DIMACS Technical Report 99-2
January 1999

Scheduling to Minimize Average Stretch

by

Johannes E. Gehrke[1] S. Muthukrishnan[2] Rajmohan Rajaraman[3]
Anthony Shaheen[4]

[1]Department of Computer Science, University of Wisconsin, Madison, WI 53706. Email: johannes@cs.wisc.edu.
[2]AT&T Shannon Laboratories, Florham Park, NJ 07932. Email: muthu@research.att.com.
[3]College of Computer Science, Northeastern University, Boston MA 02115. Email: rraj@ccs.neu.edu. Part of this work was done when the author was at DIMACS.
[4]Loyola Marymount University, Los Angeles, CA 90045. Email: tshaheen@mail.earthlink.net. Part of this work was done when the author was at DIMACS as part of the Research Experiences for Undergraduates program.

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, Bellcore and NEC Research Institute.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

We consider the classical problem of online preemptive job scheduling on uniprocessor and multiprocessor machines. For a given job, we measure the quality of service provided by an algorithm by the *stretch* of the job, which is defined as the ratio of the amount of time that the job spends in the system to the processing time of the job. For a given sequence of jobs, we measure the performance of an algorithm by the *average stretch* achieved by the algorithm over all the jobs in the sequence.

We first prove that no on-line algorithm can achieve a competitive ratio that is smaller than 1.03. The main contribution of this paper is to show that the *shortest remaining processing time* algorithm (SRPT) is $O(1)$-competitive with respect to average stretch for both uniprocessors as well as multiprocessors. For uniprocessors, we prove that SRPT is 2-competitive. We also establish an essentially matching lower bound on the competitve ratio of SRPT on uniprocessors. For multiprocessors, we show that the competitive ratio of SRPT is at most 14.

1 Introduction

We consider the following online scheduling scenario. A sequence of jobs arrives online and the processing time of each job is known at the time of its arrival. Our goal is to produce a schedule to process the continuously arriving stream of such jobs. We focus on the *preemptive* version of scheduling; that is, jobs may be stopped before their completion and resumed later after processing other jobs in the interim. The quality of service we consider is the *stretch* of a job, which is defined as the ratio of the amount of time that the job spends in the system to the processing time of the job. Thus, if an job j with processing time p_j arrives at time r_j and is completed at time C_j, then the stretch of the job is given by $s_j = (C_j - r_j)/p_j$. We seek to minimize the *average stretch*, or equivalently, the total stretch, of all the jobs in a given instance.

We study the problem of minimizing average stretch for both uniprocessor as well as multiprocessor scheduling. In the case of multiprocessor scheduling, we assume that all of the processors are identical. Thus, in the standard 3-field scheduling notation [7], the problems that we study may be listed as $1 \,|\, pmtn, r_j \,|\, \sum s_j$ and $Pm \,|\, pmtn, r_j \,|\, \sum s_j$, respectively.

Since our focus is on online algorithms, we perform a competitive analysis [11] of the algorithms under consideration. An online algorithm $\mathcal{A}$ is *r-competitive* if there exists a constant c such that for every instance $\mathcal{J}$, $S(\mathcal{J})$ is at most $rS^*(\mathcal{J})+c$, where $S(\mathcal{J})$ and $S^*(\mathcal{J})$ represent the total stretch achieved by $\mathcal{A}$ and an optimal offline algorithm, respectively, on instance $\mathcal{J}$. The *competitive ratio* of an algorithm $\mathcal{A}$ is the infimum over all r such that $\mathcal{A}$ is r-competitive.

Much of our work concerns the analysis of the well-known *shortest remaining processing time* algorithm (SRPT). For uniprocessors, SRPT is simply defined as follows: In each time step, process a job with the shortest remaining processing time among all the unfinished jobs. For an m-processor machine each time step consists of the following operation: if there are at least m unfinished jobs, a set of m jobs that have the m shortest remaining processing times are processed, where we break ties arbitrarily; otherwise, every unfinished job is processed. (Since the processors are assumed to be identical, the particular assignment of an unfinished job to a processor at any step is irrelevant [1].)

Our results are threefold:

- We show that there exists no on-line algorithm that minimizes average stretch for every instance; specifically, we show that the competitive ratio of every online algorithm is at least 1.036.
- We show that for any instance $\mathcal{J}$ on a uniprocessor machine, the total stretch $S(\mathcal{J})$ achieved by SRPT is at most $S^*(\mathcal{J}) + |\mathcal{J}|$, thus implying that SRPT is 2-competitive. We also establish an essentially matching lower bound on SRPT for uniprocessors.
- For multiprocessors, we show that for any instance $\mathcal{J}$, the total stretch $S(\mathcal{J})$ achieved

[1] We note that since preemption is allowed, it is possible for a job to be preempted at one processor and later resumed at a different processor.

by SRPT is at most $3S^*(\mathcal{J}) + 11|\mathcal{J}|$, which implies an upper bound of 14 on the competitive ratio.

Our results are the first known results for minimizing average stretch. For several applications, the stretch of a job i may be considered a "fairer" measure than *response time*[2], which is defined as $C_i - r_i$. For example, stretch is the chosen metric for measuring the performance of scheduling algorithms in web servers [2, 4, 5]. This is because large jobs (e.g., large web pages) typically have larger response times, and hence, the metric of average response time tends to assign more weight to large jobs. On the other hand, the stretch of a job normalizes the response time of the job by its processing time and thus measures the quality of service provided to a job in proportion to its demand.

While average response time has been studied extensively in the literature [1, 6, 10], the analytical study of stretch as a performance measure was initiated only recently [2]. All of the results presented in [2], however, concern the metric of *maximum stretch* and do not yield any useful bounds for average stretch. Classical results in scheduling theory [1] show that SRPT is an optimal algorithm with respect to minimizing average response time on uniprocessors. In contrast, we have shown that average stretch is a provably harder measure to optimize online, but not significantly more so since SRPT is shown to be $O(1)$-competitive for both uniprocessors as well as multiprocessors. Our upper bound for SRPT makes it an attractive algorithm for uniprocessor scheduling since it is nearly optimal with respect to average stretch while being truly optimal with respect to average response time. Indeed, recent research on scheduling algorithms for web servers [5] has shown that SRPT outperforms several other scheduling policies currently used in web servers.

