Minimizing Total Flow Time and Total Completion Time with Immediate Dispatching

Nir Avrahami [*] Yossi Azar [†]

February 5, 2003

Abstract

We consider the problem of scheduling jobs arriving over time in a multiprocessor setting, with *immediate dispatching*, disallowing job migration. The goal is to minimize both the total flow time (total time in the system) and the total completion time.

Previous studies have shown that while preemption (interrupt a job and later continue its execution) is inherent to make a scheduling algorithm efficient, migration (continue the execution on a different machine) is not. Still, the current non-migratory online algorithms suffer from a need for a central queue of unassigned jobs which is a "no option" in large computing system, such as the Web.

We introduce a simple online non-migratory algorithm IMD, which employs *immediate dispatching*, i.e., it immediately assigns released jobs to one of the machines. We show that the performance of this algorithm is within a logarithmic factor of the optimal *migratory offline* algorithm, with respect to the total flow time, and within a small constant factor of the optimal *migratory offline* algorithm, with respect to the total completion time. This solves an open problem suggested by Awerbuch et al [STOC99].

1 Introduction

Almost all classical work on scheduling of jobs released over time in a multiprocessor setting assumes that unassigned jobs are held in a central queue. The decision on assignment of a job is not done upon its arrival but postponed until the dispatcher acquires enough information. In many cases, such as in large computing systems (e.g. the WEB), this is impossible since the the number of unassigned jobs (with their associated data) may be large, requiring huge amount of resources (e.g. memory). Moreover, the delay in transferring the job to the appropriate machine may be large resulting in dramatic deterioration of the performance. Hence, the architecture of many systems requires from the dispatcher to assign a job immediately upon its arrival to one of the machines without maintaining a central queue. Each job is kept in the queue of the machine it was assigned to.

[*]Department of Computer Science, Tel Aviv University, Tel-Aviv, 69978, Israel. E-Mail: nirav@post.tau.ac.il.

[†]Department of Computer Science, Tel Aviv University, Tel-Aviv, 69978, Israel. E-Mail: azar@post.tau.ac.il.
Research supported in part by the Israel Science Foundation and by the IST Program of the EU.

In the classical multiprocessor scheduling problem, preemptive and non-preemptive schedules are often considered, in the context of minimizing the two most basic performance measures, the total flow time (overall time the jobs are spending in the system) and total completion time. These measures capture both the overall quality of service of the system and fairness of service. Since preemption was shown to be inherent to the problem of minimizing the total flow time (as noted below), while it is problematic in real multiprocessors systems, an intermediate model which disallows migration was considered. Current non-migratory online algorithms, which were devised to work in this model, tend to delay the assignment of jobs, in order to avoid early commitment to machines, hence they are required to maintain a pool of unassigned jobs. As already mentioned this may be "no option" in many architectures. Hence, the obvious question is whether one can devise an efficient algorithm that dispatches each job to a machines upon its release time. Note that this results in splitting the multiprocessor scheduling problem into two axis: the assignment problem and the single machine scheduling problem. Our somewhat surprising result shows that we can actually achieve almost the same performance for total flow time and for total completion time in the immediate dispatching model as in the model that maintains a central queue.

Our results: multiple processors with immediate dispatching. We introduce a simple *non-migratory online* algorithm IMD, which employs *immediate dispatching*, i.e., it immediately assigns released jobs to one of the machines. We show that:

- The total flow time of algorithm IMD is within the $O(\min\{\log P, \log n\})$ factor of the total flow time of the optimal *migratory offline* algorithm. This solves an open problem suggested by Awerbuch et al [2].
- The total completion time of algorithm IMD is at most 7 times the total completion time of the optimal *migratory offline* algorithm.
- For the measure of total completion time, preemption can be eliminated from algorithm IMD, resulting in algorithm IMD' which is at most 14 times the total completion time of the optimal *migratory offline* algorithm.

Existing work: total flow time. Surveys on approximation algorithms for scheduling can be found in [9, 12]. In the non-preemptive case it is impossible to achieve a "reasonable" approximation for the total flow time. Specifically, even for one machine one cannot achieve an approximation factor of $O(n^{\frac{1}{2}-\epsilon})$ unless $NP = P$ where n is the number of jobs [11]. For $m > 1$ it is impossible to achieve $O(n^{\frac{1}{3}-\epsilon})$ approximation factor unless $NP = P$ [13]. Thus, preemption really seems to be essential. Minimizing the total flow time on one machine with preemption can be done optimally in polynomial time using the natural algorithm shortest remaining processing time ($SRPT$) [3]. For more than one machine the preemptive problem becomes NP-hard [7]. Leonardi and Raz [13] 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 fact, in the online setting $SRPT$ is optimal, i.e., no algorithm can achieve a better bound up to a constant factor [13]. Note that $SRPT$ requires migration. In addition it decides to assign a job from a central pool only when a machine becomes empty.

