Semi-clairvoyant Scheduling

Luca Becchetti [1], Stefano Leonardi [1],
Alberto Marchetti-Spaccamela [1]

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Rome, Italy

Kirk Pruhs [2]

Computer Science Department, University of Pittsburgh, 210 South Bouquet Street Pittsburgh , PA 15260, US

Abstract

In [2] it was shown that the obvious semi-clairvoyant generalization of the Shortest Processing Time is $O(1)$-competitive with respect to average stretch on a single machine. In [2] it was left as an open question whether it was possible for a semi-clairvoyant algorithm to be $O(1)$-competitive with respect to average flow time on a single machine. Here we settle this open question by giving a semi-clairvoyant algorithm that is $O(1)$-competitive with respect to average flow time on a single machine. We also show a semi-clairvoyant algorithm on parallel machines that achieves up to constant factors the best known competitive ratio for clairvoyant on-line algorithms. In some sense one might conclude from this that the QoS achievable by semi-clairvoyant algorithms is competitive with clairvoyant algorithms. We finally show that, in contrast to the clairvoyant case, no semi-clairvoyant algorithm can be simultaneously $O(1)$-competitive with respect to average stretch and $O(1)$-competitive with respect to average flow time.

Key words: Scheduling, Average Flow Time, Semi-Clairvoyance, Simultaneous Competitiveness

---

Email addresses: `becchett@dis.uniroma1.it` (Luca Becchetti), `leon@dis.uniroma1.it` (Stefano Leonardi), `alberto@dis.uniroma1.it` (Alberto Marchetti-Spaccamela), `kirk@cs.pitt.edu.` (Kirk Pruhs).

[1] Partially supported by the IST Programme of the EU under contract ALCOM-FT, APPOL II, and by the MIUR Projects "Societa' dell'informazione", "Algorithms for Large Data Sets: Science and Engineering" and "Efficient algorithms for sequencing and resource allocations in wireless networks".

[2] Supported in part by NSF grant CCR-0098752, NSF grant ANIR-0123705, and a grant from the US Air Force.

1 Introduction

As observed in [2], rounding of processing times is a common/effective algorithmic technique to reduce the space of schedules of interest, thus allowing efficiently computable construction of desirable schedules. This motivated the authors of [2] to initiate a study of the quality of service (QoS) that is achievable by *semi-clairvoyant* online scheduling algorithms, which are algorithms that only require approximate knowledge of the initial processing time of each job, on a single machine. In contrast, a *clairvoyant* online algorithm requires exact knowledge of the processing time of each job, while a *nonclairvoyant* algorithm has no knowledge of the processing time of each job. An explicit categorization of what QoS is achievable by semi-clairvoyant algorithms will be useful information for future algorithms designers who wish to use rounding as part of their algorithm design.

We would also like to point out that there are applications where semi-clairvoyance arises in practice. Take for one example web servers. Currently almost all web servers use FIFO scheduling instead of SRPT scheduling even though SRPT is known to minimize average flow time, by far the most commonly accepted QoS measure, and SRPT is known to be 2-competitive with respect to average stretch [12]. The mostly commonly stated reason that web servers use FIFO scheduling is the fear of starvation. In [5] a strong argument is made that this fear of starvation is unfounded (as long as the distribution of processing times is heavily tailed, as it is the case in practice), and that web servers should adopt SRPT scheduling. Within the context of a web server, an SRPT scheduler would service the document request where the size of the untransmitted portion of the document was minimum. However document size is only an approximation of the time required by the server to handle a request, it also depends on other variables such as where the document currently resides in the memory hierarchy of the server, connection time over the Internet and associated delays etc. Furthermore, currently something like a 1/3 of web traffic consists of dynamic documents (for example popular sites such as msnbc.com personalize their homepage so that they send each user a document with their local news and weather). While a web server is constructing the dynamic content, the web server will generally only have approximate knowledge of the size of the final document.

