsilon$. Once again we focus on some fixed job i and assume without loss of generality that it is assigned by RAND to machine M_1.

For the case of speed-bounded machines, the following variant of Lemma 3.7 is useful. The proof is along the lines of proof of Lemma 3.7 in the appendix and is omitted.

Lemma 3.9 *For $0 < \delta < 1/4$, and for an arbitrary α-bounded sequence $\langle x \rangle$,*

$$\mathbf{E}\left[\max\left\{0, \max_{i\ge 1}\left\{\sum_{j\le i} x_i z_i\left(\frac{1}{m}\right) - \frac{1+\delta}{m}\sum_{j\le i} x_i\right\}\right\}\right] = O\left(\frac{\alpha}{\delta^2}\log\frac{1}{\delta}\right).$$

PROOF OF THEOREM 3.3. Let $\gamma_1 = \max\{0, S_i^{(R)}(r_i) - (1+\epsilon/2)S_i^{(*)}(r_i)\}$. Let $\gamma_2 = \max\{0, \max_{t\ge r_i}\{A_i^{(R)}(r_i,t) - (1+\epsilon/2)A_i^{(*)}(r_i,t)\}\}$. Suppose $C_i^{(R)} \le r_i + (2/\epsilon)(\gamma_1+\gamma_2)$. Then the total flow time, and hence the $(\epsilon/2)$-overhead, of job i is at most $(2/\epsilon)(\gamma_1+\gamma_2)$. Now consider the case when $C_i^{(R)} > r_i + (2/\epsilon)(\gamma_1+\gamma_2)$. Consider any time $t \in [r_i + (2/\epsilon)(\gamma_1+\gamma_2), C_i^{(R)})$. Since job i has not completed by time t, machine M_1 must have been busy executing jobs $j \preceq i$ in the interval $[r_i, t]$. Now we use the fact that RAND has machines that run at speed $(1+\epsilon)$ to obtain

$$\begin{aligned} S_i^{(R)}(t) &= S_i^{(R)}(r_i) + A_i^{(R)}(r_i, t) - (1+\epsilon)(t - r_i) \\ &\le \gamma_1 + (1+\epsilon/2)S_i^{(*)}(r_i) + \gamma_2 + (1+\epsilon/2)A_i^{(*)}(r_i,t) \\ &\qquad -(1+\epsilon/2)(t-r_i) - (\epsilon/2)(t-r_i) \\ &\le (1+\epsilon/2)\left(S_i^{(*)}(r_i) + A_i^{(*)}(t, r_i) - (t-r_i)\right) \\ &\qquad -(\epsilon/2)\left(t - r_i - (2/\epsilon)(\gamma_1+\gamma_2)\right). \end{aligned}$$

Even if OPT has been executing only jobs $j \preceq i$ during the interval $[r_i, t)$, we have $S_i^{(*)}(t) \ge S_i^{(*)}(r_i) + A_i^{(*)}(t, r_i) - (t - r_i)$. Combining this with the fact that $t - r_i > (2/\epsilon)(\gamma_1+\gamma_2)$, we obtain $S_i^{(R)}(t) \le (1+\epsilon/2)S_i^{(*)}(t)$, i.e., job i is not $(\epsilon/2)$-bad at time t.

Thus, job i can only be bad for a total duration of $(2/\epsilon)(\gamma_1+\gamma_2)$. We can use Lemma 3.9 to immediately obtain $\mathbf{E}[\gamma_2] = O((p_i/\epsilon^2)\log(1/\epsilon))$. The only missing piece is $\mathbf{E}[\gamma_1]$. Observe that

$$\begin{aligned} S_i^{(R)}(t) &= \max_{s\le t}\left\{A_i^{(R)}(s,t) - (1+\epsilon)(t-s)\right\} \\ &\le \max_{s\le t}\left\{A_i^{(R)}(s,t) - (1+\epsilon/2)(t-s)\right\} \end{aligned}$$

and

$$S_i^{(*)}(t) = \max_{s\le t}\left\{A_i^{(*)}(s,t) - (t-s)\right\}.$$

Now,

$$\begin{aligned} &S_i^{(R)}(t) - (1+\epsilon/2)S_i^{(*)}(t) \\ &\le \max_{s\le t}\left\{A_i^{(R)}(s,t) - (1+\epsilon/2)(t-s)\right\} \\ &\qquad -(1+\epsilon/2)\max_{s\le t}\left\{A_i^{(R)}(s,t) - (t-s)\right\} \\ &\le \max_{s\le t}\left\{A_i^{(R)}(s,t) - (1+\epsilon/2)A_i^{(*)}(s,t)\right\}. \end{aligned}$$

