ONLINE SCHEDULING TO MINIMIZE AVERAGE STRETCH

S. MUTHUKRISHNAN*, RAJMOHAN RAJARAMAN†, ANTHONY SHAHEEN‡, AND JOHANNES E. GEHRKE§

Abstract. We consider the classical problem of online 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. The average stretch metric has been used to evaluate the performance of scheduling algorithms in many applications arising in databases, networks and systems.

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 competitive ratio of SRPT. For multiprocessors, we show that the competitive ratio of SRPT is at most $9 + 2\sqrt{6} \leq 14$. Furthermore, we establish constant-factor lower bounds on the competitive ratio of any on-line algorithm for both uniprocessors and multiprocessors.

Key words. scheduling, online algorithms, competitive ratio, shortest remaining processing time (SRPT), stretch, slowdown

AMS subject classifications. 68W40, 68W10, 68Q99

1. Introduction.

Many servers, such as web and database servers, receive a continuous stream of requests with processing times that vary over several orders of magnitude. The main challenge in such a scenario is to schedule the stream of requests so as to provide various service guarantees such as response time, throughput, and fairness. In the combinatorial scheduling literature, this problem is typically abstracted as one of optimizing a suitably defined performance measure. Two classical performance measures are the *makespan*, which is the maximum completion time of any job, and the *total completion time*, which is the sum of the completion times of all the jobs; these are suitable for applications where the jobs are executed in batches. For jobs arriving continuously, a relevant parameter is the time that a job spends in the system, namely, the *response time* (also sometimes referred to as the *flow time*) of the job, which can be defined as the difference between the completion time and release time of the job. For over two decades, the *average response time* has been a popular performance measure in online scheduling research.

Recently, alternative performance measures have been considered. The premise underlying these studies is that an important parameter of performance for a job is the *factor by which it is slowed down relative to the time it takes on a unloaded system*. Formally, the *stretch* (also known as the *slowdown*) of a job is the ratio of the response time of the job to the processing time of the job. Intuitively, the stretch

*Department of Computer Science, Rutgers University, Piscataway, NJ 08854, Email: muthu@cs.rutgers.edu. Part of this work was done while the author was at AT&T Shannon Laboratories, Florham Park, NJ 07932.

†College of Computer & Information Science, Northeastern University, Boston MA 02115. Email: rraj@ccs.neu.edu. Part of this work was done when the author was at DIMACS.

‡Part of this work was done when the author was at DIMACS as part of the Research Experiences for Undergraduates program.

§Department of Computer Science, Cornell University, Ithaca, NY 14853. Email: johannes@cs.cornell.edu.

measure relates the users' waiting times to their demands since it measures the quality of service provided to a job in proportion to its demand. It may also reflect users' psychological expectation that in a system with highly variable job sizes users are willing to wait longer for larger requests. Thus it is a reasonably fair measure of the service that an individual job gets in the system.

The stretch measure, more precisely, the *average stretch* (or equivalently, the *total stretch*) has been used to empirically study the performance of several applications, e.g., in databases [20], operating systems [16], and parallel systems [14]. Average stretch is used for measuring the performance of scheduling algorithms in the context of web servers or clusters thereof [15, 24], that process requests for downloading files of different sizes and types. The average stretch is also a good indicator of the total load of the system. A schedule with large average stretch is likely to have a majority of jobs waiting in queue for large periods of time, while a schedule may have small average queue size, yet a large average response time, simply because one of the jobs has a large processing time.

To summarize, average stretch is a natural measure that fits many applications, and is a better indicator of system performance than the traditional average response time measure. Surprisingly, very little is known about scheduling algorithms that optimize (minimize) average stretch; in fact, the rather basic problem of minimizing average stretch on a single processor has not been studied.

In this paper, we study the problem of online scheduling to minimize average stretch on both single and multiprocessor systems. We focus on the *preemptive clairvoyant* version of the scheduling problem: by preemptive, we mean that jobs may be stopped before their completion and resumed later after processing other jobs in the interim; by clairvoyant, we mean that the processing time of each job is known at the time of its arrival. The aforementioned application of web server scheduling is an important example where preemptive clairvoyant schedules are useful.

Our main contribution is to show that the *shortest remaining processing time* (SRPT) heuristic is $O(1)$ competitive with respect to average stretch for both uniprocessors as well as multiprocessors. Our results show that the problem of optimizing the average stretch is fundamentally different from that of optimizing the average response time in the uniprocessor as well as the multiprocessor case, although, interestingly, the difference is contrasting. (See Section 1.2.) Our results, taken together with previous work on analyzing average response time [3, 19], imply that SRPT simultaneously optimizes the average response time and average stretch measures, upto constant factors, for both uniprocessors and multiprocessors.

1.1. Problem and definitions. 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. 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 a 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 total stretch of the jobs in a given instance.

We study the problem of minimizing total 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 [18], the problems that we study may be listed as $1 \mid pmtn, r_j \mid \sum_j (C_j - r_j)/p_j$

and $Pm \,|\, pmtn, r_j \,|\, \sum_j (C_j - r_j)/p_j$, respectively.

Since our focus is on online algorithms, we perform a competitive analysis [23] of the algorithms under consideration. An online algorithm $\mathcal{A}$ is *r-competitive* if for every instance $\mathcal{J}$, $S(\mathcal{J})$ is at most $rS^*(\mathcal{J})$, 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 SRPT algorithm. 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].)

1.2. Our results. Our main results are threefold:

- [UNIPROCESSOR SRPT] We show that for any instance $\mathcal{J}$ on a uniprocessor machine, the total stretch $S(\mathcal{J})$ achieved by SRPT is at most $S^*(\mathcal{J}) + |\mathcal{J}|$; since $S^*(\mathcal{J}) \geq |\mathcal{J}|$, this implies that SRPT is 2-competitive. We also establish an essentially matching lower bound on SRPT for uniprocessors. These results appear in Section 2.
  Our upper bound for SRPT makes it an attractive algorithm for uniprocessor scheduling since it is nearly optimal with respect to total stretch while it is known to be truly optimal with respect to average response time. Indeed, recent experimental and analytical studies on scheduling algorithms for web servers [4, 15] have shown that SRPT outperforms several other scheduling policies currently used in web servers on real-world workloads; the analyses in the preceding studies adopt a Poisson model for the job arrival process, and general as well as heavy-tailed distributions for job service times.
- [MULTIPROCESSOR SRPT] For multiprocessors, we show that for any instance $\mathcal{J}$, the total stretch $S(\mathcal{J})$ achieved by SRPT is at most $3S^*(\mathcal{J}) + (6 + 2\sqrt{6})|\mathcal{J}|$, which implies an upper bound of $9 + 2\sqrt{6} \leq 14$ on the competitive ratio, independent of the number of processors $m \geq 2$. These results appear in Section 3.
  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 [19] 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. In [19], 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 constant factors for both total stretch as well as average response time.
- [LOWER BOUNDS] We show that there exists no on-line algorithm that minimizes average stretch for every instance; specifically, we show that the com-

[1]A job may be preempted at one processor and later resumed at a different processor.

petitive ratio of every online algorithm is at least 1.04. We also show a lower bound of at least 7/6 on the competitive ratio for minimizing total stretch on any $2m$-processor machine, for any positive integer m. These results appear in Section 4.