In [2] it is shown that the obvious semi-clairvoyant generalization of the Shortest Processing Time is $O(1)$-competitive with respect to average stretch. Surprisingly, it is shown in [2] that the obvious semi-clairvoyant generalization of SRPT, always running the job where the estimated remaining processing time is minimum, is not $O(1)$-competitive with respect to average flow time. This result holds even if the scheduler is continuously updated with good estimates of the remaining processing time of each job. [2] gives an alternative algorithm that is $O(1)$-competitive with respect to average flow time in this stronger model where the scheduling algorithm is continuously updated with good estimates of the remaining processing time of each job. The obvious question left open in [2] was whether there exists an $O(1)$-competitive semi-clairvoyant algorithm for average flow time. In [2] it is conjectured that such an algorithm exists and some hints are given as to its construction.

In a personal communication, the authors of [2] proposed an algorithm, which we call R. In section 4, we show that in fact R is $O(1)$-competitive with respect to average flow time, thus settling positively the conjecture in [2]. In some sense one might conclude from these results that the QoS achievable by semi-clairvoyant algorithms is competitive with clairvoyant algorithms. We would like to remark that the approach we propose provides constant approximation when the processing times of jobs are known up to a constant factor and it does not use the remaining processing times of jobs.

For minimizing the average flow time on parallel machines we cannot hope for better than a logarithmic competitive ratio as there exist $\Omega(\log P)$ and $\Omega(\log n/m)$ lower bounds on the competitive ratio for clairvoyant algorithms [10]. Asymptotically tight bounds are achieved by the Shortest Remaining Processing Time heuristic (SRPT) [10]. In section 5, we show that a semi-clairvoyant greedy algorithm, which always runs a job from the densest class of jobs, can also achieve these logarithmic competitive ratios.

We now turn our attention back to the single machine case. As mentioned previously, it is known that the clairvoyant algorithm SRPT is optimal with respect to average flow time and is 2-competitive with respect to average stretch. Thus it is possible for a clairvoyant algorithm to be simultaneously competitive in both average flow time and average stretch. In contrast we show in section 6 that no semi-clairvoyant algorithm can be simultaneously $O(1)$-competitive with respect to average stretch and $O(1)$-competitive with respect to average flow time. Thus in this sense one might conclude that the QoS achievable by semi-clairvoyant algorithms is not competitive with clairvoyant algorithms. It is known that it is not possible for a nonclairvoyant algorithm to be $O(1)$-competitive with respect to average flow time [11], although nonclairvoyant algorithms can be log competitive [8,3], and can be $O(1)$-speed $O(1)$-competitive [4,6,7].

2 Preliminaries

An instance consists of n jobs $J_1, \ldots, J_n$, where job J_i has a non-negative integer release time r_i, and a positive integer processing time or length p_i. An online scheduler is not aware of J_i until time r_i. We assume that at time r_i a semi-clairvoyant algorithm learns the class c_i of job J_i, where job J_i is in class k if $p_i \in \left[2^k, 2^{k+1}\right)$. Each job J_i must be scheduled for p_i time units after time r_i. Preemption is allowed, that is, the schedule may suspend the execution of a job and later begin processing that job from the point of suspension, on the same or on a different machine. The completion time C_i^S of a job J_i in the schedule S is the earliest time that J_i has been processed for p_i time units. The flow time F_i^S of a job J_i in a schedule S is $C_i^S - r_i$, and the average flow time is $\frac{1}{n}\sum_{i=1}^n F_i^S$. The stretch of a job J_i in a schedule S is $(C_i^S - r_i)/p_i$, and the average stretch is $\frac{1}{n}\sum_{i=1}^n (C_i^S - r_i)/p_i$.

A job is alive at time t if released by time t but not yet completed by the online scheduler. Alive jobs are distinguished between *partial* and *total* jobs. Partial jobs have already been executed in the past by the online scheduler, while total jobs have never been executed by the online scheduler. Denote by $\delta^A(t)$, $\rho^A(t)$, $\tau^A(t)$ respectively the number of jobs uncompleted in the algorithm A at time t, the number of partial jobs uncompleted in the algorithm A at time t, the number of total jobs uncompleted by the algorithm A at time t. Denote by $V^A(t)$ the remaining volume, or unfinished work, for algorithm A at time t. Subscripting a variable by a restriction on the class k, restricts the variable to only jobs in classes satisfying this restriction. So for example, $V^A_{\leq k, >h}(t)$ is the remaining volume for algorithm A at time t on jobs in classes in the range $(h, k]$.

