A Simpler Proof of Preemptive Total Flow Time Approximation on Parallel Machines*

Stefano Leonardi

Dipartimento di Informatica e Sistemistica,
University of Rome "La Sapienza"
leon@dis.uniroma1.it

Abstract. We consider the classical problem of scheduling jobs in a multiprocessor setting in order to minimize the flow time (total time in the system). The performance of the algorithm, both in offline and online settings, can be significantly improved if we allow preemption: i.e., interrupt a job and later continue its execution. Preemption is inherent to make a scheduling algorithm efficient [7,8]. Minimizing the total flow time on parallel machines with preemption is known to be NP-hard on $m \geq 2$ machines. Leonardi and Raz [8] showed that the well known heuristic *shortest remaining processing time* (SRPT) performs within a logarithmic factor of the optimal offline algorithm on parallel machines. It is not known if better approximation factors can be reached and thus SRPT, although it is an online algorithm, becomes the best known algorithm in the off-line setting. In fact, in the on-line setting, Leonardi and Raz showed that no algorithm can achieve a better bound.

In this work we present a nicer and simpler proof of the approximation ratio of SRPT. The proof presented in this paper combines techniques from the original paper of Leonardi and Raz [8] with those presented in a later paper on approximating total flow time when job preemption but not job migration is allowed [2] and on approximating total flow time non-clairvoyantly [3], that is when the processing time of a job is only known at time of completion.

1 Introduction

One of the most basic performance measures in multiprocessor scheduling problems is the overall time the jobs are spending in the system. This includes the delay of waiting for service as well as the actual service time. This measure captures the overall quality of service of the system. We consider the classical problem of minimizing the total flow time in a multiprocessor setting with jobs released over time. More formally, we consider a set of n jobs and m identical parallel machines. Every job j has processing time p_j and release time r_j. The

* Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT), IST-2001-32007 (APPOL2), and by the MURST Projects "Algorithms for Large Data Sets: Science and Engineering" and "Resource Allocation in Computer Networks".

flow time of a job is the time interval it spends in the system between release and completion. The total flow time of the set of jobs is the sum of the individual flow times of the n jobs.

The performance of the algorithm, both in offline and online settings, can be significantly improved if we allow preemption: i.e., interrupt a job and later continue its execution, perhaps migrating it to a different machine. As shown below, preemption is inherent to make a scheduling algorithm efficient. In the non-preemptive case it is impossible to achieve a "reasonable" approximation. Specifically, even for one machine one cannot achieve an approximation factor of $O(n^{\frac{1}{2}-\epsilon})$ unless $NP = P$ [7]. For $m > 1$ identical parallel machines it is impossible to achieve an approximation factor of $O(n^{\frac{1}{3}-\epsilon})$ unless $NP = P$ [8]. Thus, preemptions really seem to be essential.

Minimizing the flow time on one machine with preemption can be done optimally in polynomial time using the natural algorithm shortest remaining processing time (SRPT) [4]. For more than one machine the preemptive problem becomes NP-hard [5]. Only very recently, Leonardi and Raz [8] showed that SRPT achieves logarithmic approximation for the multiprocessor case, showing a tight bound of $O(\log(\min\{n/m, P\}))$ on $m > 1$ machines with n jobs, where P denotes the ratio between the processing time of the longest and the shortest jobs. In the offline setting, it is not known if better approximation factors can be reached in polynomial time. In fact, in the on-line setting SRPT is optimal, i.e., no algorithm can achieve a better bound up to a constant factor[8]. For the easier problem of minimizing the total completion time a constant approximation and even a PTAS can be obtained [6,1]

The analysis of SRPT we report in this paper is still based on the ideas from the original work of Leonardi and Raz [8]. In a later paper Awerbuch, Azar, Leonardi and Regev [2] presented an algorithm that achieves an $O(\log P)$ and an $O(\log n)$ approximation factor for the problem of minimizing total flow time with preemption when job migration is not allowed, i.e. preempted jobs must be resumed later on the same machine they were run at the time of preemption. The analysis of the algorithm proposed in [2] borrows several ideas from [8] however classifyng jobs into classes allows to remove several difficulties from the analysis. The proof of the $O(\log P)$ approximation for SRPT follows the lines of the proof in [2].