The average response time and total 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 total stretch is strictly greater than 1. On the other hand, for multiprocessors, while a constant-factor competitive online algorithm exists for total stretch, the best competitive ratio for average response time grows logarithmically with the number of jobs.

1.3. Related work. While average response time has been studied extensively in the literature [3, 17, 22], the analytical study of stretch as a performance measure was initiated only recently [6]. All of the results presented in [6], however, concern the metric of *maximum stretch* (that is, $\max_i S_i$) and do not yield any useful bounds for total stretch. Recently, max-stretch was also studied in the context of performing file transfers over a network with given bandwidth constraints on the underlying links [12].

Two measures closely related total stretch are weighted completion time and weighted flow time, each of which associate a weight with each job i. If we set the weight of job i to the reciprocal of flow time completion time also optimizes total stretch. From the point of view of approximation, however, the two metrics are different. A randomized online algorithm that achieves a competitive ratio of 4/3 with respect to weighted completion time is given in [21]; this result, however, does not directly yield any useful upper bound on the competitive ratio for total stretch. In fact, it is easy to construct schedules that have a constant factor approximation ratio with respect to weighted completion time, with $w_i = 1/p_i$, and yet have a linear approximation ratio with respect to total stretch. For other results on the online and offline complexity of weighted completion time, see [13].

The weighted flow time with weights given by the reciprocal of processing times is identical to the total stretch metric. The best known approximation result for weighted flow time is the recent approximation scheme of [8], which takes time superpolynomial, but subexponential, in the input size. In a subsequent study [9], it has been shown that a quasi-PTAS is achievable for weighted flow time when Δ and the ratio of the maximum weight to minimum weight are both polynomially bounded. When applied to the special case of the total stretch metric, the techniques of [9] yield a PTAS. A PTAS for total stretch is also given in [7].

For multiprocessors, the work most closely related to our study is that of [19], which is discussed in Section 1.2. In recent work [1], a polynomial-time approximation scheme is derived for the weighted completion time problem on multiprocessors.

1.4. Overview of our analyses. 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. For the case of uniprocessors, we derive a tight bound on the total stretch of SRPT by placing a separate bound, for *every* unfinished job at *every* time step, on the contribution of the job to the total stretch of SRPT. A key component behind this approach is to appropriately map the jobs in SRPT's queue to the jobs in the queue associated with an arbitrary scheduling algorithm at every time step. The particular mapping enables us to analyze the total stretch of SRPT time step by time step. The relevant property of this mapping is illustrated in Figure 2.1 of Section 2 and formally established in Lemma 2.8.

Our analysis for multiprocessors is more involved. The difficulty in the analysis arises due to the following observation made in [19]: there exist multiprocessor scheduling instances that may force SRPT to have many more unfinished jobs at certain time steps than an alternative schedule has at that time. As a result, the contribution to the total stretch corresponding to such time steps may be disproportionately larger for SRPT than for the other schedule. We overcome this hurdle by showing the existence of an appropriate partial mapping from the jobs in SRPT's queue to the jobs in the queue associated with the other schedule. In addition to placing an upper bound on the contribution of the mapped jobs in each step, the particular mapping enables us to place an upper bound on the exact or amortized contribution of the unmapped jobs in SRPT's queue at each time step. The mapping is illustrated in Figure 3.1 of Section 3 and formally established in Lemma 3.5.

2. Analysis of SRPT for uniprocessor scheduling.

In this section, we analyze the competitiveness of SRPT on uniprocessors with respect to total stretch. An instance $\mathcal{J}$ of our scheduling problem consists of a set of jobs with arbitrary release times and processing times. For job $j \in \mathcal{J}$, let $r(j)$ and $p(j)$ denote the release time and processing time, respectively, of j. The performance of a schedule is measured by the total stretch achieved. Recall that the total stretch of a schedule for a given instance is the sum, taken over all jobs $j \in \mathcal{J}$, of the stretch of j under the schedule, where the stretch of a job is the ratio of the response time of the job to the processing time of the job. Let $S(\mathcal{J})$ and $S^*(\mathcal{J})$ denote the total stretch of SRPT and the optimal total stretch, respectively, for instance $\mathcal{J}$. The main result of this section is the following theorem that places an upper bound on the total stretch achieved by SRPT.

THEOREM 2.1. *For any instance $\mathcal{J}$, $S(\mathcal{J})$ is at most $S^*(\mathcal{J}) + |\mathcal{J}|$.*

The proof of Theorem 2.1 is given in Section 2.1. In terms of competitive ratio, Theorem 2.1 implies the following result.

COROLLARY 2.2. 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 2.1 that $S(\mathcal{J}) \leq 2S^*(\mathcal{J})$. □

We also show an essentially matching lower bound in Theorem 2.3, which is proved in Section 2.2.

THEOREM 2.3. *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 $S^*(\mathcal{J}) + n - 1 - \varepsilon$.*

2.1. Upper bound.

In this section, we prove Theorem 2.1. We begin by introducing 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 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 ∞. We refer to a job as *active* (resp., *waiting*) at time t if the rank of the job is equal to 1 (resp., greater than 1). Note that in time step t, SRPT processes one unit of the job that is active at that time. Let U_t denote the set of waiting jobs at time t.

We now express the total stretch $S(\mathcal{J})$ in SRPT as a sum, taken over all time steps t and over all jobs in Q_t, of the reciprocal of the processing time of the job.

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

We now introduce two new variables for notational convenience. Let α_t denote the sum of the reciprocals of the processing times of the jobs in Q_t. Thus, the total stretch $S(\mathcal{J})$ is simply $\sum_t \alpha_t$. Finally, let β_t denote the sum of the reciprocals of the remaining processing times of the jobs in Q_t. Since the remaining processing time of a job at any time is at most its processing time, we obtain that $\alpha_t \leq \beta_t$ for all t.