The following lemma concerning the floor function will be used in the proof.

Lemma 1 *For all x and y, $\lfloor x + y \rfloor \leq \lceil x \rceil + \lfloor y \rfloor$ and $\lfloor x - y \rfloor \leq \lfloor x \rfloor - \lfloor y \rfloor$.*

PROOF. If either x or y is an integer, then the first inequality obviously holds. Otherwise, $\lfloor x + y \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + 1 = \lceil x \rceil + \lfloor y \rfloor$, since by hypothesis both x and y are not integers.

If either x or y is an integer, then the second inequality obviously holds. Otherwise, denoted by $\{x\}$ and $\{y\}$ respectively the fractional parts of x and y, $\lfloor x - y \rfloor = \lfloor x \rfloor - \lfloor y \rfloor$ if $\{x\} - \{y\} \geq 0$, while $\lfloor x - y \rfloor = \lfloor x \rfloor - \lfloor y \rfloor - 1$ if $\{x\} - \{y\} < 0$.

3 Description of algorithm R

The following strategy is used at each time t to decide which job to run. Note that it is easy to see that this strategy will guarantee that at all times each class of jobs has at most one partial job, and we will use this fact in our algorithm analysis.

- If at time t, all of the alive jobs are of the same class k, then run the partial job in class k if one exists, and otherwise run a total job in class k.
- Now consider the case that there are at least two classes with active jobs. Consider the two smallest integers $h < k$ such that these classes have active jobs.
(1) If h contains exactly one total job J_i and k contains exactly one partial job J_j, then run J_j. We say that the *special rule* was applied to class k at this time.
(2) In all other cases run the partial job in class h if one exists, otherwise run a total job in class h.

Observe that it is never the case that a class contains more than one partial job.

4 Analysis of Algorithm R

Our goal in this section is to prove that R is $O(1)$-competitive with respect to flow time.

Lemma 2 *For all schedules A, $\sum_{i=1}^{n} F_i^A = \int_t \delta^A(t)dt$*

Lemma 2 shows that in order to prove that R is $O(1)$-competitive with respect to flow time, it is sufficient to prove that at any time t, $\delta^{\mathrm{OPT}}(t) = \Omega(\delta^R(t))$. We thus fix an arbitrary time t for the rest of the section, and provide a proof that this relation holds at time t. We sometimes omit t from the notation when it should be clear from the context. Any variable that doesn't specify a time, is referring to time t.

We now give a roadmap of the competitiveness proof of R; the proof consists of three main steps:

(1) We first show (Lemma 4) that in order to prove that $\delta^{\mathrm{OPT}} = \Omega(\delta^R)$, it is sufficient to prove that $\delta^{\mathrm{OPT}} = \Omega(\tau^R)$.
(2) We then bound τ^R by showing that

$$\tau^R \leq 2\delta^{\mathrm{OPT}} + \sum_{k_m}^{k_M - 1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right)$$

(3) We complete the proof by proving that $\sum_{k_m}^{k_M - 1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right)$ is at most $2\delta^{\mathrm{OPT}}$.

The proof of the bound of step 1 above hinges on the following Lemma.

Lemma 3 *If at some time t it is the case that R has partial jobs in classes h and k, with $h < k$, then R has a total job in some class in the range $[h, k]$.*

PROOF. Let J_i be the partial job in h and J_j be the partial job in k. It must be the case that R first ran J_j before J_i, otherwise the fact that J_i was partial would have blocked R from later starting J_j. Consider the time s that R first started J_i. If there is a total job in a class in the range $[h, k]$ at time s, then this job will still be total at time t. If there are only partial jobs, then we get a contradiction since R would have applied the special rule at time s and would have not started J_i.

Lemma 4 $\tau^R \geq (\rho^R - 1)/2$.

PROOF. Let $u_c, u_{c-1}, \ldots, u_1$ be the classes with partial jobs. We consider $\lfloor c/2 \rfloor$ disjoint intervals, $[u_c, u_{c-1}]$, $[u_{c-2}, u_{c-3}], \ldots,$. Lemma 4 implies that there is a total job in each interval. Since the intervals are disjoint these total jobs are distinct and the lemma then follows since $\lfloor c/2 \rfloor \geq (c-1)/2$ for integer c.

