Nonclairvoyant Scheduling to Minimize the Total Flow Time on Single and Parallel Machines

LUCA BECCHETTI AND STEFANO LEONARDI

University of Rome "La Sapienza", Rome, Italy

Abstract. Scheduling a sequence of jobs released over time when the processing time of a job is only known at its completion is a classical problem in CPU scheduling in time sharing operating systems. A widely used measure for the responsiveness of the system is the average flow time of the jobs, that is, the average time spent by jobs in the system between release and completion.

The Windows NT and the Unix operating system scheduling policies are based on the Multilevel Feedback algorithm. In this article, we prove that a randomized version of the Multilevel Feedback algorithm is competitive for single and parallel machine systems, in our opinion providing one theoretical validation of the goodness of an idea that has proven effective in practice along the last two decades.

The randomized Multilevel Feedback algorithm (RMLF) was first proposed by Kalyanasundaram and Pruhs for a single machine achieving an $O(\log n \log \log n)$ competitive ratio to minimize the average flow time against the on-line adaptive adversary, where n is the number of jobs that are released. We present a version of RMLF working for any number m of parallel machines. We show for RMLF a first $O(\log n \log \frac{n}{m})$ competitiveness result against the oblivious adversary on parallel machines. We also show that the same RMLF algorithm surprisingly achieves a tight $O(\log n)$ competitive ratio against the oblivious adversary on a single machine, therefore matching the lower bound for this case.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: *performance attributes*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sequencing and scheduling*; G.3 [**Probability and Statistics**]: Probabilistic Algorithms

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Probabilistic analysis, flow time, multilevel feedback, randomized algorithms

1. *Introduction*

In this article, we study nonclairvoyant algorithms to schedule a stream of jobs released over time on single and parallel machine systems. Every job j is assigned with a release time $r_j \geq 0$ and a processing time p_j. Job j must be globally scheduled for p_j time units on m parallel identical machines before its completion.

Authors' address: Dipartmento di Informatica e Sistemistica, University of Rome "La Sapienza", Via Salaria 113, 00198 Rome, Italy, e-mail: Luca.Becchetti@dis.uniromal.it.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2004 ACM 0004-5411/04/0700-0517 $5.00

Job preemption is allowed, the execution of a job can be stopped and resumed later on the same or on a different machine. The completion time of job j is denoted by C_j. A *nonclairvoyant* scheduling algorithm knows very little about the input instance: The existence of a job is only known at the release time of the job; The processing time of a job is only known when the job is completed.

This problem has a number of motivating applications. The most classical one is processor scheduling in a time-sharing multitasking operating system, in which scheduling decisions must be taken without knowledge of the time a job needs to be executed. The obvious goal is to provide a fast response to users. Job preemption is widely recognized as a key factor to improve the responsiveness of the system. In multiprocessor computer systems, preemption requires a context switching at a processor but this cost is reasonably small.[1]

A widely accepted measure of the quality of service provided to users is the average response time of the system. The response time, or flow time, of every job is the time spent by the job in the system between release and completion, that is, $C_j - r_j$ for job j. We measure the performance of a randomized nonclairvoyant scheduling algorithm by its competitive ratio [Sleator and Tarjan 1985; Ben-David et al. 1994], the worst-case ratio between the expected average flow time of the algorithm and the optimal average flow time of an *oblivious* adversary that generates the input sequence without knowledge of the random choices of the algorithm.

This problem has been addressed for a couple of decades in the design of time sharing operating systems. Windows NT [Nutt 1999] and Unix [Tanenbaum 1992] have the Multilevel Feedback (MLF) algorithm at the very basis of their scheduling policies. The basic idea of MLF is to organize jobs into a set of queues. Each job is processed for 2^i time units if in queue Q_i, before being promoted to queue Q_{i+1} if not completed. At any time, the machines process jobs in the lowest queues, in each queue giving priority to jobs at the front. While this algorithm turns out to be very effective in practice, it behaves very poorly with respect to a worst-case analysis, as explained below.

A good rule of thumb for flow time minimization is given by the Shortest Remaining Processing Time (SRPT) first rule. SRPT prescribes the preemption of a job on execution when a job with shorter remaining processing time is released. SRPT is indeed an optimal algorithm for a single machine [Baker 1974] and provides the best known approximation for parallel machines [Leonardi and Raz 1997]. However, a nonclairvoyant scheduling algorithm cannot stick to the SRPT rule since it has no knowledge of the processing time of the jobs before they are completed. As to MLF, it behaves on some instances very differently from the SRPT rule in that it may preempt jobs in a high queue that are nearly completed to process newly released jobs with large processing time in lower queues. This may force jobs with small remaining processing time to spend a long time in the system while other long jobs are processed. It has actually been shown that no deterministic nonclairvoyant algorithm can be competitive at all against a worst case adversary [Motwani et al. 1994].

In order to circumvent these difficulties, a randomized version of MLF, called RMLF, was proposed for a single machine by Kalyanasundaram and Pruhs [1997].

[1]This in particular holds in systems that support threads [Doeppner 1987; IEEE 1994; Mueller 1993; SUN 1993].

The idea is to try to approximately behave like SRPT by having a large fraction of jobs with a remaining processing time that is still a sufficiently large share of the initial processing time when entering their respective queues of completion. Of course, this requirement can only be satisfied with some probability. Kalyanasundaram and Pruhs [1997] show that RMLF is $O(\log n \log \log n)$ competitive for minimizing the total flow time on a single machine against the on-line adaptive adversary [Ben-David et al. 1994]. The on-line adaptive adversary may decide its strategy at time t based on the knowledge of the random choices of the algorithm up to time t. However, the processing time of a job must be fixed by the adversary at release time of the job.

In this article, we present a randomized version of the Multilevel Feedback algorithm working for any number m of parallel machines. The algorithm we present is an evolution of the one presented in Kalyanasundaram and Pruhs [1997] for a single machine, for this reason we also call RMLF our algorithm. We present two main results for RMLF:

(1) We show that RMLF has competitive ratio $O(\min\{\log n \log \frac{n}{m}, \log n \log P\})$ against an oblivious adversary, where P is the ratio between the largest and the shortest processing time of a job. This is the first $o(n)$-competitive result for nonclairvoyant average flow time minimization on parallel machines. This compares with the $\Omega(\log \frac{n}{m})$ and $\Omega(\log P)$ lower bounds against the oblivious adversary given in Leonardi and Raz [1997] for the case in which the processing time of a job is known at release time.
(2) For the case of a single machine, we show that our version of RMLF surprisingly matches the $\Omega(\log n)$ lower bound against an oblivious adversary of Motwani et al. [1994].

These two results are obtained as an outcome of a unified analysis of RMLF. They are proved with a probabilistic analysis that, unlike Kalyanasundaram and Pruhs [1997], does not require any high probability argument. In turn, our analysis requires a considerable strengthening and simplification of the tools previously developed in the study of algorithms for minimizing the average flow time [Leonardi and Raz 1997; Awerbuch et al. 2002].

1.1. RELATED RESULTS. Nonclairvoyant scheduling to minimize average flow time has been first studied by Motwani et al. [1994]. The authors prove that any deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$-competitive where n is the number of jobs released, and that every randomized nonclairvoyant algorithm is $\Omega(\log n)$-competitive in the case of a single machine. The authors also present competitive algorithms for the static case where all jobs are released at time 0. Kalyanasundaram and Pruhs [1997] give the first $o(n)$ competitive nonclairvoyant scheduling result for a single machine. They prove that a randomized version of MLF is $O(\log n \log \log n)$ competitive against an on-line adaptive adversary [Ben-David et al. 1994]. They also claim an $\Omega(P)$ randomized lower bound for the problem, where P is the ratio between the maximum and the minimum processing time of a job, on an instance with a number of jobs that is exponential in P. In a previous paper, Kalyanasundaram and Pruhs [1995] study a different model in which the nonclairvoyant algorithm is equipped with a faster processor than the adversary. They prove in this case that shortest elapsed time first is a constant competitive algorithm for a single machine. More recently, Bansal et al. [2003] have applied resource augmentation to nonclairvoyant scheduling to minimize average stretch on parallel machines, proposing an $O(1)$-speed, $O(\log^2 P)$-competitive algorithm, P being the ratio between the largest and smallest job sizes. They also propose an $\Omega(\log P)$ lower bound, which is tight when all jobs are released at time 0. Bansal and Pruhs [2003], finally, among other results also prove that the Shortest Elapsed Time First heuristic is $O(1+\epsilon)$-speed, $O(1/\epsilon^{(2+2/p)})$-competitive to minimize the L_p norm of the flow time, and that with constant speed-up it is poly-logarithmically competitive (with respect to P) for the L_p norm of the stretch. Related to nonclairvoyant scheduling is the work of Bender et al. [2002], where the authors show that a constant-factor competitive ratio for average stretch is achievable even if the processing times (or remaining processing times) of jobs are known only to within a constant factor of accuracy.

Related results are also concerned with the more classical average flow time minimization problem when the processing time of a job is known at release time. Also in this case, if preemption is not allowed, achieving reasonable performances on-line is not possible. If preemption is allowed, the problem can be optimally solved on 1 machine by the well-known SRPT (Shortest Remaining Processing Time) heuristic [Baker 1974]. Leonardi and Raz [1997] proved that SRPT is $O(\log(\min\{n/m, P\}))$-competitive for $m \geq 2$ identical parallel machines. No better approximation, even off-line, is known for this problem. They also prove an $\Omega(\log \frac{n}{m})$ and an $\Omega(\log P)$ lower bound on the competitive ratio of any randomized algorithm for the problem. These lower bounds clearly extend to the nonclairvoyant case on $m \geq 2$ parallel machines. Other related work is concerned with the issues of using preemption without allowing job migration [Awerbuch et al. 2002] and with related objective functions like the stretch metric [Acharya et al. 1999; Bender et al. 1998; Gehrke et al. 1999; Becchetti et al. 2004].