Now we can use Lemma 3.9 to also bound $\mathbf{E}[\gamma_1]$. Consequently,

$$\mathbf{E}\left[T_{\epsilon/2}(i)\right] = O\left(\frac{p_i}{\epsilon^3}\log\frac{1}{\epsilon}\right).$$

□

4. DETERMINISTIC LOAD BALANCING

The deterministic algorithm that we analyze is the algorithm of Avrahami and Azar [4] that we described in Section 2. We refer to this algorithm as Algorithm $\mathcal{A}$. We set up some notation that we use in the rest of the section. Let $Z_k^i(t)$ denote the total processing time of all jobs that have been assigned to machine i in class k by time t. Algorithm $\mathcal{A}$ assigns a job arriving at time t with class k to the machine M_i with the least $Z_k^i(t)$ value; breaking ties arbitrarily. We use $V(t, X)$ to denote the volume, in other words the remaining processing time, of jobs in set X at time t. Subscripts are used to restrict the jobs to specified classes. We will often use predicates to define a set of jobs. For example $V_{\le k}(t, r_i \le t')$ indicates the volume of jobs that were released by time t' and which were in classes 1 to k. $P(X)$ denotes the processing times of jobs in set X. We use the superscript $*$ to indicate quantities associated with some fixed optimal schedule that we compare the algorithm's schedule with. The quantity $Z^i_{\le k}(t)$ denotes the total processing time of jobs assigned to machine i by time t in classes at most k. The quantity $Q^i_{\le k}(t)$ denotes the amount of time that machine i spent on jobs of class at most k till time t. Let $R^i_{\le k}(t)$ denote $Z^i_{\le k}(t) - Q^i_{\le k}(t)$ the residual amount of processing left in the queue of machine i at time t in classes up to k. Note that $V_{\le k}(t) = \sum_i R^i_{\le k}(t)$.

We now state two properties of the algorithm $\mathcal{A}$ that were shown in [4] and that we use in our analysis.

Lemma 4.1 *In the schedule of $\mathcal{A}$, for any two machines j and j' and for any time t, $|Q^j_{\le k}(t) - Q^{j'}_{\le k}(t)| \le 2^{k+2}$.*

Lemma 4.2 *In the schedule of $\mathcal{A}$, if at time t there exists a machine j such that $R^j_{\le k}(t) = 0$ then for every other machine j', $R^{j'}_{\le k}(t) \le 2^{k+3}$.*

We focus on the L_p norm F_p here, identical arguments apply to L_p norm of S_p as well. Our analysis will establish that $F_p(\mathcal{A})$ is at most $(O(1+1/\epsilon))^p F_p(\mathrm{OPT})$ where OPT denotes an optimal algorithm. For an algorithm H, let $U(H,t)$ denote the set of unfinished jobs in the queue of H at time t. Also, let $\mathrm{Age}^p(X,t)$ denote the sum over all jobs $J_i \in X$ of $(t-r_i)^{p-1}$. Then $F_p(H) = p\int_t \mathrm{Age}^p(U(H,t),t)dt$. Thus in order to show that H is c-competitive against OPT, it is sufficient to show that at all times, $\mathrm{Age}^p(U(H,t),t)$ is at most $c^p \mathrm{Age}^p(U(\mathrm{OPT},t),t)$. This is the approach taken by [7] for the single machine case where they showed that the unfinished jobs in the online algorithms queue can always be charged to unfinished jobs in OPT's queue such that the above condition holds. Unfortunately, we cannot establish such a local competitiveness condition in the multiple machines case. The main difficulty in extending the result to multiple machines comes from load balancing of jobs across the machines. The simple round-robin load balancing approach of algorithm $\mathcal{A}$ leads to non-uniform workloads across the machines. In particular, at any moment in time, while some machines are idle, another group of machines may have several unfinished jobs in their queues. Thus while algorithm $\mathcal{A}$ has unfinished jobs in its queue, OPT may have no unfinished jobs to charge to in its queue. However, as indicated by Lemmas 4.1 and 4.2, these variations in the work-load are reasonably bounded. We use this to show that the optimal algorithm carries sufficiently many jobs to enable a local charging for all jobs that have waited *sufficiently long* in the queue of $\mathcal{A}$. In other words, the jobs must wait for some time (say, a constant times their processing time) to participate in this charging scheme. While a job j is waiting to qualify for local charging, we charge its delay to the term F_j^p in OPT. Thus our analysis relies on a combination of local and global charging rules. In what follows, we describe in detail how these ideas can be made to work in the multiple machines setting. The charging scheme we describe follows the overall approach developed in [7].