Before proceeding with the second step of the proof, we need a few definitions and a lemma. Let $\Delta V(s)$ be $V^R(s) - V^{\mathrm{OPT}}(s)$. Let k_m and k_M be the minimum and maximum non-empty class at time t in R's schedule. Let b_k be the last time, prior to t, when algorithm R scheduled a job of class higher than k. Let b_k^- be the time instant just before the events of time b_k happened.

Lemma 5 *For all classes k, $\Delta V_{\leq k}(t) < 2^{k+1}$.*

PROOF. If $b_k = 0$ then obviously $\Delta V_{\leq k}(t) \leq 0$. So assume $b_k > 0$. The algorithm R has only worked on jobs of class $\leq k$ in the interval $[b_k, t)$. Hence $\Delta V_{\leq k}(t) \leq \Delta V_{\leq k}(b_k)$. Further $\Delta V_{\leq k}(b_k) < 2^{k+1}$, since at time b_k^- at most one job of class $\leq k$ was in the system. A job in class $\leq k$ can be in the system at time b_k^- in case that the special rule was invoked on a job of class $> k$ at this time.

We are now ready to return to showing that $\delta^{\mathrm{OPT}} = \Omega(\tau^R)$.

$$
\begin{aligned}
\tau^R &= \sum_{k_m}^{k_M} \tau_k^R \\
&\leq \sum_{k_m}^{k_M} \lfloor \frac{V_k}{2^k} \rfloor
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{k_m}^{k_M} \lfloor \frac{V_k^{\mathrm{OPT}} + \Delta V_k}{2^k} \rfloor \\
&\leq \sum_{k_m}^{k_M} \lceil \frac{V_k^{\mathrm{OPT}}}{2^k} \rceil + \sum_{k_m}^{k_M} \lfloor \frac{\Delta V_{\leq k} - \Delta V_{\leq k-1}}{2^k} \rfloor \\
&\leq 2\delta^{\mathrm{OPT}}_{\geq k_m, \leq k_M} + \sum_{k_m}^{k_M} \lfloor \frac{\Delta V_{\leq k} - \Delta V_{\leq k-1}}{2^k} \rfloor \\
&\leq 2\delta^{\mathrm{OPT}}_{\geq k_m, \leq k_M} + \sum_{k_m}^{k_M} \lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \sum_{k_m}^{k_M} \lfloor \frac{\Delta V_{\leq k-1}}{2^k} \rfloor \\
&= 2\delta^{\mathrm{OPT}}_{\geq k_m, \leq k_M} + \lfloor \frac{\Delta V_{\leq k_M}}{2^{k_M}} \rfloor + \sum_{k_m}^{k_M - 1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right) - \lfloor \frac{\Delta V_{\leq k_m - 1}}{2^{k_m}} \rfloor
\end{aligned}
$$

The fourth and the sixth line follow from the first and the second inequality of lemma 1, respectively.

Now observe that $\lfloor \frac{\Delta V_{\leq k_M}}{2^{k_M}} \rfloor \leq \delta^{\mathrm{OPT}}_{>k_M}$. In fact lemma 5 implies that $\lfloor \frac{\Delta V_{\leq k_M}}{2^{k_M}} \rfloor \leq 1$; moreover observe that if $\Delta V_{\leq k_M} > 0$ then $\Delta V_{>k_M} < 0$ and, hence, $\delta^{\mathrm{OPT}}_{>k_M} \geq 1$. Also note that

$$
-\lfloor \frac{\Delta V_{\leq k_m - 1}}{2^{k_m}} \rfloor = -\lfloor \frac{V^R_{\leq k_m - 1} - V^{\mathrm{OPT}}_{\leq k_m - 1}}{2^{k_m}} \rfloor \leq \lceil \frac{V^{\mathrm{OPT}}_{\leq k_m - 1}}{2^{k_m}} \rceil \leq \delta^{\mathrm{OPT}}_{\leq k_m - 1}
$$

where the first inequality follows since $V^R_{\leq k_m - 1} \geq 0$ and the last inequality follows since, for each k, we have $\delta^{\mathrm{OPT}}_k \geq \lceil \frac{V_k^{\mathrm{OPT}}}{2^{k+1}} \rceil$.