We split the total stretch $S(\mathcal{J})$ into two parts $S_1(\mathcal{J})$ and $S_2(\mathcal{J})$, which represent the total contribution of the active jobs and the waiting jobs, respectively, in SRPT.

$$S_1(\mathcal{J}) = \sum_t \frac{1}{p(q_{t,1})} \text{ and}$$
$$S_2(\mathcal{J}) = \sum_t \sum_{j \in U_t} \frac{1}{p(j)}.$$

The total contribution of the active jobs can be calculated easily. Whenever a job j is active, it contributes $1/p(j)$ to the term $S_1(\mathcal{J})$. Since an active job is always processed by SRPT, there are exactly $p(j)$ time steps when j is active. Thus, j contributes $p(j)/p(j) = 1$ to $S_1(\mathcal{J})$. We thus obtain the following lemma.

LEMMA 2.4. *For any instance $\mathcal{J}$, $S_1(\mathcal{J}) = |\mathcal{J}|$.* □

We place an upper bound on the contribution of the waiting jobs by means of a careful comparison of SRPT's queue Q_t with the corresponding queue 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 obtain the desired upper bound on $S_2(\mathcal{J})$ by showing the following key lemma that relates β_t and $\widetilde{\alpha}_t$.

LEMMA 2.5. *For any time t, we have $\beta_t \leq \widetilde{\alpha}_t + \frac{1}{\rho_t(q_{t,1})}$.*

COROLLARY 2.6. *For any instance $\mathcal{J}$, $S_2(\mathcal{J}) \leq S^*(\mathcal{J})$.*

Proof. Since $1/p(j)$ is at most $1/\rho_t(j)$ for any job j, it follows that

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

where the last step is obtained by invoking Lemma 2.5. Since the above inequality holds for every scheduling algorithm $\mathcal{A}$, the desired claim follows. □

Theorem 2.1 follows directly from Lemma 2.4 and Corollary 2.6. So all that remains to prove is Lemma 2.5. A crucial component of our proof of Lemma 2.5 is to establish the following relationship between the queues of SRPT and $\mathcal{A}$: for any $i > 1$, the remaining processing time of the job of rank i in Q_t is *at least* the remaining processing time of the job of rank $i - 1$ in $\widetilde{Q}_t$. The preceding claim is formally stated in Lemma 2.8 and illustrated in Figure 2.1.

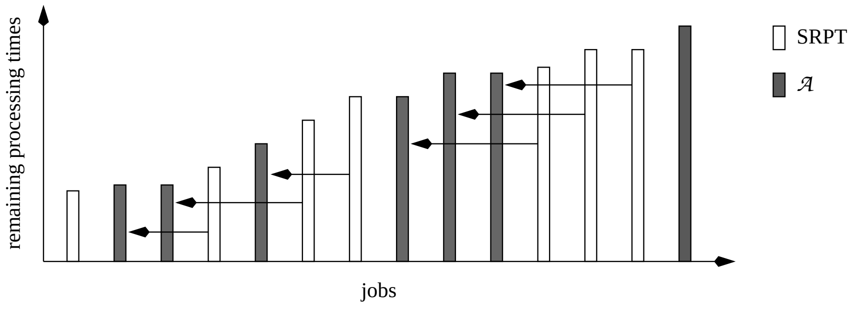

FIG. 2.1. *The remaining processing times of the jobs in the queues of* SRPT *and $\mathcal{A}$ at time t. The shaded jobs belong to $\mathcal{A}$'s queue $\widetilde{Q}_t$ while the unshaded jobs belong to* SRPT*'s queue Q_t. The arrows indicate that for any $i > 1$, the remaining processing time of the job of rank i in Q_t is at least that of the job of rank $i-1$ in $\widetilde{Q}_t$.*

We establish Lemma 2.8 by relating appropriately defined "prefixes" of the two queues Q_t and $\widetilde{Q}_t$. 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 call $V_t(i)$ the *volume* of jobs in Q_t with remaining processing time at most i.

LEMMA 2.7. *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 holds trivially since for all i, $V_0(i) = \widetilde{V}_0(i)$. For the induction hypothesis, we assume that the 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 t does not change $V_t(i) - \widetilde{V}_t(i)$ for any i. Now 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 $V_t(i) \le \widetilde{V}_t(i) + i$. 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; this happens when a job with remaining processing time $i+1$ gets processed and enters $P_t(i)$. Therefore, we have $V_t(i) \le i \le \widetilde{V}_t(i) + i$. This completes the proof of the desired claim. □

LEMMA 2.8. *For any time t and any integer $k > 1$, we have $\rho_t(q_{t,k}) \ge \widetilde{\rho}_t(\widetilde{q}_{t,k-1})$.*

Proof. We first prove that for all $k > 1$, $\rho_t(q_{t,k}) \ge \widetilde{\rho}_t(\widetilde{q}_{t,k-1})$. 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 $|P_t(b)| \ge k$, while $|\widetilde{P}_t(b)| = k-2$. We thus have the following inequality.

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

$$> b + \widetilde{V}_t(b),$$

thus contradicting Lemma 2.7. (In the third step, we use the assumption that for $1 < i < k$, $\rho_t(q_{t,i}) \geq \widetilde{\rho}_t(\widetilde{q}_{t,i-1})$. In the final step, we use the fact that $\rho_t(q_{t,1})$ is positive.) □

COROLLARY 2.9. *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 2.8.

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

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 2.5.

Proof of Lemma 2.5. 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 the 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 Corollary 2.9.

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

□

2.2. Lower bound. In this section, we prove Theorem 2.3. Consider an instance in which a job of size ℓ_1 arrives in time step 0 and one job of size $\ell_2 \ll \ell_1$ arrives in time steps $\ell_1 + \ell_2(i-1) + 1$ for $0 \leq i < n-1$. An SRPT schedule will process the large job until time $\ell_1 - \ell_2 + 1$ when the first small job arrives. Since the remaining processing time of the large job $(\ell_2 - 1)$ is smaller than the size of the small job, SRPT continues processing the large job until it is finished. Thereafter, the $n-1$ small jobs are processed one after another. The total stretch of the SRPT schedule is $n + (n-1)(\ell_2 - 1)/\ell_2 = 2n - 1 - (n-1)/\ell_2$. Given $\varepsilon > 0$, we set $\ell_2 > 2(n-1)/\varepsilon$ so that the total stretch of SRPT is at least $2n - 1 - \varepsilon/2$.

An alternative schedule is to complete the small jobs immediately upon their arrival, and complete the large job at time $\ell_1 + (n-1)\ell_2$. This yields a total stretch of $n + (n-1)\ell_2/\ell_1$. We set $\ell_1 > 2(n-1)\ell_2/\varepsilon$ so that the preceding schedule has total stretch at most $n + \varepsilon/2$.