Awerbuch et al [2] presented an online non-migratory algorithm, which performs almost as well as the best known *offline* algorithm ($SRPT$) for the preemptive problem that uses migration. This algorithm performs by at most $O(\min\{\log P, \log n\})$ factor of the optimal total flow time of any (possibly migratory) schedule. Chekuri et al. [5] designed a variant of the above algorithm which slightly improves the performance ratio to $O(\min\{\log P, \log(n/m)\})$ and matches the performance bound of $SRPT$. The above algorithms overcome the problem of migration. However, many jobs may be kept in a central pool until it is justified to assign them to machines. Postponing the assignment jobs by the dispatcher and maintaining them in the central pool is crucial for their algorithms. As already mentioned, non-immediate dispatching is not an option in many systems due to the sizes of the jobs and the delay in the network.

Existing work: total completion time. For a single machine with preemption $SRPT$ is optimal. In the multiprocessor setting $SRPT$ is 2 competitive [14]. Without preemption the best online deterministic algorithm for a single machine is 2 competitive [14, 10]. Moreover, this is optimal [10]. The best randomized algorithm is $e/(e-1)$ competitive [6] and this is optimal [15]. In the multiprocessor setting (without preemption) the best algorithm is 2.89 competitive [4, 8]. In the offline problem a PTAS for minimizing the total completion time was given for the preemptive and non-preemptive versions for a single and multiple machine [1]. All known algorithms for total completion time prior to our work require postponing the decision of the assignment and hence maintain a central pool.

Techniques. One may tempt to think that the natural approach for designing an immediate dispatching algorithm should be based on $SRPT$ or the non-migratory algorithm with a central queue. Specifically, we may try to predict for each job upon its arrival on which machine those algorithms would have assigned the job and dispatch the job immediately to that machine. The prediction would be based on the given current information, i.e., the exact residual size of all current jobs, assuming no additional jobs will arrive. Unfortunately, it is possible to show that this approach results in algorithms with poor performance. Hence a new algorithm had to be developed.

In contrast to previous algorithms (e.g $SRPT$) our algorithm IMD prefers to ignore some of the given information, reducing the communication traffic and simplifying its implementation. The main idea of IMD is to assign each job immediately on its release time so as to balance the accumulative volume of all similar jobs from time zero until the current time. Hence, it ignores the information of which jobs were already completed and the current residual volume of jobs left to be processed. Moreover, algorithm IMD ignores the exact release times of the jobs and would produce the same assignment even for different release times of the jobs as long as their relative arrival order is maintained. Hence, IMD maintains only a small amount of information about previous assignments. Moreover, the clock of the dispatcher does not need to be synchronized with the clocks of the processors. Interestingly, by ignoring information we are able to show that the residual volume of jobs left to be processed at any other given time will almost be the same on any machine, implying that the idle times are also balanced between the machines.

Since on a single machine $SRPT$ is optimal, the best algorithm that uses immediate assignment will use $SRPT$ on each machine separately, independently of the assignment strategy. Aside from the assignment strategy, which is the core of the algorithm, we also use a different (less effective) scheduling approach for each machine separately, in order to simplify the analysis of the flow time performance. The total flow time analysis basically combines new ideas with ideas used in [2, 5]. The algorithm of [2] also uses classification of jobs to classes. However, that classification is done according to the residual sizes of the jobs at any given time, meaning that the classification of a job changes along the process. In contrast, it is crucial for our algorithm to use a different classification, which is similar to the group partition in [5], and is based on a job size on its arrival. Hence, our classification does not change along the process.

As for the the total completion time, one should also notice that our algorithm tackles the problem using a technique that is substantially different from the standard techniques, such as: $SRPT$, time partitioning into intervals ($GreedyInterval$) or by solving some migratory schedule and converting it into a non-migratory schedule.

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 our model the assignment of job j to some machine should be immediate on its release time r_j, but it needn't be executed immediately on assignment. Our model allows preemption but does not allow migration. The scheduling algorithm decides which of the jobs should be executed at each time. Clearly a machine can process at most one job in 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 F is $\sum_{j \in J} F_j$, and the total completion time C is $\sum_{j \in J} C_j$. The goal of the scheduling algorithm is to minimize the total flow time (or completion time) for each given instance of the problem. In the offline version of the problem all the jobs are known in advance. In the online 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.

2 Definitions and Notations

We start by giving a few definitions and notation, which will be useful both for the algorithm definition and analysis.

- We first define the class of a job j to be k, if its size on its arrival p_j is in $[2^k, 2^{k+1})$. Note that the classification to classes does not change during the process (similar to the group partition in [5]). Denote by k_{min} and k_{max} the extremes of the jobs classes.
- $\mathcal{T}$ is used to denote the time period where all the m machines are busy (non idle).
- Denote by P the ratio of the longest job to the shortest one.
- Several function of time are used:
  - $U(t)$ denotes the cumulative sum of size of jobs arrived prior to t (sum of their size on their arrival).
  - $P(t)$ denotes the total volume of jobs that have already been processed till time t.
  - $R(t)$ denotes the total remaining volume of jobs to be processed at time t.
  - $\gamma(t)$ denotes the number of non-idle machines at time t.