It follows that

$$
\begin{aligned}
\tau^R &\leq 2\delta^{\mathrm{OPT}}_{\geq k_m, \leq k_M} + \lfloor \frac{\Delta V_{\leq k_M}}{2^{k_M}} \rfloor + \sum_{k_m}^{k_M - 1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right) - \lfloor \frac{\Delta V_{\leq k_m - 1}}{2^{k_m}} \rfloor \\
&\leq 2\delta^{\mathrm{OPT}}_{\geq k_m, \leq k_M} + \delta^{\mathrm{OPT}}_{>k_M} + \sum_{k_m}^{k_M - 1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right) + \delta^{\mathrm{OPT}}_{\leq k_m - 1} \\
&\leq 2\delta^{\mathrm{OPT}} + \sum_{k_m}^{k_M - 1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right)
\end{aligned}
$$

Our final goal will be to show that $\sum_{k_m}^{k_M - 1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right)$ is at most $2\delta^{\mathrm{OPT}}$. From this it will follow that, $\tau^R \leq 4\delta^{\mathrm{OPT}}$. We say that R is *far behind* on a class k if $\Delta V_{\leq k}(t) \geq 2^k$. Notice that for any class k on which R is not far behind, the term $\left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right)$ is not positive. If it happened to be the case that R was not far behind on any class, then we would essentially be done. We thus now turn to characterizing those classes where R can be far behind. In order to accomplish this, we need to introduce some definitions.

Definition 6 *For a class k and a time t, define by $s_k(t)$ the last time before time t when the special rule has been applied to class k. If $s_k(t)$ exists and R has only executed jobs of class $< k$ after time $s_k(t)$, then k is a* special class *at time t.*

By the definition of special class it follows:

Lemma 7 *If class k is special at time t then the special rule was never applied to a class $\geq k$ in $(s_k(t), t]$.*

Let $u_1 < \ldots < u_a$ be the special classes in R at time t. Let f_i and s_i be the first and last times, respectively, that the special rule was applied to an uncompleted job of class u_i in R. Let l_i be the unique class $< u_i$ that contains a (total) job at time s_i. We say that l_i is *associated* to u_i. Note that at time s_i, R contains a unique job in l_i, and that this job is total at this time. A special class u_i is *pure* if $l_{i+1} > u_i$, and is *hybrid* if $l_{i+1} \leq u_i$. The largest special class is by definition pure. Lemma 8 states that the special rule applications in R occur in strictly decreasing order of class. Lemma 9 states that R can not be far behind on pure classes.

Lemma 8 *For all i, $1 \leq i \leq a-1$, $s_{i+1} < f_i$.*

PROOF. At any time $t \in [f_i, s_i]$ the schedule contains a partial job in class u_i and a total job in a class $l_i < u_i$. This implies that the special rule cannot be applied at any time $t \in [f_i, s_i]$ to a class $u_{i+1} > u_i$. Moreover, after time s_i, only jobs of class $< u_i$ are executed.

Lemma 9 *For any pure special class u_k, $\Delta V_{\leq u_k}(t) \leq 0$.*

PROOF. If $b_{u_k} = 0$ then the statement is obvious, so assume that $b_{u_k} > 0$. Notice that no job of class $\leq u_k$ was in the system at time $b_{u_k}^-$. Otherwise, it would have to be the case that $b_k = s_{k+1}$ and $u_k \geq l_{k+1}$, which contradicts the fact that u_k is pure. We can then conclude that $\Delta V_{\leq u_k}(t) \leq \Delta V_{\leq u_k}(b_{u_k}) \leq 0$.

We now show that R can not be far behind on special classes where it has no partial job at time t.

Lemma 10 *If the schedule R is far behind on some class k falling in a maximal interval $[u_b < \ldots < u_c]$, where u_c is pure, and where $[u_b < \ldots < u_{c-1}]$ are hybrid, then one of the following cases must hold:*

(1) $k = u_i$*, where* $b \le i \le c-1$*, where* $l_{i+1} = u_i$*, and* R *has a partial job in* u_i *at time* t*, or*
(2) $k = l_b$*.*

PROOF. It is enough to show that if $k \neq l_b$ then 1 has to hold.