Thus, we have $S(\mathcal{J}) \geq S^*(\mathcal{J}) + n - 1 - \varepsilon$. This completes the proof of Theorem 2.3.

3. Analysis of SRPT for multiprocessor scheduling.

This section analyzes the competitiveness of SRPT on an m-processor machine, where m is any integer greater than 1. For any instance $\mathcal{J}$, let $S(\mathcal{J})$ and $S^*(\mathcal{J})$ denote the total stretch of SRPT and an optimal schedule, respectively. The main result of this section is the following upper bound on the total stretch achieved by SRPT, which is proved in Section 3.1.

THEOREM 3.1. *For any instance $\mathcal{J}$ on multiprocessors, $S(\mathcal{J})$ is at most $3S^*(\mathcal{J}) + (6 + 2\sqrt{6})|\mathcal{J}|$. Thus, the competitive ratio of* SRPT *for multiprocessors is at most* $9 + 2\sqrt{6} \leq 14$. We can also show a constant-factor lower bound on the competitive ratio of SRPT for multiprocessors. The lower bound is proved in Section 3.2.

THEOREM 3.2. *For any even integer $m > 1$ and any positive real ε, there exists a scheduling instance $\mathcal{J}$ for an m-processor machine such that $S(\mathcal{J})$ is at least $(2.5 - \varepsilon)S^*(\mathcal{J})$.*

3.1. Upper bound.

A crucial component of our upper bound proof for uniprocessors in Section 2 is the property of SRPT's queue established in Lemma 2.8. Lemma 2.8 implies 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 such that the following claim holds: 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 presented in [19], where it is used to derive a lower bound on the competitive ratio of SRPT with respect to total flow time.

While establishing an upper bound on the total flow time of SRPT, [19] provides a characterization of the queue of SRPT that implies an upper bound on the *number of jobs* in the queue of SRPT at each step. We note that the total flow time of a schedule is the sum, taken over every time step t, of the number of jobs in the queue at time t. The analysis of [19] does not suffice for our purposes, however, because the total stretch takes into account the sum of the *reciprocals of the processing times* of the jobs in the queue at any time step. In our analysis of SRPT that follows, we perform a more careful comparison between SRPT and an optimal schedule in terms of the relative distribution of the remaining processing times of the jobs in the two queues at each time step. We are able to establish a *partial* mapping from the jobs in SRPT's queue to the jobs in another schedule such that the following property of the mapping holds: the remaining processing time of a mapped job in SRPT's queue is at least that of the job to which it is mapped. This mapping, together with an appropriate characterization of the unmapped jobs in SRPT's queue, then enables us to place an upper bound on the exact or "amortized" contribution of the jobs in the queue at each time step.

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. 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 $S_2(\mathcal{J})$.

$$S_1(\mathcal{J}) = \sum_t \sum_{j \in F_t} \frac{1}{p(j)} \text{and}$$

$$S_2(\mathcal{J}) = \sum_t \sum_{j \in U_t} \frac{1}{p(j)}.$$

By invoking the same argument that is used in proving Lemma 2.4, we show that $S_1(\mathcal{J})$ equals $|\mathcal{J}|$.

LEMMA 3.3. *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. 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. For any time t and positive integer i, let $V_t(i)$ denote the volume of $P_t(i)$.

The following lemma is an easy generalization of Lemma 2.7. (As in Section 2 we follow the notational convention that the "tilde" version of a variable defined for SRPT denotes the corresponding variable for $\mathcal{A}$.)

LEMMA 3.4. *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 the 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 $V_t(i) \le \widetilde{V}_t(i) + mi$. 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 $V_t(i) \le mi \le \widetilde{V}_t(i) + mi$. □

Given any schedule $\mathcal{A}$ and any time t, we define a partial mapping f_t from U_t to $\widetilde{Q}_t$ that satisfies properties P1 and P2 defined below. Let MPD_t (resp., UMPD_t) denote the set of jobs in U_t that are mapped (resp., unmapped) by f_t.

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

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

LEMMA 3.5 (Partial mapping). *For any schedule $\mathcal{A}$ and any time t, there exists an injective mapping f_t from U_t to $\widetilde{Q}_t$ that satisfies properties P1 and P2.*

Proof. We describe a procedure for defining the mapping f_t. This procedure runs in phases, starting from phase 1. Let X denote the set of jobs in $U_t \cap P_t(i)$ with remaining processing time equal to i. Let Y denote the set of jobs in $\widetilde{P}_t(i)$ to which no job in U_t has been mapped at the start of phase i. (At the start of phase 1, Y is $\emptyset$.) In phase i, we map as many jobs in X to jobs in Y as possible.

We now argue that for every i, the following two properties hold at the start of phase i: (i) for each job j in U_t that has been mapped, $\widetilde{\rho}_t(f_t(j)) \le \rho_t(j)$; (ii) for $0 < k < i$, the total volume of the unmapped jobs in $U_t \cap P_t(k)$ is at most mk. Clearly, if the two properties hold at the end of the procedure when all of the jobs in U_t have been considered, then the desired claim follows.

By the definition of the mapping, it is evident that property (i) holds at the start of every phase. We now prove property (ii). The proof is by induction on i. For the induction basis, $i = 1$ and the property (ii) holds vacuously. We now assume that property (ii) holds at the start of phase i and consider the effect of phase i.

By property (ii) of the induction hypothesis, it follows that for $0 < k < i$, the total volume of the unmapped jobs in $U_t \cap P_t(k)$ is at most mk. Thus, it suffices to prove that at the start of phase $i + 1$, the total volume of the unmapped jobs in $U_t \cap P_t(i)$ is at most mi. 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 (i), 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 3.4 that the volume of unmapped jobs in $U_t \cap P_t(i)$ 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 U_t with remaining processing time equal to i is mapped in phase i. Therefore, we invoke property (ii) of the induction hypothesis to obtain that the total volume of the jobs in $U_t \cap P_t(i)$ that are unmapped at the end of phase i is at most $m(i-1) \le mi$. This completes the induction step and the proof of the lemma. $\square$