1.2. OUR WORK. We prove that RMLF achieves an $O(\log n \log \frac{n}{m})$ and an $O(\log n \log P)$ randomized competitive ratio against the oblivious adversary for any number m of parallel identical machines. We recall that P is the ratio between the largest and the shortest processing time of a job. Observe that one of the two bounds may be strictly better than the other depending on the values of P, n and m. We do not contradict the $\Omega(P)$ randomized lower bound claimed in Kalyanasundaram and Pruhs [1997] since it is obtained with a number of jobs exponential in P. For a single machine, we show an $O(\log n)$ bound, which is tight. Our algorithm requires the knowledge of the minimum processing time of a job that is released. This knowledge is essential for an MLF-like algorithm to establish how long it should execute a job in the lowest queue when it enters the system. A wrong estimation of the minimum processing time of a job may leave many small jobs waiting much longer than their processing times. This argument can be extended to show that in absence of this information it is not possible to be competitive at all. However, we do not need any knowledge of the maximum processing time of a job that is released. Moreover, we do not need to know the number n of jobs released over the sequence.

Our analysis of RMLF contains a set of new tools of analysis. We mentioned in the introduction that RMLF tries to approximately follow SRPT with the goal of avoiding the preemption of jobs that are nearly finished. We prove that (Lemmas 6 and 7) every job, apart from a logarithmic number, with constant probability, has a remaining processing time that is still a large share of the initial processing time when it enters the queue in which it will be completed. This allows, roughly speaking, to concentrate only on those jobs that are called "big". We bound at any time t the difference between the number of big jobs of almost equal size that are uncompleted in RMLF and in the optimum, by limiting their corresponding volume, that is, their overall remaining processing time. We then prove (Lemma 13) that at any time t, the expected number of jobs not completed by RMLF is at most $O(\log n)$ times a constant fraction of the jobs uncompleted by the adversary plus an additive term $O(\log n(m\ k(t))$, where $k(t)$ is related to the number of nonempty queues at time t in RMLF. The $O(\log n \log P)$ result for parallel machines then follows since there are at most $O(\log P)$ queues in the system.

The proof of $O(\log n \log \frac{n}{m})$ competitiveness for parallel machines and of $O(\log n)$ competitiveness for a single machine requires bounding the additional flow time due to the contribution of the $O(\log n(mk(t))$ term over time in a different way. This contribution is partly originated by the fact that on $m \geq 2$ machines RMLF may keep machines idle for more than the optimum. We will bound this contribution (Section 4.4) by at most $O(\log n \log \frac{n}{m})$ times the optimum for $m \geq 2$ machines while for a single machine we will be able to limit this contribution to $O(\log n)$ times the optimum. These two results, however, are obtained as consequences of a unified analysis of RMLF.

1.3. STRUCTURE OF THE ARTICLE. In Section 2, we formally define our scheduling problem. Section 3 presents the RMLF algorithm. In Section 4, we provide the analysis of the behaviour of RMLF. In particular, in Section 4.1, we provide some definitions and basic facts. In Section 4.2, we present some tools and preliminary results. Section 4.3 is devoted to proving the $O(\log n \log P)$ competitive ratio, while Section 4.4 concerns the $O(\log n \log(n/m))$ competitive ratio of RMLF and the $O(\log n)$ upper bound for the single machine case. Open problems are discussed in Section 5.

2. *Problem Definition*

We are given a set J of n jobs and a set of m identical machines. Each job j is assigned with a pair (r_j, p_j) where r_j is the release time of the job and p_j is its processing time. We order jobs by increasing release times and we assume the first job is released at time $r_1 = 0$. In the preemptive model, a job that is running can be preempted and continued later on any machine. The algorithm decides which of the uncompleted jobs should be executed at each time. A job cannot be processed before its release time. A job cannot be executed in parallel on multiple machines, while only a single job can be processed by a machine at any time. For any given schedule, we define C_j to be the completion time of job j in that schedule. The flow time of job j is $F_j = C_j - r_j$. The total flow time for the input instance J is $F(J) = \sum_{j \in J} F_j$. The goal of the scheduling algorithm is to minimize the total flow time or equivalently the average flow time.

The arrival of a job is unknown to the *nonclairvoyant* algorithm until the job is released. The processing time of a job is unknown to the nonclairvoyant algorithm until the completion time of the job. We assume the minimum processing time of a job that is released to be known in advance to the algorithm, and the convention $\min_{j \in J} p_j = 1$, while we do not assume any knowledge of the maximum processing time of a job.

We compare the randomized nonclairvoyant algorithm with an oblivious adversary that decides in advance, without knowledge of the random choices of the algorithm, the number n of jobs of the sequence together with the release time and the processing time of each job. The adversary is charged with the optimal flow time for the sequence. Denote by A the randomized on-line algorithm, and denote by *OPT* the optimal adversary. The generic (deterministic) execution of A over an input sequence J is defined by a set of random choices σ. We denote by $\Pr(\sigma)$ the probability of sequence σ. A randomized on-line algorithm A is c-competitive against an oblivious adversary if for any input J,

$$E_\sigma[F_\sigma^A(J)] \leq cF^{OPT}(J),$$

where the expectation is taken over any possible outcome of the random choices σ of the algorithm for input instance J. The instance J will be omitted in the following when clear from the context.

3. *The RMLF Algorithm*

We present our randomized version of the Multilevel Feedback algorithm for parallel machines, called RMLF in the following. A job is said *active* or *alive* at time t if released but not completed. Denote by $x_j(t)$ the time job j has been processed until time t. Denote by $y_j(t) = p_j - x_j(t)$ the remaining processing time of job j at time t. $\beta_{j,i}$ is a random variable with distribution $\Pr[\beta_{j,i} \leq x] = 1 - \exp(-\gamma \frac{x}{2^i} \ln j)$. In our algorithm, we choose $\gamma = 4/3$.

Active jobs are partitioned into a set of *Priority Queues* $Q_0, Q_1, \ldots$. Within each queue, the priority is determined by the Earliest Release Time first rule. For any two queues Q_i, Q_k, Q_i is said lower than Q_k if $i < k$. Job j is assigned with a target processing time $T_{j,i}$ when it enters queue Q_i. At any time t, RMLF behaves as follows:

(1) Schedule alive jobs on the m machines proceeding from the lowest to the highest queue, within any queue in order of priority.

(2) Job j released at time t enters queue Q_0 with $T_{j,0} = \max\{1, 2 - \beta_{j,0}\}$.

(3) For a job j in a queue Q_{i-1} at time t, if $x_j^A(t) = T_{j,i-1} < p_j$, job j enters Q_i with $T_{j,i} = \max\{2^i, 2^{i+1} - \beta_{j,i}\}$.

(4) For a job j in a queue Q_i at time t, if $x_j^A(t) = p_j \leq T_{j,i}$, assign $C_j(t) = t$ and discard j from Q_i.

The Earliest Release Time first rule assigns a static priority to jobs, rather than following the dynamic priority defined by the time of arrival in a queue. The reason of this choice is to fix, independently from the specific execution of the algorithm, the relative priority between any two jobs located in the same queue. This will turn out to be very useful in the probabilistic analysis of the algorithm.

We would also like to comment on the dependence of the probability distribution of target $T_{j,i}$ on both i and j. This is necessary since the number n of jobs released in the sequence is not known in advance. If n was known, we could simply replace j with n in the probability distribution of $T_{j,i}$. However, this would not improve the worst case performance of RMLF.

A very natural question at this point of the exposition is: why the exponential distribution rather than the uniform distribution? The following example, due to Kirk Pruhs (2000 personal communication), shows why a uniform distribution of target $T_{j,i} \in [2^i, 2^{i+1})$ in queue Q_i does not work, even against the oblivious adversary.

Consider a single machine and n jobs released at time 0 with processing time $2^i(1 + \frac{2}{\sqrt{n}})$, with $n = 2^i$. With probability at least $\frac{1}{\sqrt{n}}$ every job will enter Q_{i+1} with a remaining processing time at least $\sqrt{n}$ and at most $2\sqrt{n}$. This guarantees that at time $t = n2^i(1 + \frac{2}{\sqrt{n}}) - cn$, for some constant c, the algorithm will have $\Omega(\sqrt{n})$ uncompleted jobs in queue Q_{i+1} with high probability, while the optimal solution will have terminated all jobs with the exception of a constant number, for a total flow time of $O(n^3)$. Starting a time t, a job of size 1 is released for each of n^3 time units. The expected flow time of the algorithm will be $\Omega(n^3\sqrt{n})$, while the optimum is $O(n^3)$.

4. *Analysis*

4.1. PRELIMINARIES. In the analysis, we use the following notation. We denote by $\delta(t)$ the number of active jobs at time t. The volume $V(t)$ is the sum of the remaining processing times of the jobs that are active at time t. $L(t)$ denotes the total work done prior to time t, that is the overall time the m machines have been processing jobs until time t. For any function f (δ, V and L), we denote by $f_\sigma^A(t)$ and $f^{OPT}(t)$ the value of function f at time t, respectively, for a generic outcome σ of the random choices of RMLF and for the optimal solution OPT. By $\Delta_\sigma f(t) = f_\sigma^A(t) - f^{OPT}(t)$, we denote their difference.

In the analysis, without loss of generality, we restrict to input instances in which $r_1 = 0$ and $r_j \leq \sum_{i=1}^{j-1} p_i$, $j > 1$, since otherwise every input instance can be split into a set of input instances of this form that we separately analyze.

In our analysis, we classify active jobs into classes according to their processing times. A job j is of *class* i, $i \geq 0$, if its processing time is in $[2^i, 2^{i+1})$. For a generic function f (V, ΔV, δ, $\Delta\,\delta$, L or ΔL) the notation $f_{=k}(t)$ will denote the value of function f at time t when restricted to jobs of class exactly k; 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.

Denote by $P = \frac{\max_{j\in J} p_j}{\min_{j\in J} p_j}$ the ratio between the maximum and the minimum processing time of a job. The maximum class of a job is $\lfloor \log P \rfloor + 1$. Moreover:

FACT 1. *During the execution of the algorithm at most $\lceil \log P \rceil$ queues are created.*

We start by observing that the total flow time is the integral over time of the number of active jobs (see, e.g., Leonardi and Raz [1997]):

FACT 2. $F = \sum_{j\in J} F_j = \int_t \delta(t)dt$.

