Minimizing Average Flow Time on Related Machines

Naveen Garg Amit Kumar

Dept. of Computer Science and Engg.
IIT Delhi, New Delhi – 110016
email : {naveen,amitk}@cse.iitd.ac.in

November 3, 2005

Abstract

We give the first on-line poly-logarithmic competitve algorithm for minimizing average flow time with preemption on related machines, i.e., when machines can have different speeds. This also yields the first poly-logarithmic polynomial time approximation algorithm for this problem. More specifically, we give an $O(\log^2 P \cdot \log S)$-competitive algorithm, where P is the ratio of the biggest and the smallest processing time of a job, and S is the ratio of the highest and the smallest speed of a machine. Our algorithm also has the nice property that it is non-migratory. The scheduling algorithm is based on the concept of making jobs wait for a long enough time before scheduling them on slow machines.

1 Introduction

We consider the problem of scheduling jobs that arrive over time in multiprocessor environments. This is a fundamental scheduling problem and has many applications, e.g., servicing requests in web servers. The goal of a scheduling algorithm is to process jobs on the machines so that some measure of performance is optimized. Perhaps the most natural measure is the average flow time of the jobs. Flow time of a job is defined as the difference of its completion time and its release time, i.e., the time it spends in the system.

This problem has received considerable attention in the recent past[2] [10], [1]. All of these works make the assumption that the machines are identical, i.e., they have the same speed. But it is very natural to expect that in a heterogenous processing environment different machines will have different processing power, and hence different speeds. In this paper, we consider the problem of scheduling jobs on machines with different speeds, which is also referred to as *related machines* in the scheduling literature. We allow for jobs to be preempted. Indeed, the problem turns out to be intractable if we do not allow preemption. Kellerer et. al. [9] showed that the problem of minimizing average flow time without preemption has no online algorithm with $o(n)$ competitive ratio even on a single machine. They also showed that it is hard to get a polynomial time $O(n^{1/2-\varepsilon})$-approximation algorithm for this problem. So preemption is a standard, and much needed, assumption when minimising flow time.

In the standard notation of Graham et. al. [7], we consider the problem $Q|r_j, pmtn|\sum_j F_j$. We give the first poly-logarithmic competitive algorithm for minimizing average flow time on related machines. More specifically, we give an $O(\log^2 P \cdot \log S)$-competitive algorithm, where P is the ratio of the biggest and the smallest processing time of a job, and S is the ratio of the highest and the smallest speed of a machine. This is also the first polynomial time poly-logarithmic approximation algorithm for this problem. Despite its similarity to the special case when machines are identical, this problem is much more difficult since we also have to worry about the processing times of jobs. Our algorithm is also non-migratory, i.e., it processes a job on only one machine. This is a desirable feature in many applications because moving jobs across machines may have many overheads.

Related Work The problem of minimizing average flow time (with preemption) on identical parallel machines has received much attention in the past few years. Leonardi and Raz [10] showed that the Shortest Remaining Processing Time (SRPT) algorithm has a competitive ratio of $O(\log(\min(\frac{n}{m}, P)))$, where n is the number of jobs, and m is the number of machines. A matching lower bound on the competitive ratio of any on-line (randomized) algorithm for this problem was also shown by the same authors. Awerbuch et. al. [2] gave a non-migratory version of SRPT with competitive ratio of $O(\log(\min(n, P)))$. Chekuri et. al.[5] gave a non-migratory algorithm with competitive ratio of $O(\log(\min(\frac{n}{m}, P)))$. One of the merits of their algorithm was a much simpler analysis of the competitive ratio. Instead of preferring jobs according to their remaining processing times, their algorithm divides jobs into classes when they arrive. A job goes to class k if its processing time is between 2^{k-1} and 2^k. The scheduling algorithm now prefers jobs of smaller class irrespective of the remaining processing time. We also use this notion of classes of jobs in our algorithm. Azar and Avrahami[1] gave a non-migratory algorithm with immediate dispatch, i.e., a job is sent to a machine as soon as it arrives. Their algorithm tries to balance the load assigned to each machine for each class of jobs. Their algorithm also has the competitive ratio of $O(\log(\min(\frac{n}{m}, P)))$. It is also interesting to note that these are also the best known results in the off-line setting of this problem.

Kalyanasundaram and Pruhs[8] introduced the resource augmentation model where the algorithm is allowed extra resources when compared with the optimal off-line algorithm. These extra resources can be either extra machines or extra speed. For minimizing average flow time on identical parallel machines, Phillips et. al. [11] showed that we can get an optimal algorithm if we are given twice the speed as compared to the optimal algorithm. In the case of single machine scheduling, Bechheti et. al.[4] showed that we can get $O(1/\varepsilon)$-competitive algorithms if we are give $(1+\varepsilon)$ times more resources. Bansal and Pruhs[3] extended this result to a variety of natural scheduling algorithms and to L_p norms of flow times of jobs as well. In case of identical parallel machines, Chekuri et. al.[6] gave simple scheduling algorithms which are $O(1/\varepsilon 3)$-competitive with $(1+\varepsilon)$ resource augmentation.