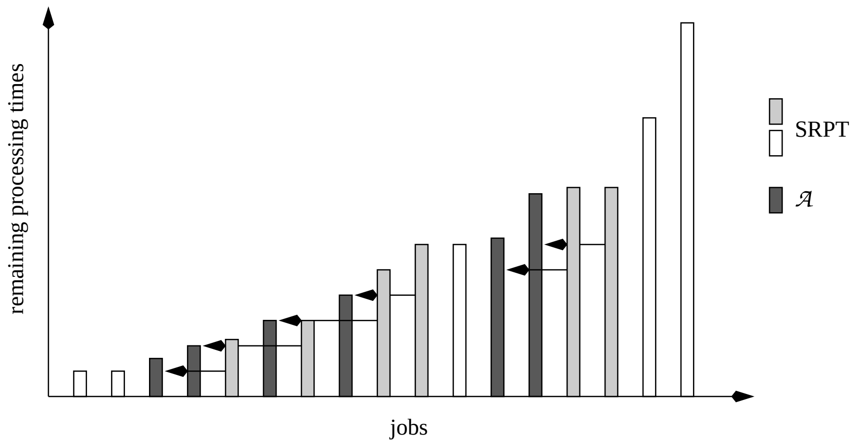

FIG. 3.1. *The remaining processing times of the jobs in the queues of* SRPT *and $\mathcal{A}$ at time t. The arrows depict the mapping f_t. The darkly shaded jobs belong to $\mathcal{A}$'s queue, while the lightly shaded and unshaded jobs belong to* SRPT*'s queue. The lightly shaded jobs are the mapped jobs in* SRPT*'s queue.*

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 3.5 holds for any schedule $\mathcal{A}$, the particular mapping f_t that we employ for the remainder of this proof is with respect to 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 2.5 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 3.5, we know that $\rho_t(j) \ge p(f_t(j))$. Since f is injective, we obtain the following lemma about the mapped jobs in Q_t.

LEMMA 3.6 (Mapped jobs). *For any time t, we have $\sum_{j\in \mathrm{MPD}_t} \frac{1}{\rho_t(j)} \le \alpha'_t$.* □

We now consider the unmapped jobs. We begin by establishing a technical lemma that is helpful in placing an upper bound on the contribution of the unmapped jobs. (The reader may choose to skip the proof of this lemma on their first pass through this section.)

LEMMA 3.7. *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.

$$\sum_{x\in X, x\ge z} \frac{1}{x} \le \frac{2m-1}{z}.$$

Proof. We first consider the case $m = 1$. By condition **(C)**, this case implies that X has at most one element. Therefore, the desired inequality follows trivially.

In the remainder of the proof, we assume that $m > 1$. Let $v(X)$ denote the term $\sum_{x\in X, x\ge z} \frac{1}{x}$. 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 the condition on X as stated in the lemma. Let x_i be the ith element added to X. We claim that that x_i may be defined as follows:

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

We first prove the following two claims by induction on i: (i) $x_i \ge x_j$ for $1 \le j < i$, and (ii) the multiset $\{x_j : 1 \le j \le i\}$ is a valid solution for X. For the base case, we consider $1 \le i \le m$. 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 $(\sum_{1\le j<i} x_j)/(m-1)$, which is at least $(\sum_{1\le j<i-1} x_j)/(m-1)$ 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 $\sum_{1\le j\le i} x_j \le mx_i$. By the definition of x_i it follows that $\sum_{1\le j\le i} x_j = mx_i$, thus ensuring that the multiset $\{x_j : 1 \le j \le i\}$ is a valid solution for X. This completes the proofs of claims (i) and (ii).

We next show that $X^* = \{x_j : j \ge 1\}$ 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 $y_i \ne x_i$. 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 $\{y_j : 1 \le j \le i\}$. This implies the following inequality.

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

$$\begin{aligned} &> y_i + (m-1)y_i \\ &= my_i, \end{aligned}$$

thus violating the condition stated in the lemma. (The argument for the second step is the following: since all of the elements in Y are at least z, we have $i > m$, implying that $(m-1)x_i = \sum_{1 \le j < i} x_j$.)

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.

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

□

In the following lemma, we invoke the inequality established above to place an upper bound on the sum of the reciprocals of the remaining processing times of certain subsets of unmapped jobs.

LEMMA 3.8. *For any positive integer i and any step t, we have*

$$\sum_{j \in \mathrm{UMPD}_t, \rho_t(j) \ge i} \frac{1}{\rho_t(j)} \le \frac{2m-1}{i}.$$

Proof. We invoke Lemma 3.7, with X equal to the set $\{j \in \mathrm{UMPD}_t : \rho_t(j) \ge i\}$, to prove the desired claim. We note that the condition C of Lemma 3.7 holds due to property P2 of Lemma 3.4. The claim of Lemma 3.7 is the desired inequality. □

Lemma 3.8 indicates that if the rank of every unmapped job exceeds the rank of $\Omega(m)$ mapped jobs at any time step t, then the total stretch contribution of the unmapped jobs at t can be upper bounded by a constant factor times the sum of the reciprocals of the remaining processing times of the mapped jobs, which in turn is bounded by Lemma 3.6. This suggests classifying the unmapped jobs at any time t into two groups as follows. A job j is *marked at time t* if $j \in \mathrm{UMPD}_t$ and the number of mapped jobs in $P_t(\rho_t(j))$ is at most m. Note that whether a job is marked depends on the particular time t; thus, for instance, a job may be marked at one time step and may be unmarked at a subsequent step. 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 ∞. Let MKD_t and UMKD_t denote the set of marked and unmarked jobs, respectively, at time t. We first account for the contribution of unmarked jobs by relating it to the contribution of mapped jobs of lesser rank.

LEMMA 3.9 (Unmarked jobs). *For any time t, we have:*

$$\sum_{j \in \mathrm{UMKD}_t} \frac{1}{p(j)} \le \sum_{j \in \mathrm{MPD}_t} \frac{2}{\rho_t(j)}.$$

Proof. For any mapped job j in Q_t, let $X(j)$ denote the set of unmarked jobs in Q_t that have rank greater than that of j. Since each job in $X(j)$ is unmapped, the following inequality follows from Lemma 3.8.

$$\sum_{j_1 \in X(j)} \frac{1}{\rho_t(j_1)} \leq \frac{2m-1}{\rho_t(j)} \tag{3.1}$$

By definition, for each unmarked job j_1, there exist at least m jobs j such that j_1 is in $X(j)$. Therefore, we obtain the following inequality.

$$\begin{aligned}\sum_{j_1 \in \mathrm{UMKD}_t} \frac{1}{p(j_1)} &\leq \sum_{j_1 \in \mathrm{UMKD}_t} \frac{1}{\rho_t(j_1)} \\ &\leq \frac{1}{m} \sum_{j \in \mathrm{MPD}_t} \sum_{j_1 \in X(j)} \frac{1}{\rho_t(j_1)} \\ &\leq \frac{1}{m} \sum_{j \in \mathrm{MPD}_t} \frac{2m-1}{\rho_t(j)} \\ &\leq \sum_{j \in \mathrm{MPD}_t} \frac{2}{\rho_t(j)}.\end{aligned}$$

(The third inequality is derived by adding Equation 3.1 over all the mapped jobs.) □

We now consider the marked jobs. We call a marked job j *dependent* at time t, if the number of jobs in F_t with release time at least the marking time of j is greater than γm, where γ is a constant in $(0, 1)$ that is specified later. A marked job that is not dependent at time t is said to be *independent.* Let DEP_t and IND_t denote the set of dependent and independent jobs, respectively, at time t. We account for the contributions of the jobs in IND_t and DEP_t in the following manner. In Lemma 3.10, we show that any job j is independent during $O(p(j))$ steps only. This implies that the total contribution of the independent jobs during the execution of SRPT is $O(|\mathcal{J}|)$. Next, in Lemma 3.11, 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 3.3 to be equal to $|\mathcal{J}|$.