Charging Scheme: Let $\mathcal{D} = U(\mathcal{A},t) - U(\mathrm{OPT},t)$ be the set of jobs which have finished processing in OPT's schedule at time t but not in the schedule produced by $\mathcal{A}$. Index the jobs in $\mathcal{D}$ in increasing order of their processing times. Consider a job $J_i \in \mathcal{D}$. Suppose J_i is in class k. Let t' denote the time $t - \epsilon'(t-r_i)$, where ϵ' is a constant less than 1 that we fix later. Let $V^*_{\le k}(t, r_j \le t')$ denote the amount of processing left in the set of jobs in $U(\mathrm{OPT},t)$ which are in class at most k and were released before time t'. We allocate to J_i an $\epsilon'' p_i$ amount of work from $V^*_{\le k}(t, r_j \le t')$ which has not been previously allocated to a lower indexed job in $\mathcal{D}$, where ϵ'' is another constant less than 1 that we fix later. We will do this allocation only when $t - r_i$ is at least $\alpha \cdot 2^k$ for some α. If J_i does not satisfy this condition, we simply charge its waiting time to J_i's contribution to the optimal's objective. Since J_i contributes at least $(2^k)^p$ to $F_p(\mathrm{OPT})$, and the maximum contribution from this waiting is $(\alpha 2^k)^p$, the charging ratio is bounded by α^p. On the other hand, for jobs that satisfy the condition, at most $2/\epsilon''$ jobs in $U(\mathcal{A},t)$ are assigned to any one job in OPT. Moreover, for each participating job we charge a contribution of $(t-r_i)^{p-1}$ to $(t-t')^{p-1} = (\epsilon'(t-r_i))^{p-1}$. Thus for these jobs, at time t, the contribution to Age^p in $\mathcal{A}$ is at most $2/(\epsilon''(\epsilon')^{p-1})$ times $\mathrm{Age}^p(U(\mathrm{OPT},t),t)$. We will show that this approach can be made to work when $\alpha, 1/\epsilon'$, and $1/\epsilon''$ are all $O(1+1/\epsilon)$. Thus we can establish that $F_p(\mathcal{A})$ is at most $(O(1+1/\epsilon))^p F_p(\mathrm{OPT})$, giving our main result.

Thus the heart of the analysis is in showing that the association described above always works, i.e., we never run out of jobs to assign to J_i. Given time u and u', $u \ge u'$, let $J[u',u]$ be the set of jobs released during $[u',u]$. Let $P_{\le k}(J[u',u])$ be the total processing time of the jobs in $J[u',u]$ which are in class at most k. Now the possible volume of jobs which we can charge J_i to is $V^*_{\le k}(t, r_j \le t')$. The claim below lower bounds this volume.