Our constant-factor competitiveness result for multiprocessors provides a surprising contrast to the recent lower bound established on the competitive ratio of any online algorithm with respect to average response time. It is shown in [8] that the best competitive ratio achievable for average response time on an m-processor machine is $\Omega(\max\{\log P, \log(n/m)\})$, where P is the ratio of the maximum to the minimum processing time of any job and n is the number of jobs. Thus, the average response time and average stretch metrics have the following interesting contrasting characteristics. For uniprocessors, while average response time can be optimized by an on-line algorithm, the best competitive ratio achievable for average stretch is strictly greater than 1. On the other hand, for multiprocessors, while a constant-factor competitive online algorithm exists for average stretch, the best competitive ratio for average response time grows logarithmically with the number of jobs. In [8], it is also shown that the competitive ratio of SRPT with respect to average response time on multiprocessors is optimal to within a constant factor. This indicates that SRPT is a compelling algorithm for multiprocessor scheduling as well since it matches the best competitive ratio up to contant factors for both average stretch as well as average response time.

A measure closely related to average stretch is *average weighted completion time*, which equals $\sum_i w_i C_i$, where w_i is the prespecified weight of job i. Note that the weighted completion time and total stretch are intimately related; any schedule that optimizes weighted

[2]Response time is also sometimes referred to as *flow time* in the scheduling literature.

completion time with $w_i = 1/p_i$ also optimizes total stretch. From the point of view of approximation, however, the two metrics are different. The best known online algorithm for minimizing average completion time is a randomized 4/3-competitive algorithm of [9] (also see [3] for more results in this area); this result, however, does not yield any useful upper bound on the competitive ratio for average stretch.

Our analysis of SRPT relies on a careful comparison of the queue of unfinished jobs in SRPT with the corresponding queue associated with any other scheduling algorithm at every time step. In Sections 3 and 4, we describe our analysis for uniprocessors and multiprocessors, respectively. In Section 2, we establish the lower bound. The lower bound relies on instances that require at most three different job sizes. In Section 5, we complement this lower bound by showing that there exists a simple optimal online algorithm for the case in which there are at most two job sizes.

2 Lower bounds

In this section, we establish a lower bound on the competitive ratio of any online uniprocessor scheduling algorithm with respect to average stretch.

Theorem 1 *The competitive ratio of any on-line algorithm is at least* 1.025.

Proof: Consider three jobs j_1, j_2, and j_3, with release times 0, 4, and 5, respectively, and processing times 5, 2, and 1, respectively. Let instance $\mathcal{J}_1$ be the set $\{j_1, j_2\}$ and instance $\mathcal{J}_2$ be the set $\{j_1, j_2, j_3\}$.

Given $\mathcal{J}_1$, the optimal solution is to process j_1 for 4 steps, then process j_2 for 2 steps and then complete j_1. The optimal total stretch is $1 + 7/5 = 2.4$. Given $\mathcal{J}_2$, the optimal solution is to process j_1 for 5 steps and thus complete it, then process j_3 for 1 step, and finally, process j_2. The optimal stretch value for this instance is 4.

Given instance $\mathcal{J}_1$, if the given on-line algorithm completes j_1 before j_2, then its stretch is at least 2.5, implying a competitive ratio of at least 1.04. If, on the other hand, the algorithm preempts j_1 at time 4 (i.e., at the start of the fifth step), then the two best scenarios for the algorithm is to finish the jobs in one of the following orders: (i) j_2, j_3, j_1 or (ii) j_3, j_2, j_1. The stretch for case (i) is 4.6 and for case (ii) is 4.1. Therefore, the competitive ratio is at least 1.025. ■

Remark. Consider the input in which a job of size 40 arrives at time 0 and one of size 20 arrives at the end of time step 28. If the algorithm preempts the first job when the second job arrives, then a third job is presented at the end of time step 40 and is of size 10. We can argue, in the same way as in the proof above, that no algorithm can have a competitive ratio at least 1.036. ■

3 Analysis of SRPT for uniprocessor scheduling

In this section, we analyze the competitiveness of SRPT with respect to average stretch. For job j, let r_j and p_j denote the release time and processing time, respectively, of j. Recall that the response time of a given job is the difference between the time it completes execution and the release time of the job and the stretch of a job is the ratio of the response time of the job to the processing time of the job. The *total stretch* of a scheduling algorithm for a given instance $\mathcal{J}$ is the sum over all jobs $j \in \mathcal{J}$ of the stretch of j under the scheduling algorithm. Let $S(\mathcal{J})$ denote the total stretch of SRPT for instance $\mathcal{J}$. The main result of this section is the following theorem.

Theorem 2 *Let $\mathcal{J}$ be a given scheduling instance and $S^*(\mathcal{J})$ be the total stretch achieved by the optimal schedule for $\mathcal{J}$. Then $S(\mathcal{J})$ is at most $S^*(\mathcal{J}) + |\mathcal{J}|$.*

We prove an essentially matching lower bound at the end of this section. In terms of competitive ratio, Theorem 2 implies the following result.

Corollary 2.1 SRPT *is 2-competitive with respect to average stretch.*

Proof: For any scheduling algorithm, the stretch of any job is at least 1. Therefore $S^*(\mathcal{J})$ is at least $|\mathcal{J}|$. It thus follows from Theorem 2 that $S(\mathcal{J}) \leq 2S^*(\mathcal{J})$. ■

We now introduce some notation to characterize the execution of SRPT on a given instance $\mathcal{J}$. For any job j in $\mathcal{J}$ and time $t \geq r_j$, let $\rho_t(j)$ denote the remaining processing time of job j at time t under SRPT. For any time t, let Q_t denote the set of all jobs in $\mathcal{J}$ that have been released at some time less than or equal to t and have not been completed by SRPT; that is, Q_t is the set of all jobs j for which $r_j \leq t$ and $\rho_t(j) > 0$.