LEMMA 3.10 (Independent jobs). *Any job j is independent during at most $2p(j)/(1-\gamma)$ time steps.*

Proof. Since any job j is independent only when it is marked, we only need to consider time steps beginning from 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 3.5, 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 all the jobs in X is at most $2mp(j)$.

Consider any time step $t' \geq t$ when j is independent. It follows from the definition of independence that there are at most γm jobs in $F_{t'}$ with release time at least t. Since j is not in $F_{t'}$, it follows that there are at least $(1-\gamma)m$ jobs in X that are in $F_{t'}$. Thus, the volume of the jobs in X decreases by at least $(1-\gamma)m$ in time step t_1. Once all of the jobs in X are completed, j can be never be independent. Since the total volume of X at time t is at most $2mp(j)$, the number of time steps when j is independent is at most $2p(j)/(1-\gamma)$. □

We next consider the dependent jobs.

LEMMA 3.11 (Dependent jobs). *For any time t, we have:*

$$\sum_{j \in \mathrm{DEP}_t} \frac{1}{p(j)} \leq \frac{3}{\gamma} \sum_{j \in F_t} \frac{1}{p(j)}.$$

Proof. Consider a job j in F_t. Let $X(j)$ be the set of jobs in DEP_t whose marking time is at most 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 $m-1$ because every job in $X(j)$ has rank greater than that of j.

Consider the jobs in $X(j) \setminus Y(j)$. For every job j_1 in $X(j) \setminus Y(j)$, we have $\rho_t(j_1) = \rho_{r(j)}(j_1) \geq p(j)$. Moreover, since every job in $X(j)$ is unmapped, we obtain the following inequality from Lemma 3.8:

$$\sum_{j_1 \in X(j) \setminus Y(j)} \frac{1}{p(j_1)} \leq \frac{2m}{p(j)} \tag{3.2}$$

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.

$$\sum_{j_1 \in Y(j)} \frac{1}{p(j_1)} \leq \frac{m-1}{p(j)} \tag{3.3}$$

Equations 3.2 and 3.3 together yield the following inequality for each job j in F_t.

$$\sum_{j_1 \in X(j)} \frac{1}{p(j_1)} \leq \frac{3m}{p(j)} \tag{3.4}$$

Now consider a job j_1 in DEP_t. Since j_1 is dependent, there are more than γm 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 3.4 over all jobs in F_t, j_1 appears at least $\lceil \gamma m \rceil$ times on the left-hand side. Therefore, we obtain the following inequality, that establishes the desired claim.

$$\sum_{j_1 \in \mathrm{DEP}_t} \frac{1}{p(j_1)} \leq \frac{3}{\gamma} \sum_{j \in F_t} \frac{1}{p(j)}$$

□

Putting Lemmas 3.6, 3.9, 3.10, and 3.11 together, we obtain the following main lemma.

LEMMA 3.12. *For any instance $\mathcal{J}$, we have:*

$$\sum_t \sum_{j \in U_t} \frac{1}{p(j)} \leq 3S^*(\mathcal{J}) + \left(\frac{2}{1-\gamma} + \frac{3}{\gamma}\right) |\mathcal{J}|.$$

Proof. We establish the desired inequality as follows.

$$\sum_t \sum_{j \in U_t} \frac{1}{p(j)} = \sum_t \left(\sum_{j \in \mathrm{MPD}_t} \frac{1}{p(j)} + \sum_{j \in \mathrm{IND}_t} \frac{1}{p(j)} + \sum_{j \in \mathrm{DEP}_t} \frac{1}{p(j)} + \sum_{j \in \mathrm{UMKD}_t} \frac{1}{p(j)} \right)$$

$$\leq \sum_t \sum_{j \in \mathrm{MPD}_t} \frac{3}{\rho_t(j)} + \sum_j \sum_{t: j \in \mathrm{IND}_t} \frac{1}{p(j)} + \frac{3}{\gamma} \sum_t \sum_{j \in F_t} \frac{1}{p(j)}$$
$$\leq \sum_t 3\alpha'_t + \frac{2|\mathcal{J}|}{1-\gamma} + \frac{3}{\gamma} \sum_t \sum_{j \in F_t} \frac{1}{p(j)}$$
$$\leq 3S^*(\mathcal{J}) + \left(\frac{2}{1-\gamma} + \frac{3}{\gamma}\right) |\mathcal{J}|.$$

(In the second step, we invoke Lemmas 3.9 and 3.11. In the third step, we invoke Lemmas 3.6 and 3.10. The final step follows from Lemma 3.3.) □

The term $2/(1-\gamma)+3/\gamma$ attains a minimum value of $5+2\sqrt{6} \leq 10$ when $\gamma = 3-\sqrt{6}$. The main theorem now directly follows from Lemmas 3.3 and 3.12.

3.2. Lower bound. We now prove Theorem 3.2. We consider the case of two processors only; the argument can be easily extended to the case of $2m$ processors for any positive integer m. The main idea in the argument is similar to the one used in [19]. Consider an instance that starts with 3 jobs, one of size 2^n, two of size 2^{n-1}, all of which arrive at time 0. At time 2^n, one job of size 2^{n-1} and two jobs of size 2^{n-2} are released. In general, for $1 \leq i \leq n$, at time $\sum_{i<j\leq n} 2^j$, one job of size 2^i and two jobs of size 2^{i-1} are released. Finally, at each of the p steps of the interval $[2^{n+1}-2, 2^{n+1}+p-3]$, a pair of unit-size jobs are released. (Here p is an integer that is specified below.) The optimal total stretch equals $4n+2p$. The total stretch achieved by SRPT is at least $3n+4p+p(1/2^{n-2}+1/2^{n-1}+\ldots 1/2)$ which is $3n+(5-1/2^{n-2})p$. For any constant $\varepsilon > 0$, we can choose p sufficiently larger than n so as to make the competitive ratio at least $2.5-\varepsilon$.