In a recent paper Becchetti and Leonardi [3] consider non-clairvoyant scheduling algorithms to minimize the total flow time. In the non-clairvoyant scheduling problem the existence of a job, but not its processing time, is known at time of release. The processing time of the job is only known when the job is actually completed. In [3], a randomized variation of the Multi-level-feedback algorithm, widely used for processor scheduling in time sharing operating system such as Unix and Windows NT, is proved to achieve a tight $O(\log n)$ competitive ratio for a single machine system and an $O(\log n \log \frac{n}{m})$ competitive ratio for a parallel machine system. A remarkable simplification of the proof of the $O(\log n)$ approximation factor for SRPT draws ideas from the techniques introduced in [3] and from a technical Lemma presented in [2].

2 The Model

We are given a set J of n jobs and a set of m identical machines. Each job j is assigned a pair (r_j, p_j) where r_j is the release time of the job and p_j is its processing time. In the preemptive model a job that is running can be preempted and continued later on any machine. The scheduling algorithm decides which of the jobs should be executed at each time. Clearly a machine can process at most one job at any given time and a job cannot be processed before its release time. For a given schedule define C_j to be the completion time of job j in this schedule. The flow time of job j for this schedule is $F_j = C_j - r_j$. The total flow time is $\sum_{j \in J} F_j$. The goal of the scheduling algorithm is to minimize the total flow time for each given instance of the problem. In the off-line version of the problem all the jobs are known in advance. In the on-line version of the problem each job is introduced at its release time and the algorithm bases its decision only upon the jobs that were already released.

Shortest Remaining Processing Time (SRPT) schedules at any time those jobs with shortest remaining processing time for a maximum number of m. It follows that a job preemption happens only when a newly released job has processing time shorter than the remaining processing time of a job on execution. When a job is completed, that job with shortest remaining processing time currently not assigned to any machine, if any, is scheduled. This results in at most n preemptions operated by SRPT along the execution of the algorithm.

For a given input instance J and a scheduling algorithm S, we denote by $F^S(J)$ the total flow time of the schedule computed by S on input J. Denote by $F^{OPT}(J)$ the minimum value of the total flow time on input istance J. A schedule S is c-approximate if for every input instance J, $F^S(J) \leq cF^{OPT}(J)$. In the following we will omit J when clear from the context.

3 Analysis of SRPT

We denote by A the scheduling algorithm $SRPT$ and by OPT the optimal off-line algorithm that minimizes the flow time for any given instance. Whenever we talk about time t we mean the moment after the events of time t happened. A job is called *alive* at time t for a given schedule if it has already been released but has not been completed yet. Our algorithm classifies the jobs that are alive into classes according to their remaining processing times. A job j whose remaining processing time is in $[2^k, 2^{k+1})$ is in *class* k for $-\infty < k < \infty$. Notice that a job changes its class during its execution. We denote the class of a job upon arrival as the *initial class* of a job.

For a given scheduling algorithm S we define $V^S(t)$ to be the volume of a schedule at a certain time t. This volume is the sum of all the remaining processing times of jobs that are alive. In addition, we define $\delta^S(t)$ to be the number of jobs that are alive. $\Delta V(t)$ is defined to be the volume difference between our algorithm and the optimal algorithm, i.e., $V^A(t) - V^{OPT}(t)$. We also define by $\Delta\delta(t) = \delta^A(t) - \delta^{OPT}(t)$ the alive jobs difference at time t between

A and OPT. For a generic function f (V, $\mathit{\Delta} V$, δ or $\mathit{\Delta}\delta$) we use $f_{\geq h, \leq k}(t)$ to denote the value of f at time t when restricted to jobs of classes between h and k. Similarly, the notation $f_{=k}(t)$ will represent the value of function f at time t when restricted to jobs of class precisely k.