We begin the proof of Theorem 2 by providing an alternative expression for the total stretch S.

$$S(\mathcal{J}) = \sum_{j \in \mathcal{J}} \sum_{t : j \in Q_t} \frac{1}{p_j} = \sum_t \sum_{j \in Q_t} \frac{1}{p_j}.$$

Let α_t denote the sum of the reciprocals of the processing times of the jobs in Q_t. Thus, $S(\mathcal{J}) = \sum_t \alpha_t$. Let β_t denote the sum of the reciprocals of the remaining processing times of the jobs in Q_t. We rank all of the jobs in Q_t in nondecreasing order of their remaining processing times, breaking ties according to an arbitrary ordering of the job IDs whenever necessary. For $i \geq 1$, let $q_{t,i}$ denote the job of rank i in Q_t; if the job of rank i is not defined, we set $q_{t,i}$ to be a job with processing time ∞. Let U_t denote the set of jobs ranked greater than 1.

We split the sum $S(\mathcal{J})$ into two parts $S_1(\mathcal{J})$ and $S_2(\mathcal{J})$ as follows:

$$\begin{aligned} S_1(\mathcal{J}) &= \sum_t \frac{1}{p_{q_{t,1}}} \text{ and} \\ S_2(\mathcal{J}) &= \sum_t \sum_{j \in U_t} \frac{1}{p_j}. \end{aligned}$$

We establish Theorem 2 by separately placing upper bounds on $S_1(\mathcal{J})$ and $S_2(\mathcal{J})$.

Lemma 3.1 *For any instance $\mathcal{J}$, $S_1(\mathcal{J}) = |\mathcal{J}|$.*

Proof: A job j contributes $1/p_j$ to the term $S_1(\mathcal{J})$ whenever j is ranked 1 in the queue. Since j is executed exactly when it is the ranked 1 in the queue, there are exactly p_j times when j is ranked 1 in the queue. Thus, j contributes $p_j/p_j = 1$ to $S_1(\mathcal{J})$. We thus obtain that $S_1(\mathcal{J})$ equals $|\mathcal{J}|$. ■

We place an upper bound on $S_2(\mathcal{J})$ by comparing the performance of SRPT with that of an arbitrary scheduling algorithm $\mathcal{A}$. Just as we have characterized the execution of SRPT for a given instance $\mathcal{J}$ using the variables ρ_t, Q_t, and q_t, we can characterize the execution of $\mathcal{A}$ for $\mathcal{J}$. Let $\widetilde{Q}_t$, $\widetilde{\rho}_t$, $\widetilde{\alpha}_t$, and $\widetilde{\beta}_t$ denote the equivalent of Q_t, ρ_t, α_t, and β_t, respectively, for the execution of $\mathcal{A}$. Furthermore, whenever we henceforth introduce a new notation for a variable associated with SRPT, it is implicit that the "tilde" version of the notation denotes the corresponding variable for $\mathcal{A}$.

We place an upper bound on $S_2(\mathcal{J})$ by establishing the following upper bound on α_t.

Lemma 3.2 *For any time t, we have*

$$\beta_t \leq \widetilde{\alpha}_t + \frac{1}{\rho_t(q_{t,1})}.$$

Corollary 3.2.1 *For any instance $\mathcal{J}$, $S_2(\mathcal{J}) \leq S^*(\mathcal{J})$.*

Proof: Since $p_j \geq \rho_t(j)$ for any job j, $\alpha_t \leq \beta_t$. Thus, we obtain the following inequality by invoking Lemma 3.2.

$$S_2(\mathcal{J}) = \sum_t \left(\alpha_t - \frac{1}{\rho_t(q_{t,1})}\right) \leq \sum_t \left(\beta_t - \frac{1}{\rho_t(q_{t,1})}\right) \leq \sum_t \widetilde{\alpha}_t \leq \widetilde{S}(\mathcal{J})$$

Since the above inequality holds for every schedule $\mathcal{A}$, the desired claim follows. ■

We prove Lemma 3.2 by carefully comparing SRPT's queue Q_t with $\mathcal{A}$'s queue $\widetilde{Q}_t$ at each time step t. For any time t and positive integer i, let $P_t(i)$ denote the number of jobs in Q_t with remaining processing time at most i. For any time t and positive integer i, let $V_t(i)$ denote the sum of the remaining processing times of all jobs in $P_t(i)$. We call $V_t(i)$ the *volume* of jobs in Q_t with remaining processing time at most i.

Lemma 3.3 *For any time t and any positive integer i, $V_t(i)$ is at most $\widetilde{V}_t(i) + i$.*

Proof: The proof is by induction on t. The induction basis trivially holds since for all i, $V_0(i) = \widetilde{V}_0(i)$. For the induction hypothesis, we assume that claim is true at the end of step $t-1$. We now establish the induction step for time step t. Let i be any positive integer. We first note that the release of new jobs at the start of step does not change $V_t(i) - \widetilde{V}_t(i)$ for any i. Let X denote the union of the set $P_{t-1}(i)$ and the set of jobs with processing time at most i that are released at time t.

We consider two cases depending on whether X is empty. In the case where X is nonempty, SRPT executes one job from X. Since $\mathcal{A}$ performs at most one unit of work, we

can invoke the induction hypothesis and obtain that $V_t(i) \leq \widetilde{V}_t(i) + i$. We now consider the case where X is empty. In this case, at most one job in Q_t can have remaining processing time at most i at the end of step t. Therefore, we have $V_t(i) \leq i \leq \widetilde{V}_t(i) + i$. This completes the proof of the desired claim. ■

Lemma 3.4 *For any time t and any integer $k > 1$, we have $\rho_t(q_{t,k}) \geq \widetilde{\rho}_t(\widetilde{q}_{t,k-1})$.*

Proof: We first prove that for all $k > 1$, $\rho_t(q_{t,k}) \geq \widetilde{\rho}_t(\widetilde{q}_{t,k-1})$. The proof is by contradiction. Let, if possible, $k > 1$ be the smallest integer such that $\rho_t(q_{t,k}) < \widetilde{\rho}_t(\widetilde{q}_{t,k-1})$. Let b denote $\rho_t(q_{t,k})$. It follows that $|P_t(b)| \geq k$, while $|P_t(b)| = k - 2$. We thus have the following inequality.