Moreover, we have the following basic lower bound to the total flow time:

FACT 3. $F = \sum_{j\in J} F_j \geq \sum_{j\in J} p_j$.

The following fact holds for the algorithm:

FACT 4. *A job j of class i ends either in queue Q_i or in queue Q_{i+1}.*

PROOF. Job j, with processing time $p_j \in [2^i, 2^{i+1})$, has target $T_{j,i-1} < 2^i$ in Q_{i-1} while it has target $T_{j,i+1} \geq 2^{i+1}$ in queue Q_{i+1}. $\square$

Definition 1. Job j of class i is *unlucky* in a given execution of the algorithm if it has processing time $p_j \leq 2^i + 2^{i-2}$ and it ends in queue Q_{i+1}. A job that is not unlucky is said lucky.

We denote by $\delta^l_\sigma(t)$ and by $\delta^u_\sigma(t)$ respectively the number of lucky and unlucky jobs that are active at time t for a specific execution σ of RMLF.

Definition 2. At any time t, a lucky job j in queue Q_i is said *big* if it entered Q_i at time $\bar{t} \leq t$ and $y_j(\bar{t}) \geq \frac{p_j}{\ln j}$, it is said *small* otherwise.

Observe that, according to this definition, a lucky job of class i is always big while in queues Q_{i-1} or lower. We denote by $\delta^b_\sigma(t)$ the number of big (lucky) jobs that are alive at time t for a specific outcome σ of the random choices of RMLF.

Finally, we define a set of random variables that will be used in the analysis of the algorithm. More in detail, for each job j X^l_j has value 1 if job j is lucky, while $X^l_j = 0$ if j is unlucky. Observe that this variable does not depend upon time but only on the specific execution of the algorithm. Moreover, for each job j and for each time t, four binary variables are defined: X^l_j, $X_j(t)$, $X^l_j(t)$ and $X^b_j(t)$. The value of $X_j(t)$ is 1 if job j is alive at time t, 0 otherwise. $X^l_j(t)$ is defined in terms of X^l_j and $X_j(t)$, namely, $X^l_j(t) = X^l_j \cdot X_j(t)$. Finally, $X^b_j(t) = 1$ if $X^l_j(t) = 1$ and job j is big at time t, $X^b_j(t) = 0$ otherwise.

The following gives a basic property of the RMLF algorithm:

FACT 5. *At any time t and for any i, at most m jobs, alive at time t, have been executed in queue Q_i but have not been promoted to Q_{i+1}.*

PROOF. Let $j_1, j_2, \ldots$ be the jobs in Q_i at time t ordered by decreasing priority. Let k be the largest index such that j_k has been processed while in queue Q_i. Let $t' \leq t$ be the last time that j_k was processed in Q_i. We show below that all jobs j_h, $h < k$, are also executed at time t'. Since at most m jobs are executed at the same time, this implies $k \leq m$ and the claim follows. Two cases are possible for a job j_h, $h < k$: (i) if j_h was in queue Q_i at time t', it would be on execution since it precedes job j_k in the earliest released time first order; (ii) if at time t' job j_h was in a queue Q_l, $l < i$, then it would also be on execution since a job is executed in a higher queue Q_i. $\square$

We first present the probabilistic statements we use in the analysis. Then we give the analysis of the competitive ratio for RMLF.

4.2. PROBABILISTIC ANALYSIS.

We start by showing that a job of class i with processing time close to 2^i is likely to be completed in queue i, therefore overcoming the counterexample for the uniform distribution described in Section 3. As illustrated in Figure 1(a), we prove that the probability that target $T_{j,i}$ falls in the interval $[2^i, 2^i + 2^{i-2})$ is at most $1/j$.

This is formally proved in the following lemma:

LEMMA 6. *The expected number of unlucky jobs along the execution of the algorithm is H_n.*

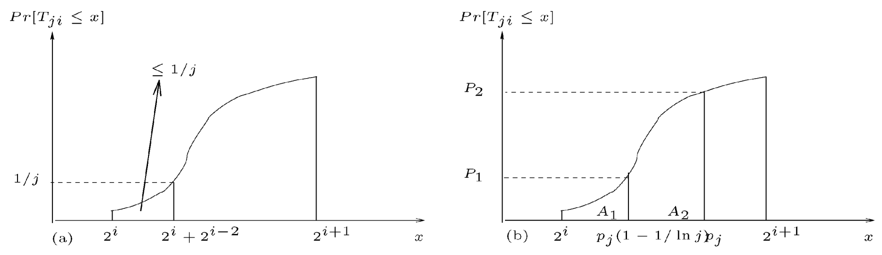

FIG. 1. Function $Pr[T_{j,i} \leq x]$ is the probability distribution of $T_{j,i}$. a) The probability that a job j of class i is unlucky is at most $1/j$. b) P_1 and $P_2 - P_1$ are the probabilities that a job is respectively big or small. They differ for at most a constant factor.

PROOF. A job j of class i with processing time $p_j \leq 2^i + 2^{i-2}$ is unlucky if it enters queue Q_{i+1}. This happens with probability at most $1/j$ as shown by:

$$\begin{aligned} \Pr\left[X_j^l = 0\right] &= \Pr[T_{j,i} < p_j] \leq \Pr\left[T_{j,i} < 2^i + 2^{i-2}\right] \\ &= \Pr\left[\beta_{j,i} > 32^{i-2}\right] = \exp(-\ln j) = \frac{1}{j}, \end{aligned}$$

since $\gamma = 4/3$. The expected number of unlucky jobs is then bounded by $\sum_{j=1}^{n} \frac{1}{j} = H_n$. $\square$

In the remainder of this section, we present a second fundamental property of our algorithm. As pointed out in the introduction we would like our algorithm to approximately follow the SRPT heuristic, implying that most of the jobs are *big*, that is, they enter the queue of completion with a remaining processing time that is at least a logarithmic fraction of the original processing time. We prove that, for any job j of class i and for any time t, the probability that job j is alive and big at time t is at least a constant fraction of the probability that job j is lucky and alive at the same time. This property implies that, at any time t, a constant fraction of lucky jobs that are alive at t are also big. This will be used to limit the difference between the expected number of jobs in the schedule of RMLF and in the optimal schedule.

Observe that to ensure the property stated above we need that when dividing the interval $[2^i, 2^{i+1})$ into $\log n$ subintervals of size $\frac{2^i}{\log n}$, for every l, the probability that the target falls in the first l intervals is at least a constant fraction of the probability that it falls in the first $l + 1$ intervals. It is straightforward to conclude that an exponential decreasing probability distribution is necessary to this purpose.

The proof of this property is straightforward if, at time t, a job j of class i is in a queue Q_k, $k < i$, while it needs some work if job j is in one of the queues of possible completion, namely Q_i or Q_{i+1}. Assume for instance job j is in queue Q_{i+1}, the fact that job j is big when it enters queue Q_{i+1} depends on the value of target $T_{j,i}$. In Figure 1b) it is pictorially shown that for a job with $p_j \geq 2^i + 2^{i-2}$ (a lucky job by definition), if $T_{j,i} < p_j$ (the job entered queue Q_{i+1}), then with constant probability $T_{j,i}$ is also smaller than $p_j(1 - 1/\ln j)$, therefore the job is big.

However, we restrict our attention to those executions in which job j is alive a time t. This might restrict the set of favourable executions, hence the argument above cannot be applied directly. The formal proof for our case is given in the following lemma:

LEMMA 7. *There exists a constant α such that, for every job $j \geq \alpha$ and time t it holds* $\Pr[X^b_j(t) = 1] \geq \exp(-4\gamma) \Pr[X^l_j(t) = 1]$.

PROOF. Since $X^b_j(t) = 1$ implies $X^l_j(t) = 1$, we have:

$$\begin{aligned}\Pr[X^b_j(t) = 1] &= \Pr\left[X^b_j(t) = 1 \cap X^l_j(t) = 1\right] \\ &= \Pr\left[X^b_j(t) = 1 | X^l_j(t) = 1\right] \cdot \Pr\left[X^l_j(t) = 1\right].\end{aligned}$$

The problem is to bound $\Pr[X^b_j(t) = 1 | X^l_j(t) = 1]$ for a job j of class i. We need to characterize the set of specific executions σ of the algorithm for which $X^l_{j,\sigma}(t) = 1$. Denote by $\mathcal{X}$ the generic assignment of all random variables with the exception of $T_{j,i-1}$ ($T_{j,i}$). We denote by $\Sigma^i_j(t)$ ($\Sigma^{i+1}_j(t)$) the set of all executions σ such that: (i) all random variables with the exception of $T_{j,i-1}$ ($T_{j,i}$) are assigned according to $\mathcal{X}$; (ii) at time t job j is in queue Q_i (Q_{i+1}); (iii) $X^l_{j,\sigma}(t) = 1$. Every execution in $\Sigma^i_j(t)$ ($\Sigma^{i+1}_j(t)$) is denoted in the following by $\sigma =< \mathcal{X}, T_{j,i-1} >$ ($\sigma =< \mathcal{X}, T_{j,i} >$). Hence, the Lemma is proved if $\Pr[X^b_j(t) = 1 | \sigma \in \Sigma^i_j(t)] \geq \exp(-4\gamma)$ and $\Pr[X^b_j(t) = 1 | \sigma \in \Sigma^{i+1}_j(t)] \geq \exp(-4\gamma)$, since a job of class i is always big in queues Q_{i-1} or lower. The following, technical Lemma proves that, for any execution σ with an assignment $T_{j,i-1} = T$ ($T_{j,i} = T$) such that j is lucky and alive at time t, j is also lucky and alive at t, for any other execution differing from σ only in the assignment $T_{j,i-1} = T' < T$ ($T_{j,i} = T' < T$). We will use this argument to complete the proof.