- $\delta(t)$ denotes the number of jobs (with $r_i \leq t$), which are alive at time t (i.e., not finished yet).
- $n(t)$ denotes the number of jobs released by time t.
- $c(t)$ denotes the number of completed jobs by time t.
- $J(t)$ denotes the set of jobs that were finished by time t.

We note that if a function is used without the time parameter t then it refers to the function at the end of the schedule.

- Several function modifiers are used:
  - For a generic function f, the notation f^S refers to the value of f when the scheduler is S. We denote our scheduler by IMD, while the optimal migratory offline scheduler will be denoted by OPT. We may omit this superscript when it refers to IMD.
  - For a generic function f, the notations $f_{=k}$, $f_{<k}$ etc. refer to the function f restricted to the set of jobs that belong to the subscript classes.
  - For a generic function f, the notation f^i refers to the function f restricted to the set of jobs that were assigned to the i^{th} machine. When the scheduler is OPT, $f^{OPT,i}$ is defined as the average $\frac{1}{m} f^{OPT}$.
  - For a generic function f, we use f^{ij} as a short form of $f^i - f^j$.
  - For a generic function $f(t)$ we use $\Delta f(t) = f(t) - f^{OPT}(t)$ denoting the difference between our scheduler and the optimal offline scheduler.
  - For a generic function $f(t)$ we use $f(J, t)$ when the input set of jobs J is not clear from the context.

3 The Algorithm

Recall from the above definitions that jobs are classified according to their sizes. Job is of class k if its size is between 2^k and 2^{k+1}. Also, by the definitions above $U^i_{=k}(t)$ denotes the total cumulative sum of the original size of jobs that arrived prior to t and were assigned to the machine i. Next we define our *immediate dispatching* algorithm:

Algorithm *IMD*:

- On arrival time t of a new job of class k, assign it to a machine i with minimum $U^i_{=k}(t)$.
- Conducted $SRPT$ on each machine separately.

The algorithm IMD balances the total volume of jobs of a specific class that were ever assigned to the machines. We note that the assignment decisions are independent of the exact release times of the previous jobs and only depend on their order. Hence, the assignment decisions are not based on the current status of the jobs in the queues of the machines.

4 Total Flow Time Analysis

In this section we prove that the total flow time of algorithm IMD is within the $O(\min\{\log P, \log n\})$ factor of the total flow time of the optimal *migratory offline* algorithm. We state a different scheduling principal for each single machine (see [5]), which is clearly less effective than $SRPT$ (see [3]), and analyze it. This is done to simplify the analysis.

The processing on the i^{th} machine will be conducted according to the following principal: Process the job with the earliest arrival time among the set of jobs of the smallest class k with unfinished jobs ($R^i_{=k}(t) > 0$).

We first observe the simple fact that the total flow time is the integral over time of the number of jobs that are alive (for example, see [13]):

Fact 4.1 *For any scheduler S,*

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

We start the analysis focusing on the $O(\log P)$ bound. In this part we are about to distinguish between times where all the machines are working ($t \in \mathcal{T}$), and times where at least one machine is idle ($t \notin \mathcal{T}$). For each of these cases we bound the number of alive jobs $\delta^{IMD}(t)$ and finally we will compute the integral of Fact 4.1.

At this stage we show that the total remaining processing time (for each class) is almost the same on the different machines at any given time.

Observation 4.2 *For any time t and any two machines i and j we have $|U^{ij}_{=k}(t)| \leq 2^{k+1}$ and hence also $|U^{ij}_{\leq k}(t)| \leq 2^{k+2}$.*

Proof: The first inequality holds since all the jobs of class k are of size $\leq 2^{k+1}$. The second inequality follows obviously. ■

Lemma 4.3 *For any t, the difference between the volume of jobs that have already been processed, on any two different machines i and j is bounded as follows: $|P^{ij}_{\leq k}(t)| \leq 2^{k+2}$.*

Proof: Assume that t_0 is the first time $|P^{ij}_{\leq k}(t)|$ gets bigger than 2^{k+2}, hence, $|P^{ij}_{\leq k}(t_0)| = 2^{k+2}$ and for any small enough $\epsilon > 0$, $|P^{ij}_{\leq k}(t_0 + \epsilon)| > 2^{k+2}$. This means that exactly one of these machines process jobs of classes not bigger than k (otherwise the difference value does not change), assume it's machine i. Since the algorithm always processes a job from the smallest class on each machine, machine j must have already processed all of the jobs of classes $\leq k$ by t_0 while machine i did not finished to process all the jobs of classes $\leq k$. Hence

$$U^j_{\leq k}(t_0) = P^j_{\leq k}(t_0) < P^i_{\leq k}(t_0) < U^i_{\leq k}(t_0)$$

which yields

$$2^{k+2} = |P^{ij}_{\leq k}(t_0)| < |U^{ij}_{\leq k}(t_0)| \ .$$

This contradicts Observation 4.2. ■

Lemma 4.4 *For any t, the difference between the residual volume of jobs that needs to be processed, on any two different machines i and j is bounded as follows $|R^{ij}_{\leq k}(t)| \leq 2^{k+3}$*

Proof: Combining Observation 4.2, Lemma 4.3 and the fact that $R(t) = U(t) - P(t)$ by definition, we get

$$|R^{ij}_{\leq k}(t)| \leq |U^{ij}_{\leq k}(t)| + |P^{ij}_{\leq k}(t)| \leq 2^{k+3} .$$

■

We handle the case where at least one of the machines is idle ($t \notin \mathcal{T}$), implying that the other machines are not heavily loaded.