Claim 4.3

$$V^*_{\le k}(t, r_j \le t') \ge V^*_{\le k}(r_i, r_j \le r_i) + P_{\le k}(J[r_i, t']) - m(t - r_i). \tag{4}$$

Proof. The machines in OPT's schedule can perform at most $m(t-r_i)$ during $[r_i, t]$. The volume of jobs of group at most k that needs to be processed during this interval is at least $V^*_{\le k}(r_i, r_j \le r_i) + P_{\le k}(J[r_i, t'])$. Further, the release time of all the jobs in this expression is at most t'. This proves the claim. □

Recall that $V^*_{\le k}(t, r_j \le t')$ is the volume we want to use to charge J_i. Some of this volume has been consumed while charging jobs in $J_1, \ldots, J_{i-1}$. Consider the jobs in this set which were released *after* time r_i. Since all these jobs were finished in OPT by time t, their total processing time can be at most $m(t-r_i)$. Now consider those jobs in this set which were released *before* time r_i. These jobs are a subset of the jobs in $U(\mathcal{A}, r_i)$. So their processing time is at most $P_{\le k}(U(\mathcal{A}, r_i))$. Thus, the total volume of jobs that is sufficient for charging jobs $J_1, \ldots, J_i$ is

$$\epsilon'' \left(m(t - r_i) + P_{\le k}(U(\mathcal{A}, r_i)) \right). \tag{5}$$

From Equations 4 and 5, it suffices to show that

$$\begin{aligned} V^*_{\le k}(r_i, r_j \le r_i) + P_{\le k}(J[r_i, t']) - m(t - r_i) \\ \ge \epsilon'' \left(m(t - r_i) + P_{\le k}(U(\mathcal{A}, r_i)) \right). \end{aligned} \tag{6}$$

In what follows, we first establish for the case of machines with extra speed in section 4.1, and then shjow that the analysis can be carried over to the case of extra machines as well with minor modifications.

4.1 Extra speed

We start by establishing a relation between the unprocessed volume of jobs in the optimal algorithm and the unfinished jobs in $\mathcal{A}$.

Lemma 4.4 *Consider the schedule produced by $\mathcal{A}$. Suppose a machine remains busy in the interval $[t_1, t_2]$ and processes jobs only from classes 1 to k during this interval. Then the total amount of processing done by all the m machines on jobs of class at most k during $[t_1, t_2]$ is at least $(1+\epsilon)m(t_2 - t_1) - m2^{k+3}$.*

PROOF. Given a machine j, and a time u, recall that $Q^j_{\leq k}(u)$ is the total volume of jobs of class at most k processed by machine j till time u. Then from Lemma 4.1, if j and j' are two arbitrary machines, then $|Q^j_{\leq k}(u) - Q^{j'}_{\leq k}(u)| \leq 2^{k+2}$. Let j be the machine which remains busy from t_1 to t_2 processing jobs from class at most k. So, $Q^j_{\leq k}(t_2) - Q^j_{\leq k}(t_1) = (1+\epsilon)(t_2 - t_1)$. If j' is any other machine, we know from Lemma 4.1 that $|Q^j_{\leq k}(t_1) - Q^{j'}_{\leq k}(t_1)| \leq 2^{k+2}$ and $|Q^j_{\leq k}(t_2) - Q^{j'}_{\leq k}(t_2)| \leq 2^{k+2}$. Therefore,

$$\begin{aligned}
&Q^{j'}_{\leq k}(t_2) - Q^{j'}_{\leq k}(t_1) \\
&= \left(Q^{j'}_{\leq k}(t_2) - Q^j_{\leq k}(t_2)\right) + \left(Q^j_{\leq k}(t_1) - Q^{j'}_{\leq k}(t_1)\right) \\
&\quad + \left(Q^j_{\leq k}(t_2) - Q^j_{\leq k}(t_1)\right) \\
&\geq \left(Q^j_{\leq k}(t_2) - Q^j_{\leq k}(t_1)\right) - |Q^{j'}_{\leq k}(t_2) - Q^j_{\leq k}(t_2)| \\
&\quad - |Q^j_{\leq k}(t_1) - Q^{j'}_{\leq k}(t_1)| \\
&\geq (1+\epsilon)(t_2 - t_1) - 2^{k+3}
\end{aligned}$$

Adding this for all j' gives the desired result. $\square$

The above lemma helps us to lower bound the amount of work remaining in OPT's schedule at t. Let $V_{\leq k}(u)$ be the amount of processing time remaining for the set of jobs in $U(\mathcal{A}, u)$ which are of class at most k. In other words $V_{\leq k}(u)$ is shorthand $V_{\leq k}(u, U(\mathcal{A}, u))$. Recall that $P_{\leq k}(U(\mathcal{A}, u))$ is the total processing time of the jobs in $U(\mathcal{A}, u)$.