LEMMA 8. *Consider a job j of class i and a specific execution $\sigma =< \mathcal{X}, T_{j,i-1} = T >\in \Sigma^i_j(t)$ ($\sigma =< \mathcal{X}, T_{j,i} = T >\in \Sigma^{i+1}_j(t)$). Then, for every $T' \in [2^{i-1}, T)$ ($T' \in [2^i, T)$) it holds $\sigma' =< \mathcal{X}, T_{j,i-1} = T' >\in \Sigma^i_j(t)$ ($\sigma' =< \mathcal{X}, T_{j,i} = T' >\in \Sigma^{i+1}_j(t)$).*

PROOF. We prove the claim for a job j of class i and $\sigma \in \Sigma^i_j(t)$. The proof for $\sigma \in \Sigma^{i+1}_j(t)$ proceeds exactly the same, with straightforward changes, by replacing i with $i + 1$ and is therefore omitted. Intuitively, we prove that if j is alive, lucky and in queue Q_i at time t in accordance with execution σ, the same happens with execution σ', when the target $T_{j,i-1}$ is reduced from T to $T' < T$. Let t' denote the time at which j is processed for T' units in queue Q_{i-1} according to execution σ'. The schedules for σ and σ' are identical up to time t', hence we shall always assume $t > t'$.

We first prove (Fact a) that, for any $\hat{t} \in (t', t]$ such that j is alive at $\hat{t}$ in execution σ', we have $y_{h,\sigma'}(\hat{t}) \leq y_{h,\sigma}(\hat{t})$ for every job $h \neq j$. Observe that at least one such time instant $\hat{t}$ when job j is alive exists since by assumption, $T' < T$ and $\sigma =< \mathcal{X}, T_{j,i-1} = T >\in \Sigma^i_j(t)$. The claim above intuitively means that, by reducing the target of job j in queue Q_{i-1} to T', during interval $(t', t]$, every job other than j receives at least the same processing time in the new execution.

We consider jobs in Q_l, $l < i - 1$, and in queues Q_{i-1}, Q_i and Q_{i+1} separately. The processing of jobs in queues lower than Q_{i-1} is by no way affected by the processing in queues Q_{i-1} or higher, hence the claim is true for any $\hat{t} \in (t', t]$ and for any job $h \neq j$ in a queue lower than Q_{i-1} at time $\hat{t}$.

As to jobs in queues Q_{i-1}, Q_i and Q_{i+1}, the proof is by contradiction. Assume there exists at least one job $h \neq j$ in Q_{i-1}, Q_i or Q_{i+1}, such that $y_{h,\sigma'}(\hat{t}) > y_{h,\sigma}(\hat{t})$, for some $\hat{t} \in (t', t]$. Let $\bar{t} \in [t', \hat{t})$ be the last time when $y_{h,\sigma'}(\bar{t}) \leq y_{h,\sigma}(\bar{t})$ for all $h \neq j$. There has to be a job $h \neq j$ such that $y_{h,\sigma'}(\bar{t}) = y_{h,\sigma}(\bar{t})$ while $y_{h,\sigma'}(\bar{t}+\epsilon) > y_{h,\sigma}(\bar{t}+\epsilon)$ for every ϵ sufficiently small. The contradiction follows by proving that the set of jobs that have priority over h in σ at time $\bar{t}$ contains the set of jobs that have priority over h in σ' at time $\bar{t}$. Recall that a job j' has priority over a job j'' if j' is in a lower queue than j'' or if they are in the same queue but j' has been released earlier.

At this point of the proof, we use the Earliest Release Time First priority rule in order to have a fixed priority between jobs independently of the time they are promoted into a queue. In fact, if jobs' priorities were determined by their arrival times in the queue, the relative priority between two jobs promoted to a queue Q_i could change if the target of job j in queue Q_{i-1} were reduced. Assume job j' promoted to queue i after job j'' in execution σ. By stopping the execution of job j at time t' in execution σ' we may anticipate the processing of some job j' that could therefore be promoted to queue i before job j''.

In determining the set of jobs that have priority over h, we consider j and all jobs other than j or h separately. According to execution σ', at time $\bar{t}$, job j is in queue Q_i, while at the same time it is either in queue Q_{i-1} or Q_i in execution σ. On the other side, since $y_{h,\sigma'}(\bar{t}) = y_{h,\sigma}(\bar{t})$, at $\bar{t}$ h is in the same queue in the two executions. Therefore, if j has priority over h in σ', so it does in σ. For all jobs other than h and j, we observe that in execution σ' and at time $\bar{t}$ they have a remaining processing time that is smaller or equal than in σ, by definition of $\bar{t}$. Therefore, in execution σ and at time $\bar{t}$, they are in the same queue or in a lower queue than in execution σ', for which, if at $\bar{t}$ and in execution σ' they have priority over h, the same happens with execution σ. This proves that the set of jobs that have priority over h in σ at time $\bar{t}$ contains the set of jobs that have priority over h in σ' at time $\bar{t}$.

But this contradicts the assumption above and hence (Fact a) $y_{h,\sigma'}(\hat{t}) \leq y_{h,\sigma}(\hat{t})$ for every job $h \neq j$ and for any $\hat{t} \in (t', t]$ such that j is alive at $\hat{t}$. As a corollary of (Fact a), we have (Fact b) for any $\hat{t} \in (t', t]$ such that j is alive at $\hat{t}$, we have $\delta^A_{\sigma'}(\hat{t}) \leq \delta^A_{\sigma}(\hat{t})$.

We can now prove that $X^l_{j,\sigma}(t) = 1$ implies $X^l_{j,\sigma'}(t) = 1$. Assume this is not the case and consider the last $\hat{t} \in (t', t]$ such that j is alive at time $\hat{t}$. (Fact b) implies that the overall work done by the machines during interval $(t', \hat{t}]$ in execution σ' is not more than in σ. This consideration and (Fact a) together imply $y_{j,\sigma'}(\hat{t}) \geq y_{j,\sigma}(\hat{t})$, which contradicts the assumption. As a consequence, j is alive at time t in execution σ' and it is obviously lucky, since $T_{j,i-1}$ has been reduced, while $T_{j,i}$ was left unchanged, that is $X^l_{j,\sigma'}(t) = 1$ (for the case $\sigma \in \Sigma^{i+1}_j(t)$, observe that $X^l_{j,\sigma}(t) = 1$ implies $p_j > 2^i + 2^{i-2}$). Finally, since $y_{j,\sigma'}(\hat{t}) \geq y_{j,\sigma}(\hat{t})$, j is in queue Q_i at time t, according to execution σ'. This, and the fact that $X^l_{j,\sigma'}(t) = 1$, together imply $\sigma' \in \Sigma^i_j(t)$. □

COROLLARY 9. *Consider a job j of class i and a specific assignment $\mathcal{X}$ of all random variables with the exception of $T_{j,i-1}$ ($T_{j,i}$). Consider the set $\Sigma^i_j(t)$ ($\Sigma^{i+1}_j(t)$) of all sequences with assignment $\mathcal{X}$ where job j is in queue Q_i (Q_{i+1}) at time t, such that $X^l_{j,\sigma}(t) = 1$. Then, there exists a value $x \in [2^{i-1}, 2^i)$ ($x \in [2^i, 2^{i+1})$) such that for every $T_{j,i-1} \in [2^{i-1}, x)$ ($T_{j,i} \in [2^i, x)$) the corresponding sequence* *is in* $\Sigma_j^i(t)$ ($\sigma \in \Sigma_j^{i+1}(t)$)*, while it is not in* $\Sigma_j^i(t)$ ($\sigma \in \Sigma_j^{i+1}(t)$) *for every* $T_{j,i-1} \in [x, 2^i)$ ($T_{j,i} \in [x, 2^{i+1})$)*.*

Denoted by x the smallest value for variable $T_{j,i-1}$ ($T_{j,i}$) such that for the corresponding execution σ we have $X^l_{j,\sigma}(t) = 0$, the proof of Corollary 9 follows by a simple contradiction argument and is therefore omitted.

We can now prove Lemma 7. Consider a job $j \geq e^5$ of class i and consider all sequences σ for which $X^l_{j,\sigma}(t) = 1$. For all sequences in which j is in queue Q_k, $k \leq i-1$, j is big, since $T_{j,i-2} < 2^{i-1}$ and then, if $\bar{t}$ denotes the time j was promoted to queue Q_k, $y_j(\bar{t}) > p_j - 2^{i-1} \geq p_j/2 \geq p_j/\ln j$, for $j \geq 8$. As a consequence, the only possibility for a lucky job of class i to be small is to enter queue Q_i or queue Q_{i+1} at time $\bar{t} \leq t$, with $y_j(\bar{t}) < \frac{p_j}{\ln j}$.

First, consider any subset $\Sigma_j^i(t)$ ($\Sigma_j^{i+1}(t)$) of the set of sequences in which job j is alive, lucky and in queue Q_i (Q_{i+1}) at time t, such that all sequences in $\Sigma_j^i(t)$ ($\Sigma_j^{i+1}(t)$) only differ for the value of $T_{j,i-1}$ ($T_{j,i}$).

If we prove a bound on the probability that job j is big when conditioned to sequences $\sigma \in \Sigma_j^i(t)$ ($\sigma \in \Sigma_j^{i+1}(t)$), then the claim is proved for j in Q_i (j in Q_{i+1}) at time t. Let x denote the value derived from Corollary 9. We proceed as follows:

$$\begin{aligned}
\Pr\left[T_{j,i-1} \leq p_j - \frac{p_j}{\ln j} \mid \sigma \in \Sigma_j^i(t)\right] &= \Pr\left[T_{j,i-1} \leq p_j\left(1 - \frac{1}{\ln j}\right) \middle| \, T_{j,i-1} < x\right] \\
&\geq \Pr\left[T_{j,i-1} \leq p_j\left(1 - \frac{1}{\ln j}\right) \middle| \, T_{j,i-1} < 2^i\right] \\
&\geq \frac{\Pr\left[T_{j,i-1} \leq p_j\left(1 - \frac{1}{\ln j}\right)\right]}{\Pr\left[T_{j,i-1} < 2^i\right]} \\
&\geq \Pr\left[\beta_{j,i-1} \geq 2^i - p_j\left(1 - \frac{1}{\ln j}\right)\right] \\
&\geq \exp\left(-\gamma \frac{2^i - p_j\left(1 - \frac{1}{\ln j}\right)}{2^{i-1}} \ln j\right) \\
&= \exp\left(-\gamma \frac{2^i - p_j}{2^{i-1}} \ln\ j - \gamma \frac{p_j}{2^{i-1}}\right) \\
&\geq \exp\left(-\gamma \frac{p_j}{2^{i-1}}\right) \geq \exp(-4\gamma),
\end{aligned}$$

where the second inequality follows since $x < 2^i$, the seventh inequality stems from $p_j \geq 2^i$, while the last inequality from $p_j < 2^{i+1}$.