Lemma 4.5 *For any $t \notin \mathcal{T}$, the number of jobs from the range $[k_1, k_2]$ of classes on any machine i can be bounded as follows: $\delta^i_{[k_1,k_2]}(t) \leq 9(k_2 - k_1 + 1)$.*

Proof: Since $t \notin \mathcal{T}$, there exists a machine j, which is idle (i.e., with $R^j(t) = 0$). Obviously for any k, $R^j_{\leq k}(t) = 0$. By Lemma 4.4 we get that for any (non-idle) machine i, $R^i_{\leq k}(t) \leq 2^{k+3}$, and obviously also $R^i_{=k}(t) \leq 2^{k+3}$ follows. Since the algorithm process the job with the earliest arrival time among a set of jobs from the same class k, we can deduce that on machine i, there is at most one job from class k with remaining processing time $< 2^k$. Hence, we bound the number of jobs of class k at this time by $\delta^i_{=k}(t) \leq \frac{R^i_{=k}(t)}{2^k} + 1 \leq 8 + 1 = 9$. The result follows. ■

Corollary 4.6 *For any $t \notin \mathcal{T}$, the number of jobs in the whole system can be bounded as follows: $\delta(t) \leq 9\gamma(t)(\log P + 2)$.*

Proof: The result follows immediately from Lemma 4.5 with $k_2 = k_{max}$ and $k_1 = k_{min}$ and the fact that the number of classes $k_{max} - k_{min} + 1$ is smaller than $\log P + 2$. ■

Now, Assume none of the machines is idle ($t \in \mathcal{T}$), and let $\hat{t} < t$ the earliest time s.t. $[\hat{t}, t) \subset \mathcal{T}$. Define t_k to be the last time a job from a class bigger than k was processed in this range (in case only jobs of classes $\leq k$ were processed throughout $[\hat{t}, t)$ we set $t_k = \hat{t}$).

Lemma 4.7 *For $t \in \mathcal{T}$, $\Delta R_{\leq k}(t) \leq \Delta R_{\leq k}(t_k)$.*

Proof: By definition of t_k, it is obvious that the algorithm process only jobs whose class is at most k in the range $[t_k, t)$ on all machines, therefore the offline algorithm cannot process a bigger share of these jobs. Note also that the release of new jobs of classes $\leq k$ does not affect the value of $\Delta R_{\leq k}$, hence it may only decrease in the range $[t_k, t)$. ■

Lemma 4.8 *For $t \in \mathcal{T}$, $\Delta R_{\leq k}(t_k) \leq m2^{k+3}$.*

Proof: From the definition of t_k it follows that there exist a machine i s.t. for every small enough $\epsilon > 0$, $R^i_{\leq k}(t_k - \epsilon) = 0$. This is either the machine that processed the last job of class bigger than k in the range $[\hat{t}, t)$ or alternatively the machine that was last idle (in case $t_k = \hat{t}$). Hence by Lemma 4.4, any other machine j complies with $R^j_{\leq k}(t_k - \epsilon) \leq 2^{k+3}$, yielding also $\Delta R^j_{\leq k}(t_k - \epsilon) \leq 2^{k+3}$. Since jobs which arrive exactly at t_k increment R also for the offline algorithm, not affecting ΔR, we get $\Delta R_{\leq k}(t_k) \leq m2^{k+3}$ ■

Lemma 4.9 *For* $t \in \mathcal{T}$, $\Delta R_{\le k}(t) \le m2^{k+3}$.

Proof: Combining Lemmas 4.7 and 4.8 yields $\Delta R_{\le k}(t) \le \Delta R_{\le k}(t_k) \le m2^{k+3}$. ∎

Lemma 4.10 *For* $t \in \mathcal{T}$, *for any machine* i, $\Delta R^i_{\le k}(t) \le 2^{k+4}$.

Proof: From Lemma 4.9 we have that: $\min_j \Delta R^j_{\le k}(t) \le 2^{k+3}$. From Lemma 4.4 we derive also: $|\Delta R^{ij}_{\le k}(t)| = |R^{ij}_{\le k}(t)| \le 2^{k+3}$. Combining the above yields $\Delta R^i_{\le k}(t) \le \min_j \Delta R^j_{\le k}(t) + |\Delta R^{ij}_{\le k}(t)| \le 2^{k+4}$. ∎

Lemma 4.11 *For any* $t \in \mathcal{T}$, *the number of jobs from the range* $[k_1, k_2]$ *of classes on any machine* i *can be bounded as follows:* $\delta^i_{[k_1,k_2]}(t) \le 9(k_2 - k_1 + 2) + 2\delta^{OPT,i}_{\le k_2}(t)$.