Lemma 4.5 *For any time u,*

$$V^*_{\leq k}(u, r_j \leq u) \geq \frac{\epsilon}{1+\epsilon} P_{\leq k}(U(\mathcal{A}, u)) + \frac{1}{1+\epsilon} V_{\leq k}(u) - \frac{m2^{k+4}}{1+\epsilon}.$$

PROOF. Let u' be the earliest time such that there is some machine j which is processing jobs of class at most k during the interval $(u', u]$. So, at time u', machine j does not have any job from classes up to k. From Lemma 4.2, at time u', $R^j_{\leq k}(u') \leq 2^{k+3}$ for every machine j. It follows that

$$V_{\leq k}(u') = \sum_j R^j_{\leq k}(u') \leq m2^{k+3}. \tag{7}$$

Note that $u' \leq r_j$ for any job $j \in U(\mathcal{A}, u)$ since j was assigned to some machine at time r_j and this machine, by the nature of the algorithm, cannot process a job of class greater that k while j is unfinished. It follows that $P_{\leq k}(J[u', u]) \geq P_{\leq k}(U(\mathcal{A}, u))$. Now, Lemma 4.4 implies that the machines do at least $(1+\epsilon)m(u - u') - m2^{k+3}$ amount of work on jobs of class at most k during $[u', u]$ in $\mathcal{A}$. Combining this with inequality (7), we get

$$\begin{aligned}
&V_{\leq k}(u) \\
&\leq m2^{k+3} + P_{\leq k}(J[u', u]) - \left((1+\epsilon)m(u - u') - m2^{k+3}\right) \\
&= P_{\leq k}(J[u', u]) + m2^{k+4} - (1+\epsilon)m(u - u'). \qquad (8)
\end{aligned}$$

Now OPT can do at most $m(u - u')$ amount of work during $[u', u]$. Further, it receives jobs of class at most k which have total processing time $P_{\leq k}(J[u', u])$ during this interval. So, it follows that

$$V^*_{\leq k}(u, r_j \leq u) \geq P_{\leq k}(J[u', u]) - m(u - u'). \tag{9}$$

Combining inequalities (8) and (9), we get the desired result. $\square$

We now come back to the proof of inequality (6). Using the expression for $V^*_{\leq k}(r_i, r_j \leq r_i)$ obtained from Lemma 4.5 (by substituting r_i for u) in inequality (6), it is sufficient to show the following:

$$\begin{aligned}
&\frac{\epsilon}{1+\epsilon} P_{\leq k}(U(\mathcal{A}, r_i)) + \frac{V_{\leq k}(r_i)}{1+\epsilon} - \frac{m2^{k+4}}{1+\epsilon} + \\
&\quad P_{\leq k}(J[r_i, t']) - m(t - r_i) \\
&\qquad \geq \epsilon'' \left[m(t - r_i) + P_{\leq k}(U(\mathcal{A}, r_i))\right]. \qquad (10)
\end{aligned}$$

Rearranging terms, we need to show that

$$\begin{aligned}
&\left(\frac{\epsilon}{1+\epsilon} - \epsilon''\right) P_{\leq k}(U(\mathcal{A}, r_i)) + \frac{V_{\leq k}(r_i)}{1+\epsilon} \\
&\qquad + P_{\leq k}(J[r_i, t']) - (1+\epsilon'')m(t - r_i) \geq \frac{m2^{k+4}}{1+\epsilon}.
\end{aligned}$$