$$\begin{aligned} V_t(b) &= \sum_{1 \leq i \leq k} \rho_t(q_{t,i}) \\ &= \rho_t(q_{t,1}) + \rho_t(q_{t,k}) + \sum_{2 \leq i < k} \rho_t(q_{t,i}) \\ &\geq \rho_t(q_{t,1}) + b + \sum_{1 \leq i < k-1} \widetilde{\rho}_t(\widetilde{q}_{t,i}) \\ &= \rho_t(q_{t,1}) + b + \widetilde{V}_t(b) \\ &> b + \widetilde{V}_t(b), \end{aligned}$$

thus contradicting Lemma 3.3. (In the third step, we use the assumption that for $1 < i < k$, $\rho_t(q_{t,i}) \leq \widetilde{\rho}_t(\widetilde{q}_{t,i-1})$. In the final step, we use the fact that $\rho_t(q_{t,1})$ is positive.) ■

Corollary 3.4.1 *For any time t, we have*

$$\beta_t \leq \widetilde{\beta}_t + \frac{1}{\rho_t(q_{t,1})}.$$

Proof: The desired claim follows from Lemma 3.4.

$$\begin{aligned} \beta_t - \widetilde{\beta}_t &= \sum_{k \geq 1} \left(\frac{1}{\rho_t(q_{t,k})} - \frac{1}{\widetilde{\rho}_t(\widetilde{q}_{t,k})} \right) \\ &\leq \frac{1}{\rho_t(q_{t,1})} + \sum_{k \geq 2} \left(\frac{1}{\rho_t(q_{t,k})} - \frac{1}{\widetilde{\rho}_t(\widetilde{q}_{t,k-1})} \right) \\ &\leq \frac{1}{\rho_t(q_{t,1})}, \end{aligned}$$

since $\rho_t(q_{t,k}) \geq \widetilde{\rho}_t(\widetilde{q}_{t,k-1})$ for all $k \geq 2$. ■

We are now ready to prove Lemma 3.2.

Proof of Lemma 3.2: Consider an arbitrary time step t. Let $\mathcal{B}$ be a scheduling algorithm that minimizes the sum of the reciprocals of the processing times of the jobs in the queue at time t. Let α'_t denote the sum of the reciprocals of the processing times of the jobs in $\mathcal{B}$'s queue at time t.

We now argue that $\mathcal{B}$ can be chosen such that for every job j in $\mathcal{B}$'s queue, the remaining processing time at time t is same as the processing time p_j. If not, we can define a schedule $\mathcal{C}$ that mimics $\mathcal{B}$ except that $\mathcal{C}$ never processes any of the jobs that remain in $\mathcal{B}$'s queue at time t. Clearly, $\mathcal{C}$'s queue at time t contains exactly the same jobs as $\mathcal{B}$. Moreover, none of the jobs in $\mathcal{C}$'s queue at time t have been processed prior to time t. Since the remaining processing time of every job in $\mathcal{C}$'s queue is the same as the actual processing time, we obtain the desired inequality from Lemma 3.4.1.

$$\beta_t \leq \alpha'_t + \frac{1}{\rho_t(q_{t,1})} \leq \widetilde{\alpha}_t + \frac{1}{\rho_t(q_{t,1})}.$$

■

Theorem 2 directly follows from Lemma 3.1 and Corollary 3.2.1. We now prove a lower bound on SRPT that essentially matches the upper bound of Theorem 2.

Theorem 3 *For any real $\varepsilon > 0$ and any positive integer $n \geq 3$, there exists an instance $\mathcal{J}$ with n jobs such that $S(\mathcal{J})$ is at least $S^*(\mathcal{J}) + n - 1 - \varepsilon$.*

Proof: We consider jobs of three sizes ℓ_1, ℓ_2, and ℓ_3. We have $\ell_1 \gg \ell_2$, and $\ell_2 = \ell_3 + 2$. Consider the instance in which a job of size ℓ_1 arrives at time 0, followed by a job of size ℓ_2 at time $\ell_1 - \ell_2 + 1$, followed by $n-2$ jobs of size ℓ_3 at times $\ell_1 + 1$, $\ell_1 + \ell_3 + 1$, $\ell_1 + 2\ell_3 + 1$, $\ldots$, $\ell_1 + (n-3)\ell_3 + 1$.

The algorithm SRPT completes the job of size ℓ_1 at time ℓ_1 and then processes one unit of the job of size ℓ_2 before preempting it in favor of the first job of size ℓ_3. The job of size ℓ_2 is completed only after the $n-2$ jobs of size ℓ_3 are completed. Therefore, the total stretch achieved by SRPT equals $n - 1 + ((n-2)\ell_3 + \ell_2 + \ell_2 - 1)/\ell_2 = 2n - 2n/\ell_2 - 1 + 3/\ell_2$. For any $\varepsilon > 0$, we can select ℓ_2 to be sufficiently large such that the total stretch achieved by SRPT is at least $2n - 1 - \varepsilon/2$.

An alternative schedule is the following. Process the size-ℓ_1 job until the size-ℓ_2 job arrives. Then complete the size-ℓ_2 job, followed by the n size-ℓ_2 jobs. Finally, complete the size-ℓ_1 job. The total stretch of this schedule is $n + ((n-1)\ell_3 + 2)/\ell_1$. Given any ℓ_2 and any $\varepsilon > 0$, we can select ℓ_1 to be sufficiently large so that the total stretch achieved in the alternative schedule is at most $n + \varepsilon/2$. Thus, we have $S(\mathcal{J}) \geq S^*(\mathcal{J}) + n - 1 - \varepsilon$. ■

We note that the proof of Theorem 3 uses three job sizes. If we assume that SRPT does not discriminate while breaking ties when there is more than one job with the shortest remaining processing time, then we can establish the same lower bound using only two job sizes.