We then prove the claim for sequences belonging to $\Sigma_j^{i+1}(t)$. Since job j is lucky, $p_j > 2^i + 2^{i-2}$. We show that job j is big when it enters Q_{i+1} with (conditioned) probability at least $e^{-2\gamma}$.

Corollary 9 ensures that there exists a value x such that $\sigma =< \mathcal{X}, T_{j,i} > \in \Sigma_j^{i+1}(t)$ for every $T_{j,i} \in [2^i, x)$. If we prove a bound on the probability that job j is big when conditioned to sequences $\sigma \in \Sigma_j^{i+1}(t)$, then the claim is proved. We proceed as follows:

$$\begin{aligned}
\Pr\left[T_{j,i} \leq p_j - \frac{p_j}{\ln j} \,\middle|\, \sigma \in \Sigma_j^{i+1}(t)\right] &= \Pr\left[T_{j,i} \leq p_j \left(1 - \frac{1}{\ln j}\right) \middle| \, T_{j,i} < x\right] \\
&\geq \Pr\left[T_{j,i} \leq p_j \left(1 - \frac{1}{\ln j}\right) \middle| \, T_{j,i} < p_j\right] \\
&= \frac{\Pr\left[T_{j,i} \leq p_j\left(1 - \frac{1}{\ln j}\right)\right]}{\Pr[T_{j,i} < p_j]} \\
&= \frac{\Pr\left[\beta_{j,i} \geq 2^{i+1} - p_j\left(1 - \frac{1}{\ln j}\right)\right]}{\Pr[\beta_{j,i} > 2^{i+1} - p_j]} \\
&\geq \frac{e^{-\gamma \frac{2^{i+1} - p_j\left(1 - \frac{1}{\ln j}\right)}{2^i} \ln j}}{e^{-\gamma \frac{2^{i+1} - p_j}{2^i} \ln j}} \\
&= \frac{e^{-\gamma \frac{2^{i+1} - p_j}{2^i} \ln j - \gamma \frac{p_j}{2^i}}}{e^{-\gamma \frac{2^{i+1} - p_j}{2^i} \ln j}} \\
&\geq \exp\left(-\gamma \frac{p_j}{2^i}\right) \geq \exp(-2\gamma).
\end{aligned}$$

Here, the second inequality follows since $x < p_j$ (because $\sigma \in \Sigma_j^{i+1}(t)$), while the fourth inequality holds since $T_{j,i,} = \max\{2^i, 2^{i+1} - \beta_{j,i}\}$, since $p_j \geq 2^i + 2^{i-2}$ and $p_j(1 - 1/\ln j) \geq 2^i$ when $j \geq e^5$. The last inequality follows from $p_j \leq 2^{i+1}$. Taking $\alpha = \lceil e^5 \rceil$ the claim of Lemma 7 is proved.

4.3. The $O(\log n \log P)$-Competitive Ratio.

In this section, we prove that RMLF is $O(\log n \log P)$ competitive.

Intuitively, the $\log n$ factor is the price of nonclairvoyance and it is a direct consequence of Definition 2 (see Eq. (3) in the proof of Lemma 13). The $\log P$ factor, instead, comes from the fact that during the execution, at most $\log P$ queues are created and each might have a number of jobs that have been initiated and subsequently preempted by the algorithm. As pointed out further in Section 4.4, this is a consequence of the fact that the on line algorithm in general does not optimally balance the load among the available machines.

Recall that, for any possible execution σ, $\delta_\sigma^A(t) = \delta_\sigma^u(t) + \delta_\sigma^l(t)$, for which the total flow time of RMLF can be expressed as:

$$\begin{aligned}
E_\sigma[F_\sigma^A] &= E_\sigma\left[\int_{t\geq 0} \delta_\sigma^A(t)dt\right] = E_\sigma\left[\int_{t\geq 0} (\delta_\sigma^u(t) + \delta_\sigma^l(t))dt\right] \\
&= E_\sigma\left[\int_{t\geq 0} \delta_\sigma^u(t)dt\right] + E_\sigma\left[\int_{t\geq 0} \delta_\sigma^l(t)dt\right] \\
&= \int_{t\geq 0} E_\sigma[\delta_\sigma^u(t)]dt + \int_{t\geq 0} E_\sigma[\delta_\sigma^l(t)]dt. \qquad (1)
\end{aligned}$$

In the remainder of this section, we need to bound the two contributions to Eq. (1). The following lemma, based on the claim of Lemma 6, bounds the first contribution to the total flow time.

LEMMA 10. $\int_{t\geq 0} E_\sigma\left[\delta^u_\sigma(t)\right]dt = O(\log n)\sum_j p_j$.

PROOF. First, observe that $E_\sigma[\delta^u_\sigma(t)] = O(\log n)$. This follows from:

$$\begin{aligned} E_\sigma\left[\delta^u_\sigma(t)\right] &= \sum_{j=1}^{n} \Pr\left[X^l_j = 0 \cap X_j(t) = 1\right] \\ &\leq \sum_{j=1}^{n} \Pr\left[X^l_j = 0\right] = O(\log n). \end{aligned}$$

Since the algorithm never keeps machines idle, no job is in the system after time $\sum_j p_j$. This, together with the inequality above implies:

$$\begin{aligned} \int_{t\geq 0} E_\sigma\left[\delta^u_\sigma(t)\right]dt &\leq \int_{0\leq t\leq \sum_j p_j} E_\sigma\left[\delta^u_\sigma(t)\right]dt = \int_{0\leq t\leq \sum_j p_j} O(\log n)dt \\ &= O(\log n)\sum_j p_j. \end{aligned}$$ □

The next lemma uses the claim of Lemma 7 to relate the expected number of lucky jobs alive at time t to the expected number of big jobs alive at time t.

LEMMA 11. *There exists a constant α such that, at any time t, the number of lucky jobs in the system satisfies the following relation:*

$$E_\sigma\left[\delta^l_\sigma(t)\right] \leq \alpha + \exp(4\gamma)E_\sigma\left[\delta^b_\sigma(t)\right].$$

PROOF. Set $\alpha = \lceil e^5 \rceil$. We have the following inequalities:

$$\begin{aligned} E_\sigma\left[\delta^l_\sigma(t)\right] &= \sum_{j=1}^{n} \Pr\left[X^l_j(t) = 1\right] \\ &\leq \alpha + \sum_{j=\alpha+1}^{n} \Pr\left[X^l_j(t) = 1\right] \\ &\leq \alpha + \exp(4\gamma) \sum_{j=\alpha+1}^{n} \Pr\left[X^l_b(t) = 1\right] \\ &\leq \alpha + \exp(4\gamma)E_\sigma\left[\delta^b_\sigma(t)\right], \end{aligned}$$

where the third inequality follows from the claim of Lemma 7. □

The next lemma bounds the expected time spent by the algorithm while all machines are busy.

LEMMA 12. $\int_{t\geq 0} \Pr[\delta^A(t) \geq m]\, dt \leq \frac{1}{m}\sum_{j\in J} p_j$.

PROOF. The claim is derived from the following relations:

$$\begin{aligned}\int_{t\geq 0} \Pr[\delta^A(t) \geq m]\, dt &= \int_{t\geq 0} \sum_{\sigma:\delta^A_\sigma(t)\geq m} \Pr[\sigma]\, dt \\ &= \sum_\sigma \Pr[\sigma] \int_{t:\delta^A_\sigma(t)\geq m} dt \\ &\leq \frac{1}{m}\sum_{j\in J} p_j \sum_\sigma \Pr[\sigma] \\ &= \frac{1}{m}\sum_{j\in J} p_j,\end{aligned}$$

where the third inequality follows since for every σ $\int_{t:\delta^A_\sigma(t)\geq m} dt$ is bounded by $\frac{1}{m}\sum_{j\in J} p_j$. □

The expected cost of RMLF of expression (1) is then rewritten as follows:

$$\begin{aligned}&E_\sigma\left[F^A_\sigma\right] = E_\sigma\left[\int_{t\geq 0} \delta^A_\sigma(t)dt\right] \\ &= \int_{t\geq 0} E_\sigma\left[\delta^u_\sigma(t)\right]dt + \int_{t\geq 0} E_\sigma\left[\delta^l_\sigma(t)\right]dt \\ &\leq O(\log n)\sum_j p_j + \int_{0\leq t\leq \sum_j p_j} (\alpha + \exp(4\gamma)E_\sigma\left[\delta^b_\sigma(t)\right])dt \\ &= O(\log n)\sum_j p_j + \alpha\sum_j p_j + \exp(4\gamma)\int_{t\geq 0} E_\sigma\left[\delta^b_\sigma(t)\right]dt \\ &= O(\log n)\sum_j p_j + \exp(4\gamma)E_\sigma\left[\int_{t:\delta^A_\sigma(t)<m} \delta^b_\sigma(t)dt\right] + \exp(4\gamma)E_\sigma\left[\int_{t:\delta^A_\sigma(t)\geq m} \delta^b_\sigma(t)dt\right] \\ &= O(\log n)\sum_j p_j + \exp(4\gamma)\int_{t\geq 0} E_\sigma\left[\delta^b_\sigma(t)|\delta^A(t)\geq m\right]\Pr[\delta^A(t)| \geq m]dt. \qquad (2)\end{aligned}$$

The third inequality follows by Lemma 10, by Lemma 11, and by recalling that no job is in the system after time $\sum_j p_j$, hence $\delta^l(t) = 0$, for any $t > \sum_j p_j$. The fifth inequality follows by the linearity of expectation and by partitioning the time axis, for each possible execution σ, into the instants for which $\delta^A_\sigma(t) < m$ and into those for which $\delta^A_\sigma(t) \geq m$. Finally, the last inequality follows by observing that $\int_{t:\delta^A_\sigma(t)<m} \delta^b_\sigma(t)dt \leq \int_{t:\delta^A_\sigma(t)<m} \delta^A_\sigma(t)dt \leq \sum_{j\in J} p_j$, since the overall work done by the m machines in any possible execution of the algorithm is bounded by the sum of the processing times of the n jobs, and by expanding $E_\sigma[\int_{t:\delta^A_\sigma(t)\geq m} \delta^b_\sigma(t)dt]$.