First note that since R is far behind on class k, it must be the case that $b_k > 0$. If R had no alive jobs of class $\le k$ at time b_k^- then $\Delta V_{\le k}(t) \le \Delta V_{\le k}(b_k) \le 0$. Since this is not the case, R was running a job in a special class u_i at time b_k, and $l_i \le k$. By the definition of b_k it must be the case that $b_k = s_i$. If $l_i < k$ then $\Delta V_{\le k}(t) \le \Delta V_{\le k}(b_k) < 2^{l_i+1} \le 2^k$. Therefore R would not be far behind on class k. Thus we must accept the remaining alternative that $k = l_i$.

However, after s_i, R only executed jobs of class at most l_i. This, together with $f_{i-1} > s_i$, implies $u_{i-1} \le l_i$. Furthermore, $u_{i-1} \ge l_i$ since u_{i-1} is hybrid. Thus, $u_{i-1} = l_i$.

We also need the following property.

Lemma 11 *If* R *is far behind on a hybrid class* u_i *then it must have a partial job of class* u_i *at time* t*.*

PROOF. Assume by contradiction that R is far behind on class u_i but it has no partial job of class u_i at t. It must be the case that R completed a job at time s_i. At s_i there were exactly one total job of class l_i and one partial job of class u_i and no jobs in classes between l_i and u_i. Hence $\Delta V_{u_i}(s_i) \le 0$ and $\Delta V_{\le u_i}(s_i) = \Delta V_{\le l_i}(s_i)$. Since by the definition of R, R didn't run a job of class $\ge u_i$ after time s_i, it must be the case that $\Delta V_{\le u_i}(t) \le \Delta V_{\le u_i}(s_i) = \Delta V_{\le l_i}(s_i) < 2^{l_i+1} \le 2^{u_i}$.

We now essentially analyze R separately for each of the maximal subsequences defined above. Lemma 12 establishes that in the cases where $k = u_i$, OPT has an unfinished job in between any two such classes. Hence, lemma 12 associates with class u_i on which R is far behind, a unique job that OPT has unfinished at time t. Lemma 13 handles cases where $k = l_b$ by observing that OPT has at least one unfinished job in classes in the range $[l_b + 1, u_c]$. Hence, lemma 13 associates with each class l_b on which R is far behind, a unique job that OPT has unfinished at time t. From this we can conclude that the number of classes on which R is far behind, and hence the value of the term $\sum_{k_m}^{k_M - 1} \left(\lfloor \frac{\Delta V_{\le k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\le k}}{2^{k+1}} \rfloor \right)$ is at most $2\delta^{\mathrm{OPT}}(t)$.

Lemma 12 *Consider a hybrid class u_i, $b \le i < c$, on which R is far behind. Let u_j be the smallest special class larger than u_i containing a partial job at time t, for some $j \le c$. If no such class exists then let $u_j = u_c$. Then at time t,* OPT *has at least one unfinished job in classes in the range $[u_i + 1, u_j]$.*

PROOF. By Lemma 11, R has a partial job unfinished in u_i at time t. Now, two cases are possible.

(1) If $u_j = u_c$, then by Lemma 9 we have $\Delta V_{\le u_c}(t) \le 0$. On the other hand, $\Delta V_{\le u_i}(t) \ge 2^{u_i}$, since R is far behind on u_i. Hence, it has to be the case that $\Delta V_{>u_i, \le u_c} \le -2^{u_i}$, that is, OPT has at least one job in the interval $[u_i + 1, u_c]$.

(2) Assume $u_j < u_c$. The partial job of class u_j is present over the whole interval $[s_j, t]$. This, together with Lemma 3, implies that R has a total job in a class in the range $[u_i + 1, u_j]$ at time t.

Now, let $k > u_i$ be the smallest class for which R contains a total job J at time t. Assume by contradiction that OPT does not have any unfinished job in $[u_i + 1, k]$. Hence, $\Delta V_{\le k, \ge u_i+1} \ge 2^k$ and, since R is far behind on u_i, it is also far behind on k.