Proof: We count the number of jobs on machine i by class, and bound it as follows:

$$
\begin{aligned}
\delta^i_{[k_1,k_2]}(t) &= \sum_{j=k_1}^{k_2} \delta^i_{=j}(t) \\
&\le \sum_{j=k_1}^{k_2} \{\frac{\Delta R^i_{=j}(t) + R^{OPT,i}_{=j}(t)}{2^j} + 1\} \\
&= \sum_{j=k_1}^{k_2} \frac{\Delta R^i_{\le j}(t) - \Delta R^i_{\le j-1}(t)}{2^j} + (k_2 - k_1 + 1) + \sum_{j=k_1}^{k_2} \frac{R^{OPT,i}_{=j}(t)}{2^j} \\
&\le \frac{\Delta R^i_{\le k_2}(t)}{2^{k_2}} + \sum_{j=k_1}^{k_2-1} \frac{\Delta R^i_{\le j}(t)}{2^{j+1}} - \frac{\Delta R^i_{\le k_1-1}(t)}{2^{k_1}} + (k_2 - k_1 + 1) + 2\delta^{OPT,i}_{[k_1,k_2]}(t) \\
&\le 16 + \sum_{j=k_1}^{k_2-1} 8 + \delta^{OPT,i}_{\le k_1-1}(t) + (k_2 - k_1 + 1) + 2\delta^{OPT,i}_{[k_1,k_2]}(t) \\
&\le 9(k_2 - k_1 + 2) + 2\delta^{OPT,i}_{\le k_2}(t)
\end{aligned}
$$

where the second line is due to the fact that there is at most one job on machine i of each class k with a residual volume less than 2^k. The fourth line is derived from the fact that the residual of each job of class k is smaller than 2^{k+1} by definition. The fifth line is derived by applying Lemma 4.10. ∎

Corollary 4.12 *For any* $t \in \mathcal{T}$, *the number of jobs in the whole system can be bounded as follows:* $\delta(t) \le 9m(\log P + 3) + 2\delta^{OPT}(t)$.

Proof: First note that $k_{max} - k_{min} + 2 \le \log P + 3$. Now we apply Lemma 4.11 with $k_2 = k_{max}$ and $k_1 = k_{min}$ and sum over all the machines, which yields the result. ∎

We prove the $O(\log P)$ approximation ratio.

Theorem 4.13 $F^{IMD} = O(\log P) \cdot F^{OPT}$, *i.e., algorithm* IMD *has a logarithmic approximation factor, w.r.t the maximum ratio between jobs size, even when compared to the best (possibly migratory) offline algorithm.*

Proof:

$$
\begin{aligned}
F^{IMD} &= \int_t \delta(t)dt \\
&= \int_{t\notin\mathcal{T}} \delta(t)dt + \int_{t\in\mathcal{T}} \delta(t)dt \\
&\leq \int_{t\notin\mathcal{T}} 9(2+\log P)\gamma(t)dt + \int_{t\in\mathcal{T}} (9m(\log P+3) + 2\delta^{OPT}(t))dt \\
&= 9(2+\log P)\int_{t\notin\mathcal{T}} \gamma(t)dt + 9(\log P+3)\int_{t\in\mathcal{T}} \gamma(t)dt + 2\int_{t\in\mathcal{T}} \delta^{OPT}(t)dt \\
&\leq 9(\log P+3)\int_t \gamma(t)dt + 2\int_t \delta^{OPT}(t)dt \\
&\leq (29 + 9\log P)\cdot F^{OPT}
\end{aligned}
$$

where the first equality is from the definition of F^{IMD}. The second equality is obtained by looking at times in which none of the machines is idle and at times in which at least one machine is idle, separately. The third line uses Corollaries 4.6 and 4.12. The forth line is true by definition of $\mathcal{T}$. Finally, $\int_t \gamma^{IMD}(t)dt$ is the total time spent processing jobs by the machines which is exactly the sum of all jobs. This sum is upper bounded by the total flow time of OPT since each job's flow time must be at least its processing time. ■

We now turn to prove the $O(\log n)$ bound. We start this part focusing on a single machine i. We define $\bar{k}^i$ to be the maximal class of a job assigned to i throughout the process. Define τ_k^i to be the set of time units, in which machine i processed a job of class k.