4 Analysis of SRPT for multiprocessor scheduling

In this section, we analyze SRPT on am m-processor machine and show that it is $O(1)$-competitive with respect to average stretch for any m. A crucial part of our proof for uniprocessors in Section 3 is the property we establish in Lemma 3.4 about SRPT's queue

at any time t. We show that given any scheduling algorithm $\mathcal{A}$ and any time t, we can map each job j in SRPT's queue except the one with the smallest remaining processing time to a unique job in $\mathcal{A}$'s queue at time t that has remaining processing time at most that of j. For multiprocessors, however, a similar claim does not hold. In fact, even for a two-processor system, one can construct an instance for every positive integer k that satisfies the following property: there exists a time step t when the number of jobs in SRPT's queue at time t is k while the queue in the optimal schedule at time t is empty. (A nice example of such an instance is given in [8].)

As in the case of uniprocessor scheduling, we rank all of the jobs in Q_t in nondecreasing order of their remaining processing times, breaking ties according to an arbitrary ordering of the job IDs whenever necessary. As before, for $i \geq 1$, let $q_{t,i}$ denote the job of rank i in Q_t. (If the job of rank i is not defined, we set $q_{t,i}$ to be a job with processing time ∞.) Let F_t denote the set of jobs in Q_t ranked at most m. Let U_t denote the set $Q_t \setminus F_t$.

Analogous to the uniprocessor case, we split the sum $S(\mathcal{J})$ into two parts $S_1(\mathcal{J})$ and $S_2(\mathcal{J})$ as follows.

$$\begin{aligned} S_1(\mathcal{J}) &= \sum_t \sum_{j \in F_t} \frac{1}{p_j} \text{ and} \\ S_2(\mathcal{J}) &= \sum_t \sum_{j \in U_t} \frac{1}{p_j}. \end{aligned}$$

By invoking the same argument that is used in the proof of Lemma 3.1, we show that $S_1(\mathcal{J})$ equals $|\mathcal{J}|$.

Lemma 4.1 *For any instance $\mathcal{J}$, $S_1(\mathcal{J}) = |\mathcal{J}|$.*

Proof: Since each job in F_t is processed at time t, a job j is in F_t for exactly p_j steps. The desired claim follows. ■

We now consider the jobs in U_t. For any time t and positive integer i, let $P_t(i)$ denote the set of jobs in Q_t with remaining processing time at most i. For any time t and positive integer i, let $V_t(i)$ denote the sum of the remaining processing times of all jobs in $P_t(i)$. We define the *volume* of any subset X of Q_t as the sum of the remaining processing time of the jobs in X at time t.

Given any schedule $\mathcal{A}$ and any time t, we define a partial mapping f_t from Q_t to $\tilde{Q}_t$ that satisfies properties P1 and P2 defined below. (As in Section 3 we follow the notational convention that the "tilde" version of a variable defined for SRPT denotes the corresponding variable for $\mathcal{A}$.) Let MPD_t (resp., UMPD_t) denote the set of jobs in Q_t that are mapped (resp., unmapped) by f_t.

(P1) For each job j in MPD_t, $\tilde{\rho}_t(f_t(j)) \leq \rho_t(j)$.

(P2) For any $i > 0$, the total volume of the jobs in $\text{UMPD}_t \cap P_t(i)$ is at most mi.

The following lemma is a generalization of Lemma 3.3.

Lemma 4.2 *For any time t and any positive integer i, $V_t(i)$ is at most $\widetilde{V}_t(i) + mi$.*

Proof: The proof is by induction on t. The induction basis trivially holds since for all i, $V_0(i) = \widetilde{V}_0(i)$. For the induction hypothesis, we assume that claim is true at the end of step $t-1$.

We now establish the induction step for time step t. Let i be any positive integer. We first note that the release of new jobs at the start of step does not change $V_t(i) - \widetilde{V}_t(i)$ for any i. Let X denote the union of the set $P_{t-1}(i)$ and the set of jobs with processing time at most i that are released at time t.

We consider two cases depending on whether $|X|$ is at least m. In the case where $|X|$ is at least m, all of the jobs processed by SRPT are in X. Since $\mathcal{A}$ can perform at most m units of work, we invoke the induction hypothesis and obtain that $V_t(i) \leq \widetilde{V}_t(i) - mi$. We now consider the case where $|X|$ is less than m. In this case, the number of jobs not in X that are processed in step t is at most $m - |X|$. Therefore, at most $m - |X|$ jobs outside of X may have remaining processing time at most i after time step t. We thus obtain that $V_t(i) \leq mi \leq \widetilde{V}_t(i) + mi$. ■

Lemma 4.3 *For any schedule $\mathcal{A}$ and any time t, there exists an injective function f_t from Q_t to $\widetilde{Q}_t$ that satisfies properties P1 and P2.*

Proof: We describe a procedure for defining the function f_t. This procedure runs in phases; in phase i, $i > 0$, we map a subset of the jobs in Q_t with remaining processing time i to jobs in $\widetilde{P}_t(i)$. While defining the mapping, we ensure that for every i, the following two properties hold at the start of phase i: (i) for $0 < j < i$, the total volume of the unmapped jobs in $P_t(j)$ is at most mj, and (ii) for each mapped job j in Q_t, $f_t(j)$ $\widetilde{\rho}_t(f_t(j)) \leq \rho_t(j)$. Clearly, if the two properties hold at the end of the procedure when all of the jobs in Q_t have been considered, then the desired claim follows.

Initially, when $i = 1$, properties (i) and (ii) vacuously hold. We now assume that properties (i) and (ii) hold at the start of phase i and consider the effect of phase i. Let X denote the set of jobs in $P_t(i)$ with remaining processing time equal to i. Let Y denote the set of jobs in $\widetilde{P}_t(i)$ to which no job in Q_t has been mapped at the start of phase i. In phase i, we map as many jobs in X to jobs in Y as possible. Clearly, property (ii) holds at the start of phase $i+1$.