Clearly, $V_{\leq k}(r_i) \leq P_{\leq k}(U(\mathcal{A}, r_i))$. So if we assume $\epsilon'' \leq \frac{\epsilon}{1+\epsilon}$, and replace $P_{\leq k}(U(\mathcal{A}, r_i))$ by $V_{\leq k}(r_i)$ it suffices to show

$$(1-\epsilon'')V_{\leq k}(r_i) + P_{\leq k}(J[r_i, t']) - (1+\epsilon'')m(t - r_i) \geq \frac{m2^{k+4}}{1+\epsilon}.$$

Lemma 4.4 implies that $\mathcal{A}$ does at least $(1+\epsilon)m(t' - r_i) - m2^{k+3}$ amount of work during $[r_i, t']$. So, it must be the case that $V_{\leq k}(r_i) + P_{\leq k}(J[r_i, t']) \geq (1+\epsilon)m(t' - r_i) - m2^{k+3}$, because the left hand side is the total volume of jobs of group at most k that need to be processed during $[r_i, t']$. Thus, it is enough to show that

$$\begin{aligned}
&(1-\epsilon'')(1+\epsilon)(t' - r_i) - (1+\epsilon'')(t - r_i) \\
&\qquad \geq \frac{2^{k+4}}{1+\epsilon} + (1-\epsilon'')2^{k+3}. \qquad (11)
\end{aligned}$$

Observing that $t' - r_i = (1-\epsilon')(t - r_i)$, and choosing $\epsilon' = \epsilon'' = \frac{\epsilon}{4(1+\epsilon)}$, we get

$$\begin{aligned}
&\left[(1+3\epsilon/4)\left(\frac{1+3\epsilon/4}{1+\epsilon}\right) - \frac{1+5\epsilon/4}{1+\epsilon}\right](t - r_i) \\
&\qquad \geq \frac{2^{k+4}}{1+\epsilon} + \frac{1+3\epsilon/4}{1+\epsilon} 2^{k+3}.
\end{aligned}$$

Simplifying each side of the equation, we see that it is enough to have

$$\frac{\epsilon}{4}(t - r_i) \geq 3 \cdot 2^{k+3}.$$

The equation above is satisfied whenever $(t - r_i) \geq (3 \cdot 2^{k+5})/\epsilon$. Thus we can set α to be $O(1 + 1/\epsilon)$ as well.

4.2 Extra machines

The analysis for the extra machines is very similar to the one with extra speed. We will simply highlight the modifications needed in the analysis above. Lemmas 4.4 and 4.5 each become slightly weaker, as stated below.

Lemma 4.6 *Consider the schedule produced by $\mathcal{A}$. Suppose a machine remains busy in the interval $[t_1, t_2]$ and processes jobs only from classes 1 to k during this interval. Then the total amount of processing done by all the m machines on jobs of class at most k during $[t_1, t_2]$ is at least $(1+\epsilon)m(t_2 - t_1) - (1+\epsilon)m2^{k+3}$.*

Lemma 4.7 *For any time u,*

$$V^*_{\leq k}(u, r_j \leq u) \geq \frac{\epsilon}{1+\epsilon}P_{\leq k}(U(\mathcal{A}, u)) + \frac{1}{1+\epsilon}V_{\leq k}(u) - m2^{k+4}.$$

Thus Equation 10 can now be rewritten as

$$\begin{aligned} &\frac{\epsilon}{1+\epsilon}P_{\leq k}(U(\mathcal{A}, r_i)) + \frac{V_{\leq k}(r_i)}{1+\epsilon} - \frac{m2^{k+4}}{1+\epsilon} \\ &\quad + P_{\leq k}(J[r_i, t']) - m(t - r_i) \\ &\geq \quad \epsilon'' \left[m(t - r_i) + P_{\leq k}(U(\mathcal{A}, r_i))\right]. \end{aligned} \tag{12}$$

Doing similar manipulations as before, we get

$$\begin{aligned} &(1 - \epsilon'')(1 + \epsilon)(t' - r_i) - (1 + \epsilon'')(t - r_i) \\ &\geq \quad 2^{k+4} + (1 - \epsilon'')2^{k+3}. \end{aligned} \tag{13}$$

Finally, substituting $t' - r_i = (1 - \epsilon')(t - r_i)$, and choosing $\epsilon' = \epsilon'' = \frac{\epsilon}{4(1+\epsilon)}$, we see that it is enough to have

$$\frac{\epsilon}{4(1+\epsilon)}(t - r_i) \geq 3 \cdot 2^{k+3}.$$

The equation above is satisfied whenever $(t - r_i) \geq (3 \cdot 2^{k+5})(1 + \epsilon)/\epsilon$. Thus we can set α to be $O(1 + 1/\epsilon)$ once again.