Let $\gamma^S(t)$ be the number of non-idle machines at time t. We denote by $\mathcal{T}$ the set of times in which $\gamma^A(t) = m$, that is, the set of times in which none of the machines is idle. We indicate with $\mathcal{S}$ also the size $\int_{t\in\mathcal{S}} dt$ of a set of times $\mathcal{S}$. Denote by P_{min} the processing time of the shortest job and by P_{max} the processing time of the longest job and $P = P_{max}/P_{min}$. Denote by $k_{min} = \lfloor \log P_{min} \rfloor$ and $k_{max} = \lfloor \log P_{max} \rfloor$ the classes of the shortest and longest jobs upon their arrival, that is the maximum and the minimum initial class of a job.

We start by observing the simple fact that the flow time is the integral over time of the number of jobs that are alive (for example, see [8]):

Fact 1. *For any scheduler S,*

$$F^S = \int_t \delta^S(t)dt.$$

The following is an obvious lower bound on the flow time of any schedule:

Lemma 1. $F^S \geq \sum_j p_j$.

Lemma 2. *There are at most $2 + \log P$ initial classes for a job.*

Proof. The number of initial classes of a job is at most $k_{max} - k_{min} + 1 \leq 2 + \log P$.

The proof of the following lemma is straightforward since the total time spent by the m machines processing jobs is excatly $\sum_j p_j$.

Lemma 3. $\int_t \gamma^A(t)dt = \sum_j p_j$.

Now, assume that $t \in \mathcal{T}$ and let $\hat{t} < t$ be the earliest time for which $[\hat{t}, t) \subset \mathcal{T}$. We denote the last time in which a job of class more than k was processed by t_k. In case such jobs were not processed at all in the time interval $[\hat{t}, t)$ we set $t_k = \hat{t}$. So, $\hat{t} \leq t_{k_{max}} \leq t_{k_{max-1}} \leq ... \leq t_{k_{min}} \leq t$.

Lemma 4. *For $t \in \mathcal{T}$, $\mathit{\Delta} V_{\leq k}(t) \leq \mathit{\Delta} V_{\leq k}(t_k)$.*

Proof. Notice that in the time interval $[t_k, t)$, algorithm A is constantly processing on all the machines jobs whose class is at most k. The off-line algorithm may process jobs of higher classes. Moreover, that can cause jobs of class more than k to actually lower their classes to k and below therefore adding even more to $V^{OPT}_{\leq k}(t)$. Finally, the release of jobs of class $\leq k$ in the interval $[t_k, t)$ is not affecting $\mathit{\Delta} V_{\leq k}(t)$. Therefore, the difference in volume between the two algorithms cannot increase between t_k and t.

Lemma 5. *For $t \in \mathcal{T}$, $\Delta V_{\leq k}(t_k) \leq m2^{k+1}$.*

Proof. First we claim that at any moment $t_k - \epsilon$, for any $\epsilon > 0$ small enough, the algorithm holds $m_1 < m$ jobs whose class is at most k, with total processing time bounded by $m_1 2^{k+1}$. In case $t_k = \hat{t}$, at any moment just before t_k there is at least one idle machine. Otherwise, $t_k > \hat{t}$ and by definition we know that a job of class more than k is processed just before t_k. At time $t_k - \epsilon$, $m_2 \leq m - m_1$ jobs of class $k+1$ may change their class to k. Moreover, at time t_k jobs of class at most k might arrive. However, these jobs increase both $V^{OPT}_{\leq k}(t_k)$ and $V^{A}_{\leq k}(t_k)$ by the same amount, so jobs that arrive exactly at t_k do not change $\Delta V_{\leq k}(t_k)$ and can be ignored. Altogether, we have $\Delta V_{\leq k}(t_k) \leq (m_1 + m_2)2^{k+1} \leq m2^{k+1}$.

Lemma 6. *For $t \in \mathcal{T}$, $\Delta V_{\leq k}(t) \leq m2^{k+1}$.*

Proof. Combining Lemma 4 and 5, we obtain $\Delta V_{\leq k}(t) \leq \Delta V_{\leq k}(t_k) \leq m2^{k+1}$

The claim of the following lemma states a property that will be used in the proof of both the $O(\log P)$ and the $O(\log \frac{n}{m})$ approximation results.