We are left to bound the last term of Eq. (2) and to this purpose we present the critical part of the proof, in which we bound the number of big jobs at time t in the generic execution of RMLF determined by a set of random choices of the algorithm.

LEMMA 13. *For any outcome σ of the random choices of the algorithm, for any time $t : \delta^A_\sigma(t) \geq m$, $\delta^b_{\sigma,\geq k_1,\leq k_2}(t) \leq 2\ln n(2(m-1)(k_2-k_1+1) + 3\delta^{OPT}(t)) + m(k_2-k_1+2)$.*

PROOF. In the following, we omit σ when clear from the context. For $\delta^b_{\geq k_1, \leq k_2}(t)$ we write the following relations:

$$\delta^b_{\geq k_1, \leq k_2}(t) \;\leq\; m(k_2 - k_1 + 2) + \ln n \sum_{i=k_1}^{k_2} \frac{V^A_{=i}(t)}{2^i}. \tag{3}$$

The bound follows since every big job of class i has remaining processing time at least $2^i / \ln n$. We should additionally consider those big jobs of class i, $i = k_1, \ldots, k_2$, with a processing time smaller than $2^i / \ln n$ since they have been processed after entering the queue they are in. For a job of class i this may happen in queues $i-1$, i or $i+1$. However, for every queue, there are at most m jobs that have been processed after entering the queue (Fact 5), for which the total number of such jobs is bounded by $m(k_2 - k_1 + 2)$. We continue with:

$$\begin{aligned}
\sum_{i=k_1}^{k_2} \frac{V^A_{=i}(t)}{2^i} &= \sum_{i=k_1}^{k_2} \frac{V^{OPT}_{=i}(t) + \Delta V_{=i}(t)}{2^i} \\
&\leq 2\delta^{OPT}_{\geq k_1, \leq k_2}(t) + \sum_{i=k_1}^{k_2} \frac{\Delta V_{=i}(t)}{2^i} \\
&= 2\delta^{OPT}_{\geq k_1, \leq k_2}(t) + \sum_{i=k_1}^{k_2} \frac{\Delta V_{\leq i}(t) - \Delta V_{\leq i-1}(t)}{2^i} \\
&= 2\delta^{OPT}_{\geq k_1, \leq k_2}(t) + \frac{\Delta V_{\leq k_2}(t)}{2^{k_2}} + \sum_{i=k_1}^{k_2-1} \frac{\Delta V_{\leq i}(t)}{2^{i+1}} - \frac{\Delta V_{\leq k_1-1}(t)}{2^{k_1}} \\
&\leq 2\delta^{OPT}_{\geq k_1, \leq k_2}(t) + \delta^{OPT}_{\leq k_1-1}(t) + 2\sum_{i=k_1}^{k_2} \frac{\Delta V_{\leq i}(t)}{2^{i+1}} \\
&\leq 2\delta^{OPT}_{\leq k_2}(t) + 2\sum_{i=k_1}^{k_2} \frac{\Delta V_{\leq i}(t)}{2^{i+1}},
\end{aligned} \tag{4}$$

where the second inequality follows since a job of class i has size less than 2^{i+1}, while the fifth inequality follows since $-\frac{\Delta V_{\leq k_1-1}(t)}{2^{k_1}} \leq \frac{V^{OPT}_{\leq k_1-1}}{2^{k_1}} \leq \delta^{OPT}_{\leq k_1-1}$.

We are left to study the term $\sum_{i=k_1}^{k_2} \frac{\Delta V_{\leq i}(t)}{2^{i+1}}$. For any $t_1 \leq t_2 \leq t$, for a generic function f, denote by $f^{[t_1,t_2]}(t)$ the value of function f at time t when restricted to jobs released between t_1 and t_2, for example, $L^{[t_1,t_2]}_{\leq i}(t)$ is the work done by time t on jobs of class at most i released between time t_1 and t_2. Let $t_0(t)$ be the last time prior to time t in the execution determined by the set of random choices σ, such that $\delta^A_\sigma(t_0) < m$. Denote by $t_i < t$ the maximum between $t_0(t)$ and the last time prior to time t in which a job was processed in queue Q_{i+1} or higher in this specific execution of RMLF. Observe that, for $i = k_1, \ldots, k_2$, $[t_{i+1}, t) \supseteq [t_i, t)$.

At time t_{i+1} either the algorithm was processing a job in queue Q_{i+2} or higher, or $t_{i+1} = t_0(t)$, for which at time t_{i+1} at most $m-1$ jobs were in queues $Q_0, \ldots, Q_{i+1}$.

At time t we have:

$$\begin{aligned}
&\Delta V_{\leq i}(t) \leq V_{\leq i}^{A[0,t_{i+1}]}(t) + \Delta V_{\leq i}^{(t_{i+1},t]}(t)\\
&\leq V_{\leq i}^{A[0,t_{i+1}]}(t) + V_{>i}^{A[0,t_{i+1}]}(t) - V_{>i}^{A[0,t_{i+1}]}(t_{i+1}) + L_{>i}^{A(t_{i+1},t]}(t) - L_{>i}^{OPT(t_{i+1},t]}(t)\\
&\leq (m-1)2^{i+2} + L_{>i}^{A(t_{i+1},t]}(t) - L_{>i}^{OPT(t_{i+1},t]}(t).
\end{aligned}$$

The first inequality follows since $\Delta V_{\leq i}(t)$ is bounded by the sum of a first contribution equal to the volume of the jobs released until time t_{i+1} in RMLF's schedule and a contribution equal to the volume difference of jobs released after time t_{i+1}. The second inequality states that the volume difference is bounded by the time spent in the interval $(t_{i+1}, t]$ by RMLF processing jobs of class bigger than i minus the time spent by the optimal solution processing jobs of class bigger than i released in the interval $(t_{i+1}, t]$. The third inequality follows from $V_{\leq i}^{A[0,t_{i+1}]}(t) + V_{>i}^{A[0,t_{i+1}]}(t) - V_{>i}^{A[0,t_{i+1}]}(t_{i+1}) \leq (m-1)2^{i+2}$. This relation holds since at most $m-1$ jobs are in queues $Q_0, .., Q_{i+1}$ at time t_{i+1}. Assume x_1 such jobs are of class $\leq i$ and x_2 such jobs are of class $> i$. The remaining volume of the x_1 jobs of class $\leq i$ at time t_{i+1} is at most $x_1 2^{i+1}$, while the x_2 jobs of class $> i$ are processed in $[t_{i+1}, t)$ for at most $x_2 2^{i+2}$ time units. From $x_1 + x_2 \leq m-1$, the inequality follows.

The fact that at most $m-1$ jobs are in the system in queues $Q_0, \ldots, Q_{i+1}$ at time t_{i+1} is crucial in proving the tightness result for $m = 1$. In fact for $m = 1$, this contribution to the flow time will disappear.

In the following, we adopt the convention $t_{k_1-1} = t$. From the inequality above we write:

$$\begin{aligned}
\sum_{i=k_1}^{k_2} \frac{\Delta V_{\leq i}(t)}{2^{i+1}} &\leq \sum_{i=k_1}^{k_2} \frac{(m-1)2^{i+2} + \Delta L_{>i}^{(t_{i+1},t]}(t)}{2^{i+1}} \leq 2(m-1)(k_2 - k_1 + 1)\\
&\quad + \sum_{i=k_1}^{k_2} \frac{\Delta L_{>i}^{(t_{i+1},t]}(t)}{2^{i+1}} \qquad (5)
\end{aligned}$$

We then concentrate on the term $\sum_{i=k_1}^{k_2} \frac{\Delta L_{>i}^{(t_{i+1},t]}(t)}{2^{i+1}}$ for which we have:

$$\begin{aligned}
&\sum_{i=k_1}^{k_2} \frac{L_{>i}^{A(t_{i+1},t]}(t) - L_{>i}^{OPT(t_{i+1},t]}(t)}{2^{i+1}} = \sum_{i=k_1}^{k_2} \sum_{j=k_1-1}^{i} \frac{L_{>i}^{A(t_{j+1},t_j]}(t) - L_{>i}^{OPT(t_{j+1},t_j]}(t)}{2^{i+1}}\\
&\leq \sum_{j=k_1-1}^{k_2} \sum_{i=j}^{k_2} \frac{L_{>i}^{A(t_{j+1},t_j]}(t) - L_{>i}^{OPT(t_{j+1},t_j]}(t)}{2^{i+1}},
\end{aligned}$$

where the second equality follows by partitioning the work done on the jobs released in the interval $(t_{i+1}, t]$ into the work done on the jobs released in the intervals $(t_{j+1}, t_j]$, $j = k_1 - 1, .., i$.