5. REFERENCES

[1] N. Alon, Y. Azar, G. Woeginger and T. Yadid. Approximation schemes for scheduling. *Proc. of SODA*, 1997.

[2] A. Avidor, Y. Azar and J. Sgall. Ancient and new algorithms for load balancing in the l_p norm. *Algorithmica*, 29(3):422-441, 2001.

[3] B. Awerbuch, Y. Azar, E. Grove, M. Kao, P. Krishnan and J. Vitter. Load balancing in the l_p norm. *Proc. of FOCS*, 1995.

[4] N. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. *Proc. of 15th SPAA*, 2003.

[5] B. Awerbuch, Y. Azar, S. Leonardi and O. Regev. Minimizing flow time without migration. *Proc. of STOC*, 1999.

[6] N. Bansal and M. Harchol-Balter. Analysis of SRPT scheduling : Investigating unfairness. *Proc. of ACM Sigmetrics*, 2001.

[7] N. Bansal and K. Pruhs. Server scheduling in the L_p norm : a rising tide lifts all boats. *Proc. of STOC*, 2003.

[8] N. Bansal and K. Dhamdhere. Minimizing Weighted Flowtime. *Proc. of SODA*, 2003.

[9] L. Becchetti and S. Leonardi. Non-clairvoyant scheduling to minimize average flowtime on single and parallel machines. *Proc. of STOC*, 2001.

[10] L. Becchetti, S. Leonardi, and S. Muthukrishnan. Scheduling to minimize Average Stretch without Migration. *Proc. of SODA*, 2001.

[11] L. Becchetti, S. Leonardi, A. M. Spaccamela, K. Pruhs. Online weighted flow time and deadline scheduling. *Proc. of RANDOM-APPROX*, 2001.

[12] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and Stretch Metrics for Scheduling Continuous Job Streams. *Proc. of SODA*, 1998.

[13] C. Chekuri, S. Khanna. Approximation schemes for preemptive weighted flow time. *Proc. of STOC*, 2002.

[14] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for Minimizing Weighted Flow Time. *Proc. of STOC*, 2001.

[15] M. Crovella, R. Frangioso and M. Harchol-Balter. Connection scheduling in web servers. *USENIX Symposium on Internet Technologies and Systems*, 1999.

[16] L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and indentical parallel machines. *Proc. of ESA*, 1999.

[17] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *JACM*, 47(4):617–43, 2000.

[18] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. *Proc. STOC*, 1997.

[19] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant Scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.

[20] S. Muthukrishnan, R. Rajaraman, R. Shaheen, J. Gerkhe. Online scheduling to minimize average stretch. *Proc. of FOCS*, 1999.

[21] C. Phillips, C. Stein, E. Torng, and J. Wein. Optimal Time Critical Scheduling Via Resource Augmentation. *Proc. of STOC*, 1997.

[22] B. Schroeder and M. Harchol-Balter. Web servers under overload : how scheduling can help. *Proc. of 18th International Teletraffic Congress*, 2003.

APPENDIX

It suffices to prove Lemmas 3.7 and 3.8 for $\alpha = 1$. For brevity, we will use z_k to denote the random variable $z_k\left(\frac{1}{m(1+\delta)}\right)$.

Proof of Lemma 3.7. Let $I(j)$ denote the smallest value of l such that $\sum_{k=1}^{l} x_k/m \geq j$. Define $\mu_j = \sum_{k=1}^{I(j)} x_k/(m(1+\delta))$. Define $X_j = \sum_{k=1}^{I(j)} x_k z_k$. Clearly, $\mathbf{E}[X_j] = \mu_j \geq j/(1+\delta)$.