Lemma 7. *For $t \in \mathcal{T}$, for $k_{min} \leq k_1 \leq k_2 \leq k_{max}$, $\delta^{A}_{\geq k_1, \leq k_2}(t) \leq m(k_2 - k_1 + 2) + 2\delta^{OPT}_{\leq k_2}(t)$.*

Proof. $\delta^{A}_{\geq k_1, \leq k_2}(t)$ can be expressed as:

$$
\begin{aligned}
\sum_{i=k_1}^{k_2} \delta^{A}_{=i}(t) &\leq \sum_{i=k_1}^{k_2} \frac{\Delta V_{=i}(t) + V^{OPT}_{=i}(t)}{2^i} \\
&= \sum_{i=k_1}^{k_2} \frac{\Delta V_{\leq i}(t) - \Delta V_{\leq i-1}(t)}{2^i} + \sum_{i=k_1}^{k_2} \frac{V^{OPT}_{=i}(t)}{2^i} \\
&\leq \frac{\Delta V_{\leq k_2}(t)}{2^{k_2}} + \sum_{i=k_1}^{k_2-1} \frac{\Delta V_{\leq i}(t)}{2^{i+1}} - \frac{\Delta V_{\leq k_1-1}(t)}{2^{k_1}} + 2\delta^{OPT}_{\geq k_1, \leq k_2}(t) \\
&\leq 2m + \sum_{i=k_1}^{k_2-1} m + \delta^{OPT}_{\leq k_1-1}(t) + 2\delta^{OPT}_{\geq k_1, \leq k_2}(t) \\
&\leq m(k_2 - k_1 + 2) + 2\delta^{OPT}_{\leq k_2}(t).
\end{aligned}
$$

The first inequality follows since 2^i is the minimum processing time of a job of class i. The third inequality follows since the processing time of a job of class i is less than 2^{i+1}. The fourth inequality is derived by applying Lemma 6, observing that $\Delta V_{\leq k_1-1}(t) \geq -V^{OPT}_{\leq k_1-1}(t)$ and that 2^{k_1} is the maximum processing time of a job of class at most $k_1 - 1$. The claim of the lemma then follows.

The following corollary of Lemma 7 is used in the proof of the $O(\log P)$ approximation ratio of Theorem 2

Corollary 1. *For $t \in \mathcal{T}$, $\delta^A(t) \le m(4 + \log P) + 2\delta^{OPT}(t)$.*

Proof. We write

$$\begin{aligned}\delta^A(t) &= \delta^A_{\le k_{max}, \ge k_{min}}(t) + \delta^A_{<k_{min}}(t)\\ &\le m(k_{max} - k_{min} + 2) + 2\delta^{OPT}(t) + m\\ &\le m(4 + \log P) + 2\delta^{OPT}(t).\end{aligned}$$

The first inequality follows from the claim of Lemma 7 when $k_2 = k_{max}$ and $k_1 = k_{min}$, and from the fact that since a job of class less than k_{min} is never preempted, there are at any time t at most m jobs of class less than k_{min} in the SRPT schedule. The second inequality is obtained since $k_{max} - k_{min} \le \log P + 1$.

Theorem 2. *$F^A \le (6+\log P)\cdot F^{OPT}$, that is, algorithm SRPT has a $(6+\log P)$ approximation factor.*

Proof.

$$\begin{aligned}F^A &= \int_t \delta^A(t)dt\\ &= \int_{t\notin\mathcal{T}} \delta^A(t)dt + \int_{t\in\mathcal{T}} \delta^A(t)dt\\ &\le \int_{t\notin\mathcal{T}} \gamma^A(t)dt + \int_{t\in\mathcal{T}} ((4 + \log P)\gamma^A(t) + 2\delta^{OPT}(t))dt\\ &\le (4 + \log P)\int_t \gamma^A(t)dt + 2\int_{t\in\mathcal{T}} \delta^{OPT}(t)dt\\ &\le (4 + \log P)\sum_j p_j + 2\int_t \delta^{OPT}(t)dt\\ &\le (6 + \log P)F^{OPT}\end{aligned}$$

The first equality derives from the definition of F^A. The second is obtained by looking at the time in which none of the machines is idle and the time in which at least one machine is idle separately. The third inequality uses Corollary 1. The fifth inequality uses Lemma 3, while the sixth inequality follows from Lemma 1.