Lemma 4.14 *The flow time of all jobs assigned to machine i can be bounded as follows:*

$$F^{IMD,i} \leq 18\sum_{j=k_{min}}^{\bar{k}^i} (\bar{k}^i - j)n^i_{=j}2^j + 18U^i_{[k_{min},\bar{k}^i]} + 2F^{OPT,i}$$

Proof: We compute the integral of Fact 4.1 according to the time partition to τ_k^i.

$$
\begin{aligned}
F^{IMD,i} &= \int_t \delta^i(t)dt \\
&= \sum_{j=k_{min}}^{\bar{k}^i} \int_{t\in\tau_j^i} \delta^i_{[j,\bar{k}^i]}(t)dt \\
&\leq \sum_{j=k_{min}}^{\bar{k}^i} \int_{t\in\tau_j^i} \{9(\bar{k}^i - j + 2) + 2\delta^{OPT,i}_{\leq\bar{k}^i}(t)\}dt \\
&\leq \sum_{j=k_{min}}^{\bar{k}^i} 9(\bar{k}^i - j + 2)U^i_{=j} + 2F^{OPT,i} \\
&\leq 18\sum_{j=k_{min}}^{\bar{k}^i} (\bar{k}^i - j)n^i_{=j}2^j + 18U^i_{[k_{min},\bar{k}^i]} + 2F^{OPT,i}
\end{aligned}
$$

where the second equality is by definition of τ_k^i. The third line is derived from Lemmas 4.5 and 4.11. The fourth line is by $|\tau_k^i| = U_{=k}^i$. The fifth line is since the jobs of class k are smaller than 2^{k+1}. ∎

To continue, we use a technical lemma proved in [2] with its proof.

Lemma 4.15 *Given a sequence $a_1, a_2, \ldots$ 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 $\sum_{i\geq 1} i a_i \leq A \log(4B/A)$.*

Proof: Define a second sequence, $b_i = \sum_{j\geq i} a_j$ for $i \geq 1$. Then it is known that $A \geq b_1 \geq b_2 \geq\geq \ldots b_i$. 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. ∎

Lemma 4.16 *For any machine i, $\sum_{j=k_{min}}^{\bar{k}^i} (\bar{k}^i - j) n_{=j}^i 2^j \leq U^i \log(4n^i)$*

Proof: We exchange variables by $l = \bar{k}^i - j$ and define $I_l = n_{=\bar{k}^i - l}^i 2^{\bar{k}^i - l} = n_{=j}^i 2^j$. Note that $\sum_{l=0}^{\bar{k}^i - k_{min}} I_l \leq U^i$ and also $\sum_{l=0}^{\bar{k}^i - k_{min}} 2^l I_l = \sum_{l=0}^{\bar{k}^i - k_{min}} 2^l n_{\bar{k}^i - l}^i 2^{\bar{k}^i - l} = n^i 2^{\bar{k}^i}$.

We apply Lemma 4.15 to our problem using $a_l = I_l$, $l = 0, \ldots, \bar{k}^i - k_{min}$, $A = U^i$ and $B = n^i 2^{\bar{k}^i}$ and obtain $\sum_{j=k_{min}}^{\bar{k}^i} (\bar{k}^i - j) n_{=j}^i 2^j = \sum_{l=0}^{\bar{k}^i - k_{min}} l I_l \leq U^i \log(\frac{4n^i 2^{\bar{k}^i}}{U^i}) = U^i \log(4n^i)$ due to the fact that $2^{\bar{k}^i} \leq U^i$ by definition of $\bar{k}^i$. ∎

We prove the $O(\log n)$ approximation ratio.

Theorem 4.17 $F^{IMD} = O(\log n) \cdot F^{OPT}$, *i.e., algorithm IMD has a logarithmic approximation factor, w.r.t the number of jobs n, even when compared to the best (possibly migratory) offline algorithm.*

Proof: We sum over the different machines contribution to the total flow.

$$\begin{aligned}
F^{IMD} &= \sum_{i=1}^{m} F^{IMD,i} \\
&\leq \sum_{i=1}^{m} \{18 \sum_{j=k_{min}}^{\bar{k}^i} (\bar{k}^i - j) n_{=j}^i 2^j + 18 U_{[k_{min}, \bar{k}^i]}^i + 2F^{OPT,i}\} \\
&\leq 18 \sum_{i=1}^{m} U^i \log(4n^i) + 18U + 2F^{OPT} \\
&\leq O(\log n) \sum_{i=1}^{m} U^i + 20F^{OPT} = O(\log n) F^{OPT}
\end{aligned}$$

where the second line is due to Lemma 4.14 and the third line is due to Lemma 4.16. ∎

5 Total Completion Time Analysis

We prove that the total completion time of algorithm IMD is at most 7 times the total completion time of the optimal *migratory offline* algorithm. Later we show how to eliminate the preemption and construct an algorithm IMD' which is at most 14 times the total completion time of the optimal *migratory offline* algorithm. The analysis of this section appears in the Appendix.

6 Conclusions

In this paper we considered the problem of finding a preemptive schedule that optimizes both the total flow time and the total completion time of a set of jobs released over time, when the assignment of jobs to machines should be immediate disallowing job migration. We presented a new online algorithm that is still within a logarithmic factor of the best (possibly migratory) offline algorithm with respect to the total flow time. This algorithm also achieves a small constant approximation factor of the best offline algorithm with respect to the total completion time.