Let $\bar{i}(j) \in [j, .., k_2]$ be the index that maximizes $L_{>i}^{A(t_{j+1},t_j]} - L_{>i}^{OPT(t_{j+1},t_j]}$. We then have:

$$\begin{aligned}
\sum_{i=k_1}^{k_2} \frac{\Delta L_{>i}^{(t_{i+1},t]}(t)}{2^{i+1}} &\le \sum_{j=k_1-1}^{k_2} \sum_{i=j}^{k_2} \frac{L_{>\bar{i}(j)}^{A(t_{j+1},t_j]}(t) - L_{>\bar{i}(j)}^{OPT(t_{j+1},t_j]}(t)}{2^{i+1}} \\
&\le \sum_{j=k_1-1}^{k_2} \frac{L_{>\bar{i}(j)}^{A(t_{j+1},t_j]}(t) - L_{>\bar{i}(j)}^{OPT(t_{j+1},t_j]}(t)}{2^{j}} \\
&\le 2 \sum_{j=k_1-1}^{k_2} \delta_{>\bar{i}(j)}^{OPT(t_{j+1},t_j]}(t) \le 2\delta_{\ge k_1}^{OPT(t_{j+1},t]}(t) \le 2\delta_{\ge k_1}^{OPT}(t). \qquad (6)
\end{aligned}$$

To prove the third inequality, observe that every job of class bigger than $\bar{i}(j) \ge j$ released in the time interval $(t_{j+1}, t_j]$ is processed by RMLF in the interval $(t_{j+1}, t]$ for at most 2^{j+2} time units. Order the jobs of this specific set by increasing $x_j(t)$. Now, observe that each of these jobs has remaining processing time at least $2^{\bar{i}(j)+1} \ge 2^{j+1}$ at the beginning of interval $(t_{j+1}, t]$ and we give to the optimum the further advantage that it finishes every such job when processed for an amount $x_j(t) \le 2^{j+2}$. To maximize the number of finished jobs, the optimum places the work $L_{>\bar{i}(j)}^{OPT(t_{j+1},t_j]}$ on the jobs with smaller $x_j(t)$. The optimum is then left at time t with a number of jobs

$$\left(\delta_{>\bar{i}(j)}^{OPT(t_{j+1},t_j]} \ge \frac{L_{>\bar{i}(j)}^{A(t_{j+1},t_j]} - L_{>\bar{i}(j)}^{OPT(t_{j+1},t_j]}}{2^{j+1}} \right),$$

for which the third inequality holds.

Altogether, from (3), (4), (5) and (6) we obtain:

$$\begin{aligned}
&\delta_{\ge k_1, \le k_2}^{b}(t) \\
&\le m(k_2 - k_1 + 2) + 2\ln n(\delta_{\le k_2}^{OPT}(t) + 2(m-1)(k_2 - k_1 + 1) + 2\delta_{\ge k_1}^{OPT}(t)) \\
&\le m(k_2 - k_1 + 2) + 2\ln n\left(2(m-1)(k_2 - k_1 + 1) + 3\delta^{OPT}(t)\right),
\end{aligned}$$

for which the claim of the Lemma is proved. □

We then plug the claim of Lemma 13 into the expression (2) of the total flow time of RMLF to achieve our first result:

$$\begin{aligned}
E_\sigma[F_\sigma^A] &= O(\log n)\sum_j p_j + \exp(4\gamma)\int_{t\ge 0} E_\sigma\left[\delta_\sigma^b(t) | \delta^A(t) \ge m\right] \Pr[\delta^A(t)| \ge m]dt \\
&\le O(\log n)\sum_j p_j \\
&\quad + \exp(4\gamma)\int_{t\ge 0} (2\ln n(2(m-1)(\log P + 2) + 3\delta^{OPT}(t))) \\
&\qquad \Pr[\delta^A(t) \ge m]dt
\end{aligned}$$

$$
\begin{aligned}
&= O(\log n) \sum_j p_j \\
&\quad + O(\log n \log P) \int_{t \geq 0} m \Pr[\delta^A(t) \geq m] dt + O(\log n) \int_{t \geq 0} \delta^{OPT}(t) dt \\
&= O(\log n) \sum_{j \in J} p_j + O(\log n \log P) \sum_j p_j + O(\log n) F^{OPT} \\
&= O(\log n \log P) F^{OPT},
\end{aligned}
$$

where in the second inequality we apply Lemma 13 and Fact 1, while in the fourth we apply the claim of Lemma 12. We therefore state our first result:

THEOREM 14. *RMLF is an $O(\log n \log P)$ competitive nonclairvoyant randomized algorithm for minimizing the total flow time on parallel machines.*

4.4. THE $O(\text{LOG } n)$ COMPETITIVE RATIO FOR $m = 1$, $O(\text{ LOG } n \text{ LOG } \frac{n}{m})$ COMPETITIVE RATIO FOR $m \geq 2$. In this section, we strengthen the analysis to obtain an $O(\log n)$ tight bound for a single machine and an $O(\log n \log \frac{n}{m})$ bound for $m \geq 2$ parallel machines.

Again, the $\log n$ factor is the price of nonclairvoyance, as already commented at the beginning of Section 4.3. The $\log \frac{n}{m}$ term, instead, comes from the fact that the optimum might work more than the algorithm, due to a better distribution of the overall load among the machines. In the worst case, the unbalance of MLF with respect to the optimum brings to a multiplicative factor $O(\log n \log \frac{n}{m})$ in the competitive ratio. As expected, this term vanishes in the single machine case, where instead of a competitive ratio $O(\log^2 n)$, we have a tighter $O(\log n)$. In this respect, this contribution is similar to the one emerging in the analysis of SRPT [Leonardi and Raz 1997].

Let $\bar{k}$ be the lowest class such that less than m jobs of class $\geq \bar{k}$ are released. Let $T(\sigma) = \{t : \delta_\sigma^A(t) \geq m\}$ be the set of time instants where all machines are busy in the execution of RMLF defined by σ. Let $T_j(\sigma) \subseteq T(\sigma)$, $j = 0, .., \bar{k}$, be the set of time instants where at least one machine is busy with jobs in queue Q_j and no machine is busy with jobs in queues higher than Q_j in the execution of RMLF defined by σ. Let $T_{\bar{k}+1}(\sigma) \subseteq T(\sigma)$ be the set of time instants when at least one machine is processing a job in queue $Q_{\bar{k}+1}$ or higher and all machines are busy. Observe that for each σ, $\{T_0(\sigma), T_1(\sigma), \ldots, T_{\bar{k}+1}(\sigma)\}$ defines a partition of $T(\sigma)$. For the sake of simplicity, in the sequel, with a slight abuse of notation, we use $T_j(\sigma)$ to denote both the above defined set and its size. For the total flow time of RMLF, the last term of Eq. (2) can be rewritten using the claim of Lemma 13 as follows:

$$
\begin{aligned}
&\int_{t \geq 0} \mathrm{E}_\sigma\left[\delta_\sigma^{\mathrm{b}}(\mathrm{t}) | \delta^{\mathrm{A}}(\mathrm{t}) \geq \mathrm{m}\right] \Pr[\delta^{\mathrm{A}}(\mathrm{t}) \geq \mathrm{m}] dt \\
&\quad = \sum_\sigma \Pr(\sigma) \int_{t \in T(\sigma)} \delta_\sigma^b(t) dt \\
&\quad \leq \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} \int_{t \in T_j(\sigma)} (2m + \delta^b_{\sigma, \geq j, \leq \bar{k}-1}(t)) dt + \sum_\sigma \Pr(\sigma) \int_{t \in T_{\bar{k}+1}(\sigma)} \delta_\sigma^b(t) dt
\end{aligned}
$$

$$
\begin{aligned}
&\le \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} \int_{t \in T_j(\sigma)} (2 \ln n (2(m-1)(\bar{k}-j) + 3\delta^{OPT}(t)) + m(\bar{k}-j+3)) dt \\
&\quad + \sum_\sigma \Pr(\sigma) 2m \int_{t \in T_{\bar{k}+1}(\sigma)} dt \\
&= \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} \int_{t \in T_j(\sigma)} 4 \ln n (m-1)(\bar{k}-j) dt \\
&\quad + \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} \int_{t \in T_j(\sigma)} 6 \ln n \delta^{OPT}(t) dt \\
&\quad + \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} \int_{t \in T_j(\sigma)} m(\bar{k}-j+3) dt + 2 \sum_j p_j \\
&= O(\log n) \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} (m-1)(\bar{k}-j) T_j(\sigma) + \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} m(\bar{k}-j) T_j(\sigma) \\
&\quad + 3m \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} T_j(\sigma) + O(\log n) \sum_\sigma \Pr(\sigma) \int_{t \in T_j(\sigma)} \delta^{OPT}(t) dt + 2 \sum_j p_j \\
&= O(\log n) \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} (m-1)(\bar{k}-j) T_j(\sigma) + \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} m(\bar{k}-j) T_j(\sigma) \\
&\quad + O(\log n) F^{OPT}.
\end{aligned}
$$

The second inequality follows by partitioning for every σ, the set $\{t \ge 0 : \delta_\sigma^A(t) \ge m\}$ into the subsets $\{T_0(\sigma), T_1(\sigma), \ldots, T_{\bar{k}+1}(\sigma)\}$, and by observing that at any time $t \in T_j(\sigma)$ there are at most m jobs of class less than j and $m-1$ jobs of class bigger or equal than $\bar{k}$. The third inequality follows by the claim of Lemma 13 and by observing that for any time $t \in T_{\bar{k}+1}(\sigma)$ at most $2m$ jobs are in the system since (i) a machine is processing a job in a queue $\bar{k}+1$ or higher and hence at most $m-1$ jobs of class less than $\bar{k}$ are in the system; (ii) at most $m-1$ jobs of class $\ge \bar{k}$ have been released. The other inequalities follow by rearranging the terms and observing that (i) $\sum_\sigma Pr(\sigma) \int_{t \in T_j(\sigma)} \delta^{OPT}(t) dt \le F^{OPT}$, (ii) $\sum_j p_j \le F^{OPT}$, (iii) by applying the claim of Lemma 12.