To prove property (i), we consider two cases. The first case is when every job in $\widetilde{P}_t(i)$ is the image of some job after phase i. By property (ii), each job in MPD_t is mapped to a job in $\widetilde{Q}_t$ with no larger remaining processing time. Therefore, it follows from Lemma 4.2 that the volume of unmapped jobs in $P_t(i)$ is at most mi. The second case is when there exists a job in $\widetilde{P}_t(i)$ to which no job is mapped at the end of phase i. In this case, we note that every job in Q_t with remaining processing time equal to i is mapped in phase i. Therefore, we invoke property (i) of the induction hypothesis to obtain that the total volume of the jobs in $P_t(i)$ that are unmapped at the end of phase i is at most $m(i-1) \leq mi$. This completes the proof of the lemma. ■

Properties P1 and P2 help in placing upper bounds on the contribution of the mapped jobs and unmapped jobs, respectively, to $S_2(\mathcal{J})$. While Lemma 4.3 holds for any schedule

$\mathcal{A}$, the particular mapping f_t that we employ for the remainder of this proof is with respect a schedule $\mathcal{B}$ that minimizes the sum of the reciprocals of the processing times of the jobs in the queue at time t. Let α'_t denote the sum of the reciprocals of the processing times of the jobs in $\mathcal{B}$'s queue at time t. As in Lemma 3.2 we can assume without loss of generality that none of the jobs that belong to $\mathcal{B}$'s queue at time t have been processed until time t. Therefore, for each job j in $\mathcal{B}$'s queue at time t, the remaining processing time equals p_j. By property P1 of Lemma 4.3, we know that $\rho_t(j) \geq p_{f_t(j)}$. Since f is injective, we obtain the following lemma about the mapped jobs in Q_t.

Lemma 4.4 *For any time t, we have:*

$$\sum_{j \in \text{MPD}_t} \frac{1}{\rho_t(j)} \leq \alpha'_t.$$

■

We now consider the unmapped jobs. In the following lemma, we invoke a technical inequality established in Appendix A.

Lemma 4.5 *For any positive integer i and any step t, we have:*

$$\sum_{j \in \text{UMPD}_t, \rho_t(j) \geq i} \frac{1}{\rho_t(j)} \leq \frac{2m}{i}.$$

Proof: Let X denote the set $\{j \in \text{UMPD}_t : \rho_t(j) \geq i\}$. We invoke Corollary A.1.1 to prove the desired claim. We note that condition C' of Corollary A.1.1 holds due to property P2 of Lemma 4.2. The claim of Corollary A.1.1 is the desired inequality. ■

We now introduce some additional definition that we need to further classify unmapped jobs. A job j is *marked* at time t if $j \in Q_t$ and the number of mapped jobs in $P_t(\rho_t(j))$ is at most m. We call the first time when a job gets marked as the *marking time* of the job; the marking time of a job that is never marked may be set to ∞. We call an unmapped job j *self-reliant* at time t, if j is marked at time t and the number of jobs in F_t that have release time at least the marking time of j is at most $m/2$. An unmapped job that is not self-reliant at time t is said to be *dependent*.

We partition the unmapped jobs in U_t into three sets: (i) SR_t, the set of self-reliant jobs at time t, (ii) MKD_t, the set of dependent jobs that are marked at time t, and (iii) UMKD_t, the set of dependent jobs that are not marked at time t. Each job j in UMPD_t contributes p_j to $S_2(\mathcal{J})$ at time t. At a high level, we account for the the contributions of the jobs in the three sets SR_t, MKD_t, and UMKD_t, in the following manner. In Lemma 4.6, we show that any job j is self-reliant during $O(p_j)$ steps only. This implies that the total contribution of the self-reliant jobs during the execution of SRPT is $O(\mathcal{J}|)$. Next, in Lemma 4.7, we show that the total contribution of the marked dependent jobs is within a constant factor of the contribution of the jobs in F_t, which we have already shown in Lemma 4.1 to be equal to $|\mathcal{J}|$. Finally, we show in Lemma 4.8 that the contribution of dependent jobs is within a constant

factor of the contribution of the mapped jobs, for which we have placed an upper bound in Lemma 4.4.

We now consider self-reliant jobs.

Lemma 4.6 *Any job j is self-reliant during at most $4p_j$ time steps.*

Proof: Since any job j can be self-reliant only after it has been marked, we only need to consider time steps beginning the marking time of j. Let t denote the marking time of j. Let X denote the set of jobs in Q_t that have rank less than that of j. Since j is marked at time t, it follows that the number of mapped jobs in X is at most m. Moreover, by property P2 of Lemma 4.3, the volume of all of the unmapped jobs in X at time t is at most $m\rho_t(j) \leq mp_j$. Therefore, the volume of of all the jobs in X is at most $2mp_j$.

Consider any time step $t' \geq t$ when j is self-reliant. It follows from the definition of self-reliance that there are at most $m/2$ jobs in $F_{t'}$ that were released after t. Since j is not in $F_{t'}$, it follows that there are at least $m/2$ jobs in X that are in $F_{t'}$. Thus, the volume of the jobs in X decreases by at least $m/2$ in time step t_1. Once all of the jobs in X are completed, j can be never be self-reliant. Since the total volume of X at time t is at most $2mp_j$, the number of time steps when j is self-reliant is at most $4p_j$. ■

We next consider the dependent jobs that are currently marked.

Lemma 4.7 *For any time t, we have:*

$$\sum_{j \in \text{MKD}_t} \frac{1}{p_j} \leq \sum_{j \in F_t} \frac{6}{p_j}.$$

Proof: Consider a job j in F_t. Let $X(j)$ be the set of jobs in MKD_t that were marked before the release time of j. Since every job in $X(j)$ has rank greater than m and hence greater than that of j, it follows that at the time of the release of j, the remaining processing time of every job in $X(j)$ is at least p_j; that is, for any j_1 in $X(j)$, we have $\rho_{r_j}(j_1) \geq p_j$. Moreover, if $Y(j)$ denotes the set of jobs in $X(j)$ that have been processed at any time during the time interval $[r_j, t]$, then $|Y(j)|$ is at most $m-1$. Thus, for every job j_1 in $X(j) \setminus Y(j)$, we have $\rho_t(j_1) = \rho_{r_j}(j_1) \geq p_j$.