As a consequence of this fact, if $k \in [u_i + 1, u_j - 1]$ we reach a contradiction, since by Lemmas 10 and 11, it has to be the case that k is hybrid and has a partial job in k, against the hypothesis that u_j is the first such class following u_i.

Now, if $k = u_j$, then J is of class u_j and it was released after time s_j, by definition of s_j. So, R only worked on jobs of class strictly less than u_j in $(s_j, t]$, while OPT completed J in the same interval. This and Lemma 5 imply $\Delta V_{<u_j}(t) \le \Delta V_{<u_j}(s_j) - 2^{u_j} < 0$.

If $u_i + 1 < u_j$, $\Delta V_{<u_j}(t) = \Delta V_{\le u_i}(t) + \Delta V_{>u_i, <u_j}(t) < 0$ and $\Delta V_{\le u_i}(t) \ge 2^{u_i}$ imply that OPT has at least one job in classes belonging to interval $[u_i + 1, u_j - 1]$ at t. Instead, if $u_i + 1 = u_j$, we have a contradiction, since R is far behind on u_i.

Lemma 13 *If $\Delta V_{\le l_b}(t) \ge 2^{l_b}$, then at time t,* OPT *has at least one unfinished job in some class in the range $[l_b + 1, u_c]$.*

PROOF. From $\Delta V_{\le u_c}(t) \le 0$ (Lemma 9) and $\Delta V_{\le l_b}(t) \ge 2^{l_b}$, it follows $\Delta V_{>l_b, \le u_c}(t)$
< 0. That is, R is ahead of OPT on jobs in classes $(l_b, u_c]$ at time t. This means that at time t, OPT has an unfinished job in classes $(l_b, u_c]$.

We now know that the value of the term $\sum_{k_m}^{k_M - 1} \left(\lfloor \frac{\Delta V_{\le k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\le k}}{2^{k+1}} \rfloor \right)$ is at most $2\delta^{\mathrm{OPT}}$. Hence, $\tau^R \le 4\delta^{\mathrm{OPT}}$. By applying lemma 4, which states that $\tau^R \ge$

$(\rho^R - 1)/2$, we can conclude that $\rho^R \leq 8\delta^{\mathrm{OPT}} + 1$. Then $\delta^R = \tau^R + \rho^R \leq 12\delta^{\mathrm{OPT}} + 1$. Hence, we get the desired theorem.

Theorem 14 *Algorithm R is at most 13-competitive for the problem of minimizing the average flow time.*

5 Semi-clairvoyant Scheduling on Parallel Machines

In this section we present an algorithm that achieves tight logarithmic bounds even in the semi-clairvoyant case. The class of a job is defined as in the single machine case, i.e. a job is of class j if its processing time is in the range $[2^j, 2^{j+1})$. Algorithm Lowest Class First (LCF) gives precedence to jobs of lower class, always breaking ties in favor of partial jobs. The proof of the following two theorems follows almost the same lines as the proof of competitiveness of SRPT presented in [9] and is therefore omitted.

Theorem 15 *Algorithm LCF is $O(\log P)$-competitive.*

Theorem 16 *Algorithm LCF is $O(\log n/m)$-competitive.*

6 Simultaneous Competitiveness

Theorem 17 *No semi-clairvoyant algorithm A can be both $O(1)$-competitive with respect to average stretch and $O(1)$-competitive with respect to average flow time.*

PROOF. Assume that A is s-competitive with respect to average stretch. The lower bound construction is divided into stages. Stage i starts at time b_i. During stage i one job J_i of class c_i is released. Stage 0 starts at time $b_0 = 0$ when a job J_0 of class m is released. Without loss of generality A runs J_0 from time 0 until time 2^m. Stage 1 starts at time $b_1 = 2^m$. We maintain the invariant that at the start of a stage $i \geq 1$, A has not finished any jobs released in the previous stages, and A knows no lower bound on the remaining processing time of the jobs released in the previous stages.