Consider any $i > 0$. Let $j = \left\lceil \sum_{k=1}^{i} x_k/m \right\rceil$. Clearly $I(j) \geq i$, and hence, $\sum_{k=1}^{I(j)} x_k z_k \geq \sum_{k=1}^{i} x_k z_k$. But

$$\sum_{k=1}^{I(j)} x_k/m \leq 1 + \lceil \sum_{k=1}^{i} x_k/m \rceil \leq 2 + \sum_{k=1}^{i} x_k/m.$$

Therefore,

$$\max_i \left\{ \sum_{k=1}^{i} x_k z_k - \sum_{k=1}^{i} x_k/m \right\} \leq 2 + \max_j \left\{ X_j - \mu_j(1+\delta) \right\}.$$

Hence, it suffices to analyze the random variable

$$\Delta = \max_j \left\{ X_j - \mu_j(1+\delta) \right\}.$$

Using a folklore generalization of the Chernoff bound that works for Bernoulli variables where the i-th variable is 0 with probability q_i and takes an arbitrary value $x_i \leq 1$ with probability $1 - q_i$, we obtain

$$\begin{aligned} \mathbf{Pr}\left[X_j > \mu_j(1+\delta)\right] &\leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \right)^{\mu_j} \\ &\leq \left(\frac{e^{\delta/(1+\delta)}}{1+\delta} \right)^{j} \\ &\leq e^{-j\delta^2/4}, \text{for } 0 \leq \delta \leq 1/4. \end{aligned}$$

The last inequality above can be verified easily by plotting the function in Mathematica, or with a little more effort, by using $e^{\delta} \leq 1 + \delta + \delta^2$ in the range of interest. Hence,

$$\mathbf{Pr}[\Delta > \beta] \leq \beta e^{-\beta\delta^2/4} + \sum_{j \geq \beta} e^{-j\delta^2/4} \leq \left(\beta + \frac{8}{\delta^2} \right) e^{-\beta\delta^2/4}.$$

We will only use the above inequality for $\beta > 8/\delta^2$, and hence, $\mathbf{Pr}[\Delta > \beta] \leq 2\beta e^{-\beta\delta^2/4}$.

But $\mathbf{E}[\Delta] \leq k + \sum_{\beta \geq k} \mathbf{Pr}[\Delta > \beta]$ for all k. Using standard series summations, we obtain $\mathbf{E}[\Delta] \leq k + 2\frac{r^k}{1-r}\left(k + \frac{r}{1-r}\right)$, for any k and where $r = e^{-\delta^2/4}$. Choosing $k = (c/\delta^2)\log(1/\delta)$ for a suitably large constant c, we get $\mathbf{E}[\Delta] = O((1/\delta^2)\log(1/\delta))$, completing the proof of Lemma 3.7. □

PROOF OF LEMMA 3.8. Define

$$\Delta = \max\left\{0, \max_i \left\{ \sum_{j=1}^{i} x_j z_j - (1-\delta/2)\sum_{j=1}^{i} x_j/m \right\}\right\}.$$

By an argument analogous to Lemma 3.7, we obtain

$$\mathbf{E}[\Delta] = O((1/\delta^2)\log(1/\delta)). \tag{14}$$

Define $\Delta' = (2/\delta)(\Delta + y)$. Choose k to be the smallest value such that $\sum_{j=1}^{k} x_j/m > \Delta'$. For any $k' \geq k$,

$$\begin{aligned} &\sum_{j=1}^{k'} x_j z_j - \sum_{j=1}^{k'} x_j/m \\ &= \sum_{j=1}^{k'} x_j z_j - \sum_{j=1}^{k'} (1-\delta/2) x_j/m - (\delta/2)\sum_{j=1}^{k'} x_j/m \\ &\leq \Delta - (\delta/2)(\Delta + y)(2/\delta) \\ &\leq -y \end{aligned}$$

Thus we obtain $X^{(c)}(y) \leq \Delta' + 1$. Combining with equation 14, we obtain

$$\mathbf{E}\left[X^{(c)}(y)\right] = O\left(\frac{y}{\delta} + \frac{1}{\delta^3}\log\frac{1}{\delta}\right).$$

□