We now turn to prove the $O(\log \frac{n}{m})$ approximation ratio for $SRPT$. Let $\overline{k}$ be the maximum integer such that for some time $t \in \mathcal{T}$ $\delta^A_{\ge\overline{k}}(t) \ge m$. If not such integer exists, fix $\overline{k} = k_{min} - 1$. Let $\mathcal{T}_j \subseteq \mathcal{T}$, $j = k_{min} + 1, .., \overline{k}$, be the set of time instants when all machines are busy, at least one machine is busy with jobs of class j and no machine is busy with jobs of class higher than j. Let $\mathcal{T}_{\overline{k}+1} \subseteq \mathcal{T}$ be the set of time instants when all machines are busy and at least one machine is processing a job of class higher than $\overline{k}$.

Finally, let $\mathcal{T}_{k_{min}} \subseteq \mathcal{T}$ be the set of time intants when all machines are busy with jobs of class less than or equal to k_{min}. Observe that $\{\mathcal{T}_{k_{min}}, \ldots, \mathcal{T}_{\overline{k}+1}\}$ defines a partition of $\mathcal{T}$. We can write the total flow time of SRPT as:

$$
\begin{aligned}
F^A &= \int_{t \notin \mathcal{T}} \delta^A(t)dt + \int_{t \in \mathcal{T}} \delta^A(t)dt \\
&= \int_{t \notin \mathcal{T}} \gamma^A(t)dt + \sum_{j=k_{min}}^{\overline{k}} \int_{t \in \mathcal{T}_j} \delta^A(t)dt + \int_{t \in \mathcal{T}_{\overline{k}+1}} \delta^A(t)dt \\
&\leq \int_{t \notin \mathcal{T}} \gamma^A(t)dt + \sum_{j=k_{min}}^{\overline{k}} \int_{t \in \mathcal{T}_j} (2m + \delta^A_{\geq j, \leq \overline{k}}(t))dt + \int_{t \in \mathcal{T}_{\overline{k}+1}} 2mdt \\
&\leq \int_{t \notin \mathcal{T}} \gamma^A(t)dt + \sum_{j=k_{min}}^{\overline{k}} \int_{t \in \mathcal{T}_j} (4m + m(\overline{k} - j) + 2\delta^{OPT}_{\leq \overline{k}}(t))dt \qquad (1) \\
&\quad +2 \int_{t \in \mathcal{T}_{\overline{k}+1}} mdt \\
&\leq \int_{t \notin \mathcal{T}} \gamma^A(t)dt + 4\int_{t \in \mathcal{T}} \gamma^A(t)dt + \sum_{j=k_{min}}^{\overline{k}} m(\overline{k} - j)\mathcal{T}_j + 2\int_{t \in \mathcal{T}} \delta^{OPT}(t)dt \\
&\leq 6F^{OPT} + \sum_{j=k_{min}}^{\overline{k}} m(\overline{k} - j)\mathcal{T}_j \qquad (2)
\end{aligned}
$$

where the $\mathcal{T}_|$ in the last two lines denotes the total amount of time that is spent in $calT_j$. The third inequality follows since at any time $t \in \mathcal{T}_j, j = k_{min}, .., \overline{k}+1$, by definition of $\mathcal{T}_j$, there are at most m alive jobs of class less than j and, by definition of $\overline{k}$, at most m alive jobs of class bigger than $\overline{k}$ in the SRPT schedule. The fourth inequality derives by the application of Lemma 7. Finally, the fifth and the sixth inequalities use Lemma 3 and Lemma 1.