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. In *IEEE Symposium on Foundations of Computer Science*, pages 32–44, 1999.

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

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

[4] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. In *Proc. 23rd International Colloquium on Automata, Languages and Programming*, pages 646–657, 1996.

[5] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 84–93, 2001.

[6] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. In *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 609–618, 1997.

[7] 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.

[8] L. Hall, D. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proc. of 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 142–151, 1996.

[9] L.A. Hall. Approximation algorithms for scheduling. In D.S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 1–45. PWS publishing company, 1997.

[10] J. A. Hoogeveen and Arjen P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In *Proc. 5th Conf. Integer Programming and Combinatorial Optimization*, pages 404–414, 1996.

[11] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single maching. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 418–426, Philadelphia, Pennsylvania, 1996.

[12] E.L. Lawler, J.K Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. In *Handbooks in operations research and management science*, volume 4, pages 445–522. North Holland, 1993.

[13] 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.

[14] C. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. In *Proc. 4th Workshop on Algorithms and Data Structures*, pages 86–97, 1995.

[15] L. Stougie and A. Vestjens. Randomized on-line scheduling: How low can't you go, 1997.

7 Appendix: Total Completion Time Analysis

In this section we prove the ratio of 7 for algorithm IMD and 14 for the non-preemptive algorithm IMD'. We start by defining a fair schedule. We say that a schedule S is fair, if for any two jobs i and j with $p_i = p_j$, i finishes before j if and only if $r_i \leq r_j$. We now argue that there is a fair optimal schedule.

Lemma 7.1 *For any schedule S, there exist another schedule S', which is fair and which is not worse than S, with respect to the total completion time.*

Proof: We transform the schedule S into S' in stages. At each stage we choose a pair of jobs i and j, with $p_i = p_j$ and $r_i < r_j$, which is not scheduled fairly, i.e., job j finishes before job i. We denote by T_1 the time period when only one of these jobs was processed. Let $T_{1,i}$ be the time period when only job i was processed and let $T_{1,j}$ be the time period when only job j was processed. We assign the first $|T_{1,i}|$ time units of T_1 to job i and the last $|T_{1,j}|$ units of T_1 to job j. First note that this assignment is feasible, moreover this pair is scheduled fairly, while only improving the completion time of the first job to be completed of the two (the second job completion time remains unchanged). After a finite number of stages this process stops, yielding S'. ∎

Corollary 7.2 *For any input jobs set J, there is a fair optimal schedule.*

Proof: By Lemma 7.1 there is another schedule S', which is not worse than OPT with respect to the total completion time that is also fair. Obviously S' is also optimal. ■

By Corollary 7.2, we can choose OPT to denote an optimal offline algorithm, which yields a schedule that is fair.

Recall that the input set of jobs is $J = \{(r_j, p_j)\}_{j=1}^n$. We compare the performance of IMD and OPT on J by examining the performance of OPT when running on another input set $\hat{J}$. We define this set by $\hat{J} = \{(2r_j, 2^{k_j+1})\}_{j=1}^n$, where k_j is the class of the j^{th} job in J.

Lemma 7.3 *For any input set of jobs J, $C^{OPT}(\hat{J}) \leq 2C^{OPT}(J)$.*

Proof: Let $J_2 = \{(2r_j, 2p_j)\}_{j=1}^n$. It is clear that any schedule on J can be translated by simple scaling to a schedule on J_2 and vice versa, hence $C^{OPT}(J_2) = 2C^{OPT}(J)$. On the other hand we have that $2^{k_j+1} \leq 2p_j$, therefore any schedule on J_2 is also a valid schedule on $\hat{J}$ yielding $C^{OPT}(\hat{J}) \leq C^{OPT}(J_2)$. Combining the above arguments yields $C^{OPT}(\hat{J}) \leq C^{OPT}(J_2) = 2C^{OPT}(J)$. ■

We now observe that the total completion time can be computed as an integral over time of the number of jobs that were not completed yet:

Observation 7.4 *For any scheduler S, $C^S = \int_t n - c^S(J,t)dt$.*

In view of this observation, we turn to show that for any t, algorithm IMD completes by time $3.5t$ at least the amount of jobs completed by OPT by time t when it runs on $\hat{J}$.

Recall that $J^{OPT}(\hat{J}, t)$ is the set of jobs that OPT finishes by time t when the input set of jobs is $\hat{J}$. We denote the corresponding jobs from J by $J^*(t)$.

Lemma 7.5 *For any time t, $c^{OPT}(\hat{J}, t) = c^{IMD}(J^*(t), 3.5t)$.*

Proof: First note that by definition $c^{OPT}(\hat{J}, t) = |J^{OPT}(\hat{J}, t)| = |J^*(t)|$, furthermore, it is clear that for any other time t', $|J^*(t)| \geq c^{IMD}(J^*(t), t')$, hence $c^{OPT}(\hat{J}, t) \geq c^{IMD}(J^*(t), 3.5t)$. It is left to prove that $c^{OPT}(\hat{J}, t) \leq c^{IMD}(J^*(t), 3.5t)$.