4. Lower bounds.

This section contains lower bounds on the competitive ratio of arbitrary online algorithms for uniprocessor and multiprocessor scheduling. Section 4.1 concerns uniprocessor scheduling while Section 4.2 concerns multiprocessor scheduling.

4.1. Uniprocessor scheduling. Our main result for lower bounds on online uniprocessor scheduling is the following.

THEOREM 4.1. *If there are only two distinct job sizes, there exists an optimal online algorithm for minimizing average stretch. If there are three or more distinct job sizes, the competitive ratio of any online algorithm is at least* 1.04.

Proof. Consider the case when there are only two distinct job sizes. 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 partially processed job in each category, and that is the job at the head of the FIFO order in that category at that time; let ℓ and s denote the job at the head of the FIFO order in the large and small categories, respectively. We need to decide which one of these two jobs must be processed. Suppose we complete the large job ℓ first, and follow it by processing the small job s, then the total stretch contributed by the two jobs is

$$\frac{t+\rho_t(\ell)-r(\ell)}{p(\ell)} + \frac{t+\rho_t(\ell)+\rho_t(s)-r(s)}{p(s)}.$$

On the other hand, if we complete the small job first, followed by the the large job, then the stretch contributed by the two jobs is

$$\frac{t+\rho_t(s)-r(s)}{p(s)}+\frac{t+\rho_t(\ell)+\rho_t(s)-r(\ell)}{p(\ell)}.$$

The former is desirable when $\rho_t(\ell)/p(s) \leq \rho_t(s)/p(\ell)$, that is, when $\rho_t(\ell)p(\ell) \leq \rho_t(s)p(s)$.

We now argue that the following online algorithm is optimal: at each time step t, process a job i with the smallest $\rho_t(i)p(i)$. The proof is by contradiction. Suppose there exists an optimal schedule in which there is a time step t at which a job i with the smallest $\rho_t(i)p(i)$ is not processed. Let t be the earliest such time step. Let s and ℓ denote the jobs at the head of the FIFO order in the small and large categories, respectively, at time t. Let us consider the case that $\rho_t(\ell)p(\ell) < \rho_t(s)p(s)$, yet job s is processed at time t. Let t' denote the completion time of job ℓ in the schedule. It then follows that the set of jobs that complete during the interval $[t,t')$ are all small jobs (this set is non-empty since s belongs to it). Let j denote the last small job to complete in the interval $[t,t')$. Consider the schedule in which we swap the order of jobs j and ℓ and complete ℓ before j; thus, the completion time of ℓ is decreased by at least $\rho_t(s)$, and the completion time of j is increased by at most $\rho_t(\ell)$. As a result, the decrease in total stretch is at least

$$\frac{\rho_t(s)}{p(\ell)}-\frac{\rho_t(\ell)}{p(s)}>0,$$

which contradicts the assumption that the given schedule is optimal. The analysis for the other case is symmetric, thus completing the proof of the first claim of the theorem.

We next consider the lower bound when there are three or more distinct job sizes. We give the proof of a weaker lower bound first. 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 $\{j_1,j_2,j_3\}$. For $\mathcal{J}_1$, 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 $1+7/5=2.4$. For $\mathcal{J}_2$, 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. For 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 best scenario for the algorithm with regard to instance $\mathcal{J}_2$ 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.

We can generalize the above example as follows. Let ℓ, m, and s be the jobs with processing times $p(\ell)$, $p(m)$, and $p(s)$, where $p(\ell) > p(m) > p(s)$. Consider the instance in which ℓ arrives at time 0 and m arrives at the end of time step $p(\ell)-k$, where $k \leq p(m)$. If the algorithm does not preempt the first job when the second job arrives, the competitive ratio is at least $(2+(k/p(m)))/(2+(p(m)/p(\ell)))$. Otherwise, job s is presented at the end of time step $p(\ell)$. Then we can argue as above that any such algorithm has average stretch at least $\min\{3+p(s)/p(m)+(p(m)+p(s))/p(\ell), 3+(p(m)+p(s))/p(\ell)+(p(m)-k)/p(s)\}$ while the optimum average stretch is at most $3+(k+p(s))/p_m$. The lower bound on the competitive ratio, thus obtained, is nearly maximized by setting $p(s) = 54$, $p(m) = 100$, $p(\ell) = 200$, and $k = 60$; this setting of parameters yields that the average stretch is at least 1.04. □

4.2. Multiprocessor scheduling. For multiprocessors, we can establish a slightly stronger lower bound on the competitive ratio of any online algorithm.

THEOREM 4.2. *For any positive integer m, the competitive ratio of any online algorithm for any $2m$-processor machine is at least 7/6.*

Proof. We first consider the case of a 2-processor machine. We consider two instances of the scheduling problem. One instance consists of three jobs, two of size 1 and one of size $\ell > 1$, all of which are released at time 0. The parameter ℓ is an integer, whose value is specified below. The optimal schedule for this instance is to schedule the two size-1 jobs in parallel and then schedule the size-ℓ job, thus yielding a total stretch of $3 + 1/\ell$. If the given online algorithm schedules the two unit-size jobs in sequential order on one of the processors, then it achieves a total stretch of 4, thus yielding a competitive ratio of $4\ell/(3\ell + 1)$.

If the online algorithm instead schedules the above instance optimally, then we consider its performance on another scheduling instance which consists of the same set of 3 jobs as above, together with a sequence of unit-size jobs, arriving in pairs at times $\ell, \ell+1, \ldots, \ell+n-1$. Since the incomplete size-ℓ job has at least one unit of processing time remaining, the online algorithm will be forced to delay the size-ℓ job until all of the size-1 jobs complete, thus leading to a total stretch of $2n + 3 + (n + 1)/\ell$. (Note that we are making use of our assumption that time is discrete; thus, job arrivals and preemptions occur at integer time steps.) An alternative schedule for this instance is to schedule the first two unit-size jobs in sequence on one of the processors and the size-ℓ job on the other processor so that the sequence of pairs of unit-size jobs that are released from time ℓ onwards can be scheduled immediately upon arrival without delaying the size-ℓ job. The total stretch of this schedule is $2n + 4$. Therefore, the competitive ratio of the given online algorithm is at least $(2n\ell+3\ell+n+1)/(2n\ell+4\ell)$, which tends to $(2\ell + 1)/(2\ell)$ as n tends to infinity.

We now choose integer ℓ so as to maximize the minimum of $4\ell/(3\ell + 1)$ and $(2\ell + 1)/(2\ell)$. A maximum of 7/6 is achieved at $\ell = 3$.