We are left to bound the term $F(n) = \sum_{j=k_{min}}^{\overline{k}} m(\overline{k} - j)\mathcal{T}_j$. We show this in the following Lemma:

Lemma 8.

$$F(n) = \sum_{j=k_{min}}^{\overline{k}} m(\overline{k} - j)T_j = O(\log \frac{n}{m})F^{OPT}.$$

Proof. We define $\mathcal{T}_j^l$ for $j \leq k_{min} + 1$ to be the set of time instants in $\{t \geq 0\}$, thus also considering time instants with some machine idle, when machine l is processing a job of class j. $\mathcal{T}^{\updownarrow}_{\|_{\Updownarrow}\rangle\backslash}$ denotes the time in which machine l is processing a job of class k_{min} or less. We first observe that:

$$F(n) = \sum_{j=k_{min}}^{\overline{k}} m(\overline{k} - j)\mathcal{T}_j$$

$$\leq \sum_{j=k_{min}}^{\overline{k}} \sum_{l=1}^{m} (\overline{k}-j)\mathcal{T}_j^l, \tag{3}$$

since every time $t \in \mathcal{T}_j$ is also part of m sets $\mathcal{T}_i^l$ with $i \leq j$.

Let $n_j, j = k_{min}, ..., \overline{k}-1$, be the number of jobs of initial class j in the input instance. Let $n_{\overline{k}}$ be the number of jobs of initial class bigger or equal than $\overline{k}$ released in the input instance. For sake of simplicity, we will refer in the following to jobs of initial class higher than $\overline{k}$ as jobs of initial class $\overline{k}$.

Now, observe that a job of initial class j gives a contribution to equation (3) bounded by $2\sum_{i=0}^{j-k_{min}}(\overline{k}-j+i)2^{j-i}$ since each job of initial class j has been processed for at most 2^i time units when in class $i = k_{min}+1, \ldots, j$. This contribution is, by simple algebraic manipulation, at most equal to $2(\overline{k}-j+1)2^{j+1}$.

We then continue with the following inequalities:

$$\begin{aligned} F(n) &\leq 2\sum_{j=k_{min}}^{\overline{k}} n_j(\overline{k}-j+1)2^{j+1} \\ &= 4\sum_{j=k_{min}}^{\overline{k}} n_j(\overline{k}-j)2^j + 2\sum_{j=k_{min}}^{\overline{k}} n_j 2^{j+1} \\ &\leq 4\sum_{j=k_{min}}^{\overline{k}} n_j 2^j(\overline{k}-j) + 4\sum_j p_j, \end{aligned} \tag{4}$$

since a job of initial class j has processing time at least equal to 2^j. We exchange variable number j with $i = \overline{k}-j$. Let $I_i = n_{\overline{k}-i}2^{\overline{k}-i}, i = 0, \ldots, \overline{k}-k_{min}$. The first term $\sum_{j=k_{min}}^{\overline{k}} n_j 2^j(\overline{k}-j)$ of equation (4) becomes:

$$\sum_{i=0}^{\overline{k}-k_{min}} iI_i. \tag{5}$$

We can derive an upper bound on $F(n)$ by maximizing function (5) subject to the two obvious constraints:

$$\sum_{i=0}^{\overline{k}-k_{min}} I_i \leq \sum_j p_j \tag{6}$$

$$\sum_{i=0}^{\overline{k}-k_{min}} \frac{I_i}{2^{\overline{k}-i}} \leq n. \tag{7}$$

Constraint (7) implies

$$\sum_{i=0}^{\overline{k}-k_{min}} I_i \leq n2^{\overline{k}}. \tag{8}$$

To complete the proof we need the following simple mathematical lemma proved in [2]:

Lemma 9. *Given a sequence $a_1, a_2, ...$ of non-negative numbers such that $\sum_{i\geq 1} a_i \leq A$ and $\sum_{i\geq 1} 2^i a_i \leq B$ then $\sum_{i\geq 1} i a_i \leq \log(4B/A)A$.*

Proof. Define a second sequence, $b_i = \sum_{j\geq i} a_j$ for $i \geq 1$. Then it is known that $A \geq b_1 \geq b_2 \geq ....$ Also, it is known that $\sum_{i\geq 1} 2^i a_i = \sum_{i\geq 1} 2^i(b_i - b_{i+1}) = \frac{1}{2}\sum_{i\geq 1} 2^i b_i + b_1$. This implies that $\sum_{i\geq 1} 2^i b_i \leq 2B$.