Since $X(j)$ is a subset of MKD_t, it follows from the definition of MKD_t that each job in $X(j)$ is unmapped. Therefore, by Lemma 4.5, we have the following inequality.

$$\sum_{j_1 \in X(j) \setminus Y(j)} \frac{1}{p_{j_1}} \leq \frac{2m}{p_j} \tag{1}$$

We now consider the jobs in $Y(j)$. Since there are at most $m-1$ jobs in $Y(j)$, each with processing time at least p_j, we have the following inequality.

$$\sum_{j_1 \in Y(j)} \frac{1}{p_{j_1}} \leq \frac{m-1}{p_j} \tag{2}$$

Equations 1 and 2 together yield the following inequality for each job j in F_t.

$$\sum_{j_1 \in X(j)} \frac{1}{p_{j_1}} \leq \frac{3m}{p_j} \tag{3}$$

Now consider a job j_1 in MKD_t. Since j_1 is not self-reliant, there are at least $m/2$ jobs j in F_t such that j_1 is in $X(j)$. Thus, if we add up both the left-hand and right-hand sides of Equation 3 over all jobs in F_t, j_1 appears at least $m/2$ times on the left-hand side. Therefore, we obtain the following inequality, that establishes the desired claim.

$$\sum_{j_1 \in \text{MKD}_t} \frac{1}{p_{j_1}} \leq \sum_{j \in F_t} \frac{6m}{p_j}$$

■

We finally consider unmarked dependent jobs.

Lemma 4.8 *For any time t, we have:*

$$\sum_{j \in \text{UMKD}_t} \frac{1}{p_j} \leq \sum_{j \in \text{MPD}_t} \frac{2}{\rho_t(j)}.$$

Proof: For any mapped job j in Q_t, let $X(j)$ denote the set of dependent jobs in Q_t that are unmarked and have rank greater than that of j. Since each job in $X(j)$ is unmapped, the following inequality follows from Lemma 4.5.

$$\sum_{j_1 \in X(j)} \frac{1}{\rho_t(j_1)} \leq \frac{2m}{\rho_t(j)} \tag{4}$$

By definition, for each dependent job j_1 that is not marked, there exist at least m jobs j such that j_1 is in $X(j)$. Therefore, by adding Equation 4 over all the mapped jobs, we obtain the following inequality.

$$\begin{aligned} \sum_{j_1 \in \text{UMKD}_t} \frac{1}{p_{j_1}} &\leq \sum_{j_1 \in \text{UMKD}_t} \frac{1}{\rho_t(j_1)} \\ &\leq \sum_{j \in \text{MPD}_t} \frac{2}{\rho_t(j)}. \end{aligned}$$

■

Putting Lemmas 4.4, 4.6, 4.7, and 4.8 together, we obtain the following main lemma.

Lemma 4.9 *For any instance $\mathcal{J}$, we have:*

$$\sum_t \sum_{j \in U_t} \frac{1}{p_j} \leq 3S^*(\mathcal{J}) + 10|\mathcal{J}|.$$

Proof: We establish the desired inequality as follows.

$$\begin{aligned}
\sum_t \sum_{j \in U_t} \frac{1}{p_j} &\leq \sum_t \left(\sum_{j \in \mathrm{MPD}_t} \frac{1}{p_j} + \sum_{j \in \mathrm{SR}_t} \frac{1}{p_j} + \sum_{j \in \mathrm{MKD}_t} \frac{1}{p_j} + \sum_{j \in \mathrm{UMKD}_t} \frac{1}{p_j} \right) \\
&\leq \sum_t \sum_{j \in \mathrm{MPD}_t} \frac{3}{\rho_t(j)} + \sum_j \sum_{t : j \in \mathrm{SR}_t} \frac{1}{p_j} + \sum_t \sum_{j \in F_t} \frac{6}{p_j} \\
&\leq \sum_t \alpha'_t + \sum_t \sum_{j \in F_t} \frac{10}{p_j} \\
&\leq 3S^*(\mathcal{J}) + 10|\mathcal{J}|
\end{aligned}$$

(In the second step, we invoke Lemmas 4.7 and 4.8. In the third step, we invoke Lemmas 4.4 and 4.6. The final step follows from Lemma 4.1.) ■

The main theorem now directly follows from Lemmas 4.1 and 4.9.

Theorem 4 *For any instance $\mathcal{J}$ on multiprocessors, $S(\mathcal{J})$ is at most $3S^*(\mathcal{J}) + 11|\mathcal{J}|$.* ■

Corollary 4.1 *For multiprocessors,* SRPT *is 14-competitive with respect to average stretch.* ■

5 Uniprocessor scheduling with two job sizes

Recall that one of the instances used in the proof of Theorem 1 requires three different job sizes. We now complement this lower bound by showing that there exists an optimal online algorithm for the case where there are only two job sizes.

Theorem 5 *Consider the case when all jobs are either* large *(size p_L) or* small *(size p_S) respectively. There exists an optimal online algorithm for minimizing the average stretch.*

Proof: We first observe that there is an optimal schedule in which all large (respectively, small) jobs are processed in First-in-First-out (FIFO) order. Thus at any time t, there is at most one partly processed job of either category, and that is the job at the head of the FIFO order in that category at that time; say $\rho_t(L)$ is the remaining processing time of that job in the large category, and $\rho_t(S)$ that in the small category. Say $r(L)$ and $r(S)$ are the arrival times of these two jobs respectively. We need to decide which one of these two jobs must be processed. If the large job is processed fully first followed by the small job, the average stretch is

$$\frac{t + \rho_t(L) - r(L)}{p_L} + \frac{t + \rho_t(L) + \rho_t(S) - r(S)}{p_S},$$

and if the small job is processed fully first before the large job, the average stretch is

$$\frac{t + \rho_t(S) - r(S)}{p_S} + \frac{t + \rho_t(L) + \rho_t(S) - r(L)}{p_L}.$$

Arrivals of large or small jobs in the interim does not affect $\rho_t(L)$ or $\rho_t(S)$. The former is desirable when

$$\frac{t+\rho_t(L)-r(L)}{p_L}+\frac{t+\rho_t(L)+\rho_t(S)-r(S)}{p_S} \leq \frac{t+\rho_t(S)-r(S)}{p_S}+\frac{t+\rho_t(L)+\rho_t(S)-r(L)}{p_L}$$

That is,

$$\frac{\rho_t(L)}{p_S} \leq \frac{\rho_t(S)}{p_L},$$

that is, $\rho_t(L)p_L \leq \rho_t(S)p_S$. Therefore, the optimal algorithm is one that in each time step t processes a job i with the smallest $\rho_t(i)p_i$. ■

If we have three job sizes, the algorithm above is no longer optimal; in fact, our lower bound shows that no algorithm is competitive. The lower bound example can be strengthened to show that the competitive ratio of the algorithm in the proof of Theorem 5 is at least 1.5.