We next generalize the above proof to $2m$-processor machines, for $m > 1$. Again, we consider two instances of the scheduling problem. One instance consists of $3m$ jobs, $2m$ jobs of size 1 and m jobs of size $\ell > 1$, all of which are released at time 0. The parameter ℓ is an integer, whose value is specified below. The optimal schedule for this instance is to schedule the $2m$ unit-size jobs in parallel on each of the $2m$ processors and then schedule the size-ℓ jobs on any m of the processors, thus yielding a total stretch of $3m + m/\ell$. Let k denote the number of size-ℓ jobs that the given online algorithm completes by time ℓ. It follows that at least k of the $2m$ unit-size jobs are delayed by a time unit and incur a stretch of 2. Furthermore, at least $m - k$ size-ℓ jobs are delayed by unit time. Thus the total stretch of the online algorithm is at least $3m + k + (m - k)/\ell$.

The second instance for which we evaluate the given online algorithm consists of the same set of $3m$ jobs as in the first instance, together with a sequence of $2nm$ unit-size jobs, arriving in groups of $2m$ jobs at each of the steps $\ell, \ell+1, \ldots, \ell+n-1$. For this instance, the total stretch objective forces the given online algorithm to schedule all of the $2nm$ jobs ahead of the $m-k$ size-ℓ incomplete jobs at time ℓ. The total stretch of the online algorithm for the second instance is at least $2nm+3m+k+n(m-k)/\ell$. On the other hand, an alternative (optimal) schedule completes the first set of $3m$ jobs (that arrive at time 0) within time ℓ and then schedules the remaining $2nm$ unit-size jobs as they arrive. This schedule has a total stretch of $2nm + 4m$. As n tends to infinity, the ratio $(2nm+3m+k+n(m-k)/\ell)/(2nm+4m)$ tends to $1+(m-k)/(2m\ell)$.

The competitive ratio of the online algorithm is thus at least

$$\begin{aligned}
&\min_k \max\{\frac{3m+k+(m-k)/\ell}{3m+m/\ell}, 1+\frac{m-k}{2m\ell}\}\\
&= \min_k \max\{1+\frac{k-k/\ell}{3m+m/\ell}, 1+\frac{m-k}{2m\ell}\}\\
&= 1+\min_k \max\{\frac{k-k/\ell}{3m+m/\ell}, \frac{m-k}{2m\ell}\}\\
&\geq 1+\frac{1}{2}\min_k \left[\frac{1}{2\ell}+\frac{k}{m}\left(\frac{\ell-1}{3\ell+1}-\frac{1}{2\ell}\right)\right]\\
&\geq 1+\frac{\ell-1}{6\ell+2}.
\end{aligned}$$

(The penultimate step holds since the maximum of two numbers is at least the average of the two. The last step uses the inequality $(\ell-1)/(3\ell+1) \leq 1/2\ell$.) The term $1+(\ell-1)/(6\ell+2)$ tends to $7/6$ as ℓ tends to infinity. □

5. Discussion.

We have studied the online complexity of minimizing average stretch. Our results show that SRPT is attractive for both uniprocessor and multiprocessor cases since it simultaneously achieves optimal performance (up to constant factors) for average response time as well as average stretch. In general, any algorithm aimed at optimizing average response time or average stretch, and SRPT in particular, may starve jobs on worst case inputs. However, for certain applications such as serving web traffic, it appears that the adverse effect of starvations in SRPT is limited [4, 15]. In multiprocessors, a potential drawback of SRPT is the overhead it incurs due to migrations. Recall that SRPT may preempt a job at one processor and later resume the job at a different processor. A natural variant of SRPT that does not perform any migrations has been proposed in [2]. In recent work [5], it has been shown that the "migrationless" scheduling algorithm of [2] also achieves a constant-factor competitive ratio with respect to average stretch.

Many interesting problems remain open. In particular, the lower bounds may quite possibly be strengthened. Also, it will be of interest to analyze the algorithm that schedules the job i with the smallest $p(i)\rho_t(i)$ at time t; for the special case of two job sizes, this algorithm is optimal as we showed (see Section 4). The average stretch metric may also be studied in other models, such as when preemption is not allowed.

In this paper, we have focused on online scheduling. The complexity of optimizing average stretch offline is open for both uniprocessors and multiprocessors. For uniprocessors, a PTAS is achievable as shown recently in [7, 9]. For multiprocessors, the online constant-factor upper bounds that we have shown yield the best known approximations offline. Whether the offline problem is NP-hard is open, even for the multiprocessor case. The reduction used in the NP-completeness proof for minimizing average response time on 2-processors [10] may be helpful in this regard.

REFERENCES

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for scheduling to minimize average

completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–43, October 1999.

[2] B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev. Minimizing flow time without migration. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 198–205, May 1999.

[3] K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, New York, 1974.

[4] N. Bansal and M. Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proceedings of ACM SIGMETRICS 2001/Performance 2001: Joint International Conference on Measurement & Modeling of Computer Systems*, pages 279–290, June 2001.

[5] L. Becchetti, S. Leonardi, and S. Muthukrishnan. Scheduling to minimize average stretch without migration. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 548–557, January 2000.

[6] 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.

[7] M. Bender, S. Muthukrishnan, and R. Rajaraman. Improved algorithms for stretch scheduling. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 762–771, January 2002.

[8] C. Chekuri and S. Khanna. Approximation schemes for preemptive weighted flow time. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 297–305, 2002.

[9] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 84–93, 2001.

[10] J. Du, J. Y.-T. Leung, and G. H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75:347–355, 1990.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[12] A. Goel, M. Henzinger, S. Plotkin, and E. Tardos. Scheduling data transfers in a network and the set scheduling problem. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 189–197, Atlanta, Georgia, May 1999.

[13] 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.

[14] 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.

[15] 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.

[16] R. Jain. *The Art of Computer Systems Performance Analysis.* John Wiley, New York, 1991.

[17] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In M. Atallah, editor, *Handbook of Algorithms and Theory of Computation.* CRC Press, Boca Raton, Florida, 1999. Chapter 35.

[18] 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.

[19] 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.

[20] M. Mehta and D. J. DeWitt. Dynamic memory allocation for multiple-query workloads. In *Proceedings of the 19th International Conference on Very Large Data Bases*, pages 354–367, August 1993.

[21] 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.

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

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

[24] H. Zhu, B. Smith, and T. Yang. A scheduling framework for Web server clusters with intensive dynamic content processing. Technical Report TRCS98-29, Department of Computer Science, University of California, Santa Barbara, November 1998. Poster at SIGMETRICS 1999.