We now describe stage $i \geq 1$. At the start of stage i a job J_i of class $c_i = c_{i-1} - \log(4s(i+1))$ is released. Stage $i+1$ begins at the first time that A has run J_i for at least 2^{c_i} time units. We now argue $b_{i+1} \leq b_i + 2s(i+1)2^{c_i}$. Assume to reach a contradiction that this is not the case. We then set $p_i = 2^{c_i}$. The stretch of J_i is a least the length of stage i, which is at least $2s(i+1)2^{c_i}$, divided by p_i. That is, the stretch of J_i is at least $2s(i+1)$. Since $i+1$ jobs have been released by stage i, the average stretch is then at least $2s$. This contradicts the assumption that A is s-competitive with respect to average stretch.

In order to be able to continue this construction indefinitely we need to argue that the duration $b_{i+1} - b_i$ of stage i is at most $2^{c_{i-1}}/2$. By the previous paragraph we know that $b_{i+1} - b_i < 2s(i+1)2^{c_i}$. Using the definition of c_i we can conclude that $b_{i+1} - b_i < 2s(i+1)2^{c_{i-1} - \lg(4s(i+1))}$. By algebraic simplification we get the desired result that $b_{i+1} - b_i < 2^{c_{i-1}}/2$.

The adversary's schedule is exactly the same as A's schedule up until the last stage k. During stage k, while A is running J_k, the adversary finishes all of the jobs released in previous stages. We set the processing times of the jobs so that at the end of stage k, A has k jobs unfinished, each with p_k/k remaining processing time. By releasing jobs of length p_k/k every p_k/k time units over a long time period T, A's total flow time will approach kT, while the adversary's total flow time will approach T. This contradicts the assumption that A is $O(1)$ competitive with respect to average flow time.

Acknowledgments

We want to thank Michael Bender, S.Muthukrishnan and Rajmohan Rajaram for sharing their algorithm with us. We also thank Nikhil Bansal and an anonymous referee for helpful suggestions.

References

[1] B. Awerbuch, Y. Azar, S. Leonardi and O. Regev. "Minimizing the flow time without migration", ACM Symposim on Theory of Computing, pp. 198-205, 1999.

[2] M. Bender, S. Muthukrishnan, and R. Rajaraman, "Improved algorithms for stretch scheduling", ACM/SIAM Symposium on Discrete Algorithms, 762 – 771, 2002.

[3] L. Becchetti, and S. Leonardi, "Non-Clairvoyant Scheduling to Minimize the Average Flow Time on Single and Parallel Machines" ACM Symposium on Theorey of Computing, 2001.

[4] P. Berman and C. Coulston, "Speed is more powerful than clairvoyance", *Nordic Journal of Computing*, **6**(2), 181 – 193, 1999.

[5] M. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection Scheduling in Web Servers", in Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems (USITS '99), October 1999.

[6] J. Edmonds, "Scheduling in the dark", *Theoretical Computer Science*, **235**(1), 109 – 141, 2000.

[7] B. Kalyanasundaram, and K. Pruhs, "Speed is as powerful as clairvoyance", *Journal of the ACM*, **47**(4), 617 – 643, 2000.

[8] B. Kalyanasundaram, and K. Pruhs, "Minimizing flow time nonclairvoyantly", IEEE Symposium on Foundations of Compter Science, 1997.

[9] S. Leonardi, "A simpler proof of preemptive total flow time approximation on parallel machines", To appear in *Approximation and Online Algorithms*, E. Bampis eds, Lecture Notes in computer Science, Springer-Verlag, 2003. http://www.dis.uniroma1.it/~leon/

[10] S. Leonardi and D. Raz. "Approximating total flow time on parallel machines", ACM Symposium on Theory of Computing, 110–119, El Paso, Texas, 1997.

[11] R. Motwani, S. Phillips, and E. Torng, "Non-clairvoyant scheduling", *Theoretical Computer Science*, **130**, 17 – 47, 1994.

[12] S. Muthukrishnan, R. Rajaraman, R. Shaheen, and J. Gehrke, "Online scheduling to minimize average stretch", IEEE Symposium on Foundations of Compter Science, 433 – 442, 1997.

[13] C. Phillips, C. Stein, E. Torng, and J. Wein "Optimal time-critical scheduling via resource augmentation", *Algorithmica*, **32**, 163–200, 2002.