As a consequence of the inequalities above and from Eq. (2), we can write:

$$
\begin{aligned}
E_\sigma\left[F_\sigma^A\right] &= O(\log n) F^{OPT} + \exp(4\gamma)(m-1) O(\log n) \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} (\bar{k}-j) T_j(\sigma) \\
&\quad + \exp(4\gamma) \sum_\sigma \Pr(\sigma) \sum_{j=0}^{\bar{k}} m(\bar{k}-j) T_j(\sigma). \qquad (7)
\end{aligned}
$$

We are left to bound, for any σ, the term $F_\sigma(n) = \sum_{j=0}^{\bar{k}} m(\bar{k}-j) T_j(\sigma)$. We show this in the following Lemma:

LEMMA 15. *For any outcome of the random choices σ of the algorithm, it holds*

$$F_\sigma(n) = \sum_{j=1}^{\bar{k}} m(\bar{k} - j)T_j(\sigma) = O\left(\log \frac{n}{m}\right) F^{OPT}.$$

PROOF. We define $T_j^l(\sigma) \subseteq T(\sigma)/T_{\bar{k}+1}(\sigma)$ to be the set of time instants where machine l, $l = 1, \ldots, m$, is processing a job of queue j, $j = 0, \ldots, \bar{k}$, in execution σ. Observe that, for each l, $\{T_0^l(\sigma), \ldots T_{\bar{k}}^l\}$ defines a partition of $T(\sigma)/T_{\bar{k}+1}(\sigma)$. Let n_j^l be the number of jobs finished by machine l in queue j in this specific execution of RMLF. In the following we use $T_j^l(\sigma)$ to denote both the above defined set and its size and we omit σ when clear from the context. We have the following inequalities:

$$F(n) = \sum_{j=0}^{\bar{k}} m(\bar{k} - j)T_j \le \sum_{j=0}^{\bar{k}} \sum_{l=1}^{m} (\bar{k} - j)T_j^l \le \sum_{j=0}^{\bar{k}} \sum_{i=0}^{j} \sum_{l=1}^{m} n_j^l(\bar{k} - i)2^{i+1}.$$

The second inequality follows since any time $t \in T_j$ is also part of the set T_i^l, for some $i \le j$. The third inequality follows since any job that is completed in queue Q_j has been processed in any queue Q_i, $i \le j$, for at most 2^{i+1}, therefore giving a contribution of $(\bar{k} - i)2^{i+1}$.

Since $\sum_{i=0}^{j} 2^{i+1}(\bar{k} - i) \le 2^{j+2}(\bar{k} - j + 1)$, it follows

$$F(n) \le 4 \sum_{j=0}^{\bar{k}} \sum_{l=1}^{m} n_j^l(\bar{k} - j + 1)2^j$$

Let $I_j = 2^j \sum_{l=1}^{m} n_j^l$. We show that $\sum_{j=0}^{\bar{k}}(\bar{k} - j + 1)I_j = O(\log \frac{n}{m})F^{OPT}$. Let $\mathcal{I} = \max\{2^{\bar{k}}, 1/m \sum_{j=0}^{\bar{k}} I_j\}$. Since there are more than m jobs of class at least $\bar{k} - 1$ we have $m2^{\bar{k}-1} \le F^{OPT}$. Since every job ending in a queue j has size at least 2^{j-1}, we have $\sum_{j=0}^{\bar{k}} I_j = \sum_{j=0}^{\bar{k}} 2^j \sum_{l=1}^{m} n_j^l \le 2F^{OPT}$. Therefore, we have $m\mathcal{I} \le 2F^{OPT}$.

To prove that $F(n) = O(\sum_{j=0}^{\bar{k}}(\bar{k} - j)I_j + 2F^{OPT}) = O(\log \frac{n}{m})F^{OPT}$, we study the following optimization problem:

$$\begin{aligned} \max_{\{I_0,\ldots,I_{\bar{k}}\}} F(n) &= \sum_{j=0}^{\bar{k}} (\bar{k} - j)I_j \\ n &\ge \sum_{j=0}^{\bar{k}} \frac{I_j}{2^j} \\ \mathcal{I} &\ge \frac{1}{m} \sum_{j=0}^{\bar{k}} I_j, \end{aligned}$$

where the first constraint holds since $\frac{I_j}{2^j} = \sum_{l=1}^{m} n_j^l \le n$.

We rewrite the problem using variables $Y_j = \sum_{i \le j} I_i$, $j \ge 0$, with the convention $Y_{-1} = 0$:

$$\begin{aligned} \max_{\{Y_0,\ldots,Y_{\bar{k}}\}} F(n) &= \sum_{j=0}^{\bar{k}} Y_j \\ n &\ge \sum_{j=0}^{\bar{k}} \frac{Y_j - Y_{j-1}}{2^j} \ge \sum_{j=0}^{\bar{k}} \frac{Y_j}{2^{j+1}} \\ m\mathcal{I} &\ge Y_{\bar{k}} \ge Y_{\bar{k}-1} \ge \ldots . \end{aligned}$$

The objective function is maximized assigning $m\mathcal{I} = Y_{\bar{k}} = Y_{\bar{k}-1} = Y_{\bar{k}-2} = \cdots = Y_{\bar{k}-l}$, and $0 = Y_{\bar{k}-l-1} = \cdots = Y_0$ with l being the minimum integer such that the second constraint is tight or violated, namely the minimum integer such that $\frac{m\mathcal{I}}{2^{\bar{k}-l-1}} \ge n$.

We then compute a value l such that $2^{l+1} = \frac{n}{m}\frac{2^{\bar{k}}}{\mathcal{I}}$. Since $2^{\bar{k}} \le \mathcal{I}$ we have $l = O(\log \frac{n}{m})$, thus yielding $F(n) = O(\log \frac{n}{m}) m\mathcal{I} = O(\log \frac{n}{m}) F^{OPT}$ that completes the proof. $\square$

The bound obtained from Claim 15 holds for any σ. Applying this claim in Eq. (7), we obtain our second result:

THEOREM 16. *RMLF is a $O(\log n \log \frac{n}{m})$ competitive nonclairvoyant randomized algorithm for minimizing the total flow time on parallel machines.*

Considering Eq. (7) with $m = 1$, we obtain our tight result for the single machine case:

THEOREM 17. *RMLF is a $O(\log n)$ competitive nonclairvoyant randomized algorithm for minimizing the total flow time on a single machine.*

5. *Open Problems*

There is still a logarithmic gap between the competitive result on parallel machines and the $\Omega(\log \frac{n}{m})$ randomized lower bound for the case in which the processing time of a job is known at release time [Leonardi and Raz 1997]. However, our conjecture is that alike the case of a single machine, the lack of knowledge about the processing times of the jobs leads to a logarithmic overhead.

The fact that a randomized version of MLF achieves optimal or almost optimal performances from a worst case point of view may be a good indication that MLF is also very efficient on input sequences drawn from specific probability distributions like the uniform or the exponential ones. This might be a further validation of the goodness of MLF in practice.

We finally mention nonclairvoyant minimization of other flow time related metrics, like the average stretch, that is, $\sum_{j \in J} F_j / p_j$, for which constant competitive algorithms for the clairvoyant case have been proposed [Becchetti et al. 2004; Gehrke et al. 1999].

ACKNOWLEDGMENT. We are immensely grateful to Kirk Pruhs for his help in a delicate phase of this work. We would also like to thank A. Marchetti-Spaccamela and S. Muthukrishnan for several discussions on these topics, and an anonymous referee for helpful comments and suggestions. We are also grateful to Guido Schäfer and Tjark Vredeveld for reading the article and giving us useful comments.

REFERENCES

ACHARYA, S., MUTHUKRISHNAN, S., AND SUNDARAM, G. 1999. Scheduling data delivery over multiple channels. Tech. rep., Bell Labs Technical Memo. (Available from `acharya@research.bell-labs.com`.)

AWERBUCH, B., AZAR, Y., LEONARDI, S., AND REGEV, O. 2002. Minimizing the flow time without migration. *SIAM J. Comput. 31*, 5, 1370–1382.

BAKER, K. R. 1974. *Introduction to Sequencing and Scheduling*. Wiley, New York.

BANSAL, N., DHAMDHERE, K., KONEMANN, J., AND SINHA, A. 2003. Non-clairvoyant scheduling for mean slowdown. In *Proceedings of the 20th Symposium on Theoretical Aspects of Computer Science (STACS)*. 260–270.

BANSAL, N., AND PRUHS, K. 2003. Server scheduling in the L_p norm: A rising tide lifts all boats. In *Proceedings of the 35th Symposium on Theory of Computing (STOC)*. ACM, New York, 242–250.

BECCHETTI, L., LEONARDI, S., AND MUTHUKRISHNAN, S. 2004. Scheduling to minimize average stretch without migration. *J. Comput. Syst. Sci. 68*, 80–95.

BEN-DAVID, S., BORODIN, A., KARP, R., TARDOS, G., AND WIGDERSON, A. 1994. On the power of randomization in on-line algorithms. *Algorithmica 11*, 2–14.

BENDER, M., MUTHUKRISHNAN, S., AND RAJARAMAN, R. 2002. Improved algorithms for stretch scheduling. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. ACM, New York, 762–771.

BENDER, M. A., CHAKRABARTI, S., AND MUTHUKRISHNAN, S. 1998. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM, New York, 270–279.

DOEPPNER, T. W. 1987. Threads, a system for the support of concurrent programming. Tech. Rep. CS-87-11, Brown University.

GEHRKE, J. E., MUTHUKRISHNAN, S., RAJARAMAN, R., AND SHAHEEN, A. 1999. Online scheduling to minimize avarage strech. In *Proceedings of the 40th Annual IEEE Symposium Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, Los Alamitos, Calif., 433–443.

IEEE. 1994. Institute for Electrical and Electronic Engineers. POSIX P1003.4a, Threads Extensions for Portable Operating Systems.

KALYANASUNDARAM, B., AND PRUHS, K. 1995. Speed is as powerful as clairvoyance. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*. IEEE Computer Society Press, Los Alamitos, Calif., 214–223.

KALYANASUNDARAM, B., AND PRUHS, K. 1997. Minimizing flow time nonclairvoyantly. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science* (Miami Beach, Fla., Oct. 20–22). IEEE Computer Society Press, Los Alamitos Calif., 345–352.

LEONARDI, S., AND RAZ, D. 1997. Approximating total flow time on parallel machines. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*. ACM, New York, 110–119.

MOTWANI, R., PHILLIPS, S., AND TORNG, E. 1994. Nonclairvoyant scheduling. *Theoret. Comput. Sci. 130*, 1, 17–47.

MUELLER, F. 1993. A library implementation of POSIX threads under UNIX. In *Proceedings of the USENIX Winter 1993 Technical Conference*. 29–42.

NUTT, G. 1999. *Operating System Projects using Windows NT*. Addison-Wesley, Reading, Mass.

SLEATOR, D. D., AND TARJAN, R. E. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM 28*, 202–208.

SUN. 1993. Sunos 5.3 system services.

TANENBAUM, A. S. 1992. *Modern Operating Systems*. Prentice-Hall, Englewood Cliffs, N.J.

RECEIVED FEBRUARY 2002; REVISED DECEMBER 2003; ACCEPTED DECEMBER 2003