Note that all the jobs in $J^*(t)$ are released before time $\frac{t}{2}$. By definition all the jobs in $J^*(t)$ are smaller than their corresponding jobs in $J^{OPT}(\hat{J}, t)$, consequently $U(J^*(t), \frac{t}{2}) \leq U(J^{OPT}(\hat{J}, t), t)$. By standard averaging argument, we deduce that $\min_i\{U^i(J^*(t), \frac{t}{2})\} \leq \frac{1}{m}U(J^{OPT}(\hat{J}, t), t) \leq t$.

Let k_{low} and k_{high} be the extreme classes of jobs in $J^*(t)$. Hence, the biggest job in $J^{OPT}(\hat{J}, t)$ is of size $2^{k_{high}+1}$. Since OPT finishes its corresponding job by time t, we also have that $2^{k_{high}+1} \leq t$.

Applying lemma 4.2, we bound the total volume difference between the machines as follows: $U^i(J^*(t), \frac{t}{2}) = U^i_{\leq k_{high}}(J^*(t), \frac{t}{2}) \leq U^j_{\leq k_{high}}(J^*(t), \frac{t}{2}) + 2^{k_{high}+2} \leq U^j(J^*(t), \frac{t}{2}) + 2^{k_{high}+2}$.

Combining the above arguments yields:

$$\max_i\{U^i(J^*(t), \frac{t}{2})\} \leq \min_i\{U^i(J^*(t), \frac{t}{2})\} + 2^{k_{high}+2} \leq t + 2t = 3t \ .$$

Thus, algorithm IMD finishes to process all the jobs of $J^*(t)$ before time $3.5t$, even if it starts processing jobs only at time $\frac{t}{2}$. Therefore, $c^{OPT}(\hat{J}, t) \leq c^{IMD}(J^*(t), 3.5t)$. This proves the lemma. ■

Lemma 7.6 *For any time t, $c^{IMD}(J^*(t), t) \leq c^{IMD}(J, t)$.*

Proof: Note that not only $J^*(t) \subseteq J$, but $J^*(t)$ is class-wise prefix of J, i.e. the arrival time of any job of class k in $J^*(t)$ is at most the arrival time of any job of this class in $J \setminus J^*(t)$ (by our choice of a fair OPT schedule). Hence, the assignment of the jobs in $J^*(t)$ by IMD remains the same, when it runs on J. Therefore the job set that IMD assigns to each machine when running on J is a superset of the jobs it assigned when it ran only on $J^*(t)$. Note that algorithm IMD uses $SRPT$ on each machine in order to schedule the input jobs, moreover it is well known that $c^{SRPT}(J_1, t) \leq c^{SRPT}(J_2, t)$ for any t and $J_1 \subseteq J_2$ (see [14]), hence, for any time t, each machine completes at least the same number of jobs it completed on $J^*(t)$. The lemma follows. ■

Corollary 7.7 *For any time t, $c^{OPT}(\hat{J}, t) \leq c^{IMD}(J, 3.5t)$.*

Proof: Combining Lemmas 7.5 and 7.6 yields $c^{OPT}(\hat{J}, t) = c^{IMD}(J^*(t), 3.5t) \leq c^{IMD}(J, 3.5t)$. ■

Lemma 7.8 *$C^{IMD}(J) \leq 3.5 \cdot C^{OPT}(\hat{J})$.*

Proof: We compute the total completion time.

$$\begin{aligned} C^{IMD}(J) &= \int_t n - c^{IMD}(J, t)dt = \int_u [n - c^{IMD}(J, 3.5u)]3.5du \\ &\leq 3.5 \int_u n - c^{OPT}(\hat{J}, u)du = 3.5 \cdot C^{OPT}(\hat{J}) \end{aligned}$$

where the first and the last equalities are by Observation 7.4. The second equality is obtained by the variables change $3.5u = t$. The inequality is due to Corollary 7.7. ■

We turn to prove the main result of this section.

Theorem 7.9 *$C^{IMD}(J) \leq 7 \cdot C^{OPT}(J)$, i.e., algorithm IMD has a small constant approximation factor even when compare to the best (possibly migratory) offline algorithm, with respect to the total completion time.*

Proof: Combining Lemmas 7.3 and 7.8 yields: $C^{IMD}(J) \leq 3.5 \cdot C^{OPT}(\hat{J}) \leq 7 \cdot C^{OPT}(J)$. ■

Note that the preemptive algorithm IMD can be converted into a non-preemptive algorithm IMD' by applying some single machine 'preemptive to non-preemptive' conversion to each of the machines separately. Such a conversion algorithm was introduced in [14], which basically list-schedules the jobs according to their completion time in the preemptive schedule. This conversion results in losing only a constant factor of 2 in our approximation, resulting in a non-preemptive schedule generated using immediate dispatching, with a 14 approximation factor of the best possibly migratory offline algorithm with respect to the total completion time.