The sum we are trying to upper bound is $\sum_{i\geq 1} b_i$. This can be viewed as an optimization problem where we try to maximize $\sum_{i\geq 1} b_i$ subject to $\sum_{i\geq 1} 2^i b_i \leq 2B$ and $b_i \leq A$ for $i \geq 1$. This corresponds to the maximization of a continuous function in a compact domain and any feasible point where $b_i < A, b_{i+1} > 0$ is dominated by the point we get by replacing b_i, b_{i+1} with $b_i + 2\epsilon, b_{i+1} - \epsilon$. Therefore, it is upper bounded by assigning $b_i = A$ for $1 \leq i \leq k$ and $b_i = 0$ for $i > k$ where k is large enough such that $\sum_{i\geq 1} 2^i b_i \geq 2B$. A choice of $k = \lceil \log(2B/A) \rceil$ is adequate and the sum is upper bounded by kA from which the result follows.

We apply Lemma 9 to our problem with variables $a_i = I_i$, $i = 0, \ldots, \overline{k} - k_{min}$, $A = \sum_j p_j$ by constraint (6), $B = n2^{\overline{k}}$ by constraint (8), to obtain:

$$\begin{aligned}\sum_{i=0}^{\overline{k}-k_{min}} iI_i &\leq \log(\frac{4n2^{\overline{k}}}{\sum_j p_j})\sum_j p_j \\ &\leq O(\log \frac{n}{m})F^{OPT},\end{aligned} \tag{9}$$

where the last inequality follows since, by the definition of $\overline{k}$, at some time t there are at least m jobs of class bigger or equal than $\overline{k}$, for which $\sum_j p_j \geq m2^{\overline{k}}$.

Combining equations (4) and (9) and from $\sum_j p_j \leq F^{OPT}$, we obtain the desired bound.

Finally, Lemma 8 together with equation (2) leads to our result:

Theorem 3. *Algorithm SRPT has an $O(\log \frac{n}{m})$ approximation factor.*

4 Conclusions

In this paper we present a simpler proof of the approximation of SRPT for preemptive minimization of the total flow time on parallel identical machines. The proof relies on the original ideas of [8] and on new tools of analysis introduced in later works [2,3]. A major open problem is to devise a constant approximation algorithm or even an approximation scheme for the problem.

References

1. F. N. Afrati, E. Bampis, C. Chekuri, D. R. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein and M. Sviridenko. Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates. Proceedings of the *40th Annual Symposium on Foundations of Computer Science (FOCS 1999)*, pp. 32-44, 1999.
2. B. Awerbuch, Y. Azar, S. Leonardi and O. Regev. Minimizing the flow time without migration. *Proc. of the 31st annual ACM Symposim on Theory of Computing*, pp. 198-205, 1999.
3. L. Becchetti and S. Leonardi. Non-Clairvoyant scheduling to minimize the average flow time on single and parallel machines. *Proc. of the 33rd annual ACM Symposim on Theory of Computing*, pp. 94-103, 2001.
4. K.R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.
5. J. Du, J. Y. T. Leung, and G. H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75(3):347–355, 1990.
6. L. Hall, Andreas S. Schulz, D. Shmoys and J. Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. In *Mathematics of Operations Research* 22, pp. 513-544, 1997.
7. H. Kellerer, T. Tautenhahn and G.J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM Journal on Computing*, Vol. 28, Number 4, pp. 1155-1166, 1999.
8. S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 110–119, El Paso, Texas, 1997. To appear in *Journal of Computer and System Sciences – special issue for STOC '97.*