References

[1] K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.

[2] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–279, January 1998.

[3] L. Hall, A. Schulz, D. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

[4] M. Harchol-Balter, M. Crovella, and C. Murta. Task assignment in a distributed server. In *Proceedings of the 10th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, September 1998.

[5] M. Harchol-Balter, M. Crovella, and S. Park. The case for SRPT scheduling in web servers. Technical Report MIT–LCS–TR–767, Laboratory for Computer Science, Massachussets Institute of Technology, October 1998.

[6] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In M. Atallah, editor, *Handbook of Algorithms and Theory of Computation.* CRC Press. To appear.

[7] E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. Graves, A. Rinnooy Kan, and P. Zipkin, editors, *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, pages 445–522. North-Holland, 1993.

[8] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 110–119, May 1997.

[9] A. Schulz and M. Skutella. Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria. Technical Report No. 533/1996, Technische Universität Berlin, November 1996. Revised March 1997.

[10] J. Sgall. On-line scheduling – a survey. In A. Fiat and G. Woeginger, editors, *On-Line Algorithms*. Springer-Verlag, Berlin, 1997.

[11] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

A A technical inequality

Lemma A.1 *Let m be a positive integer. Let X be a multiset of positive reals that satisfies the following condition.*

(C) *For any positive real x the sum of the elements in X with value at most x is at most mx.*

Then the following inequality holds true for any positive real z.

$$\sum_{x \in X, x \geq z} \frac{1}{x} \leq \frac{2m}{z}.$$

Proof: Let $v(X)$ denote the term $\sum_{x \in X, x \geq z} \frac{1}{x}$. Consider the following greedy algorithm for constructing X with the goal of maximizing $v(X)$. Initially set X to $\emptyset$. Iteratively, add to X the smallest element greater than or equal to z that can be added without violating condition C on X. Let x_i be the ith element added to X. We claim that that x_i may be defined as follows:

$$x_i = \begin{cases} z & \text{if } 1 \leq i \leq m, \\ \frac{\sum_{1 \leq j < i} x_j}{m-1} & \text{otherwise.} \end{cases}$$

We first prove the following two claims by induction on i: (i) $x_i \geq x_j$ for $1 \leq j < i$, and (ii) the multiset $\{x_j : 1 \leq j \leq i\}$ is a valid solution for X. For the base case, we consider $1 \leq i \leq m$. Since $x_i = z$, the desired claim holds trivially for the base case. We now assume that the desired claim holds for $i-1$. Consider iteration i. The element x_i equals $(\sum_{1 \leq j < i} x_j)/(m-1)$, which is at least $(\sum_{1 \leq j < i-1} x_j)/(m-1)$ and hence at least x_{i-1}, thus establishing part (i) of the induction step. For part (ii) of the induction step, we note that part (i) implies that we only need to check whether $\sum_{1 \leq j \leq i} x_j \leq m x_i$. By the definition of x_i it follows that $\sum_{1 \leq j \leq i} x_j = m x_i$, thus ensuring that the multiset $\{x_j : 1 \leq j \leq i\}$ is a valid solution for X. This completes the proofs of claims (i) and (ii).

We next show that $X^* = \{x_j : j \geq 1\}$ is an optimal choice for X. Let, if possible, Y be a different optimal solution. Let y_i denote the ith smallest element of Y, where we break ties arbitrarily. Let i be the smallest integer such that $y_i \neq x_i$. We now derive a contradiction in each of the following two cases: $y_i > x_i$ and $y_i < x_i$. If $y_i > x_i$, we can replace the element y_i in Y by the element x_i and obtain a valid solution yielding a value strictly larger than $v(Y)$, thus contradicting the assumption that $v(Y)$ is optimal. If $y_i < x_i$, then since Y is a valid solution, so is the multiset $\{y_j : 1 \leq j \leq i\}$. This implies the following inequality.

$$\begin{aligned}
\sum_{1 \leq j \leq i} y_j &= y_i + \sum_{1 \leq j < i} x_j \\
&= y_i + (m-1)x_i \\
&> y_i + (m-1)y_i \\
&= my_i,
\end{aligned}$$

thus violating condition C for Y.

To complete the proof of the lemma, we now compute $v(X^*)$. We first note that for $i > m$, x_i is equal to $z(m/m-1)^{i-m}$. Therefore, we can calculate $v(X^*)$ as follows.

$$\begin{aligned}
v(X^*) &= \frac{m}{z} + \sum_{i>m} \frac{1}{z} \left(\frac{m-1}{m}\right)^{i-m} \\
&= \frac{m}{z} + \sum_{j>0} \frac{1}{z} \left(\frac{m-1}{m}\right)^{j} \\
&= \frac{2m}{z}.
\end{aligned}$$

■

Corollary A.1.1 *Let m be a positive integer. Let X be a multiset of positive integers that satisfies the following condition.*

(C') *For any positive integer x the sum of the elements in X with value at most x is at most mx.*

Then the following inequality holds true for any integer z.

$$\sum_{x \in X, x \geq z} \frac{1}{x} \leq \frac{2m}{z}.$$

Proof: Since any multiset X of positive integers that satisfies condition C' also satisfies condition C of Lemma A.1, the desired claim follows from Lemma A.1. ■