Our Techniques A natural algorithm to try here would be SRPT. We feel that it will be very difficult to analyze this algorithm in case of related machines. Further SRPT is migratory. Non-migratory versions of SRPT can be shown to have bad competitive ratios. To illustrate the ideas involved, consider the following example. There is one fast machine, and plenty of slow machines. Suppose many jobs arrive at the same time. If we distribute these jobs to all the available machines, then their processing times will be very high. So at each time step, we need to be selective about which machines we shall use for processing. Ideas based on distributing jobs in proportion to speeds of machines as used by Azar and Avrahami[1] can also be shown to have problems.

Our idea of selecting machines is the following. A job is assigned to a machine only if it has waited for time which is proportional to its processing time on this machine. The intuition should be clear – if a job is going to a slow machine, then it can afford to wait longer before it is sent to the machine. Hopefully in this waiting period, a faster machine might become free in which case we shall be able to process it in smaller time. We also use the notion of class of jobs as introduced by Chekuri et. al.[5] which allows machines to have a preference ordering over jobs. We feel that this idea of delaying jobs even if a machine is available is new.

As mentioned earlier, the first challenge is to bound the processing time of our algorithm. In fact a bulk of our paper is about this. The basic idea used is the following – if a job is sent to a very slow machine, then it must have waited long. But then most of this time, our algorithm would have kept the fast machines busy. Since we are keeping fast machines busy, the optimum also can not do much better. But converting this idea into a proof requires several technical steps.

The second step is of course to bound the flow time of the jobs. Previous works on bounding flow time of jobs in the case of identical machines achieve this in the following way – consider a time t. It can be shown that there is a time t' such that at t', our algorithm and the optimum algorithm are in a very similar state and all our machines are busy during (t', t). This would then imply that at time t also, our algorithm and the optimum algorithm are in a similar state. But we can not use this idea because we would never be able to keep all machines busy (some machines can be very slow). So we have to define a sequence of time steps like t' for each time and make clever use of geometric scaling to show that the flow time is bounded.

2 Problem Definition and Notations

We consider the on-line problem of minimizing total flow time for related machines. Each job j has a processing requirement of p_j and a release date of r_j. There are m machines, numbered from 1 to m. The machines can have different speeds, and the processing time of a job j on a machine is p_j divided by the speed of the machine. The *slowness* of machine is the reciprocal of its speed. It will be easier to deal with slowness, and so we shall use slowness instead of speed in the foregoing discussion. Let s_i denote the slowness of machine i. So the time taken by job j to process on machine i is $p_j \cdot s_i$. Assume that the machines have been numbered so that $s_1 \leq s_2 \leq \cdots \leq s_m$. We shall assume without loss of generality that processing time, release dates, and slowness are integers. We shall use the term *volume* of a set of jobs for denoting their processing time on a unit speed machine.

Let $\mathcal{A}$ be a scheduling algorithm. The completion time of a job j in $\mathcal{A}$ is denoted by $C_j^{\mathcal{A}}$. The flow time of j in $\mathcal{A}$ is defined as $F_j^{\mathcal{A}} = C_j^{\mathcal{A}} - r_j$. Our goal is to find an on-line scheduling algorithm which minimizes the total flow time of jobs. Let $\mathcal{O}$ denote the optimal off-line scheduling algorithm.

We now develop some notations. Let P denote the ratio of the largest to the smallest processing times of the jobs, and S be the ratio of the largest to the smallest slowness of the machines. For ease of notation, we assume that the smallest processing requirement of any job is 1, and the smallest slowness of a machine is 1. Let α and β be suitable chosen large enough constants. We divide the jobs and the machines into *classes*. A job j is said to be in class k if $p_j \in [\alpha^{k-1}, \alpha^k)$ and a machine i is said to be in class l if $s_i \in [\beta^{l-1}, \beta^l)$. Note that there are $O(\log P)$ classes for jobs and $O(\log S)$ classes for machines. Given a schedule $\mathcal{A}$, we say that a job j is *active* at time t in $\mathcal{A}$ if $r_j \leq t$ but j has not finished processing by time t in $\mathcal{A}$.

3 Scheduling Algorithm

We now describe the scheduling algorithm. The underlying idea of the algorithm is the following — if we send a job j to machine i, we make sure that it waits for at least $p_j \cdot s_i$ units of time (which is its processing time on machine i). Intuitively, the extra waiting time can be charged to its processing time. Of course, we still need to make sure that the processing time does not blow up in this process.

The algorithm maintains a central pool of jobs. When a new job gets released, it first goes to the central pool and waits there to get assigned to a machine. Let $W(t)$ denote the set of jobs in the central pool at time t. Our algorithm will assign each job to a unique machine — if a job j gets assigned to a machine i, then j will get processed by machine i only. Let $A_i(t)$ be the set of active jobs at time t which have been assigned to machine i. We shall maintain the invariant that $A_i(t)$ contains at most one job of each class. So $|A_i(t)| \leq \log P$.

We say that a job $j \in W(t)$ of class k is *mature* for a machine i of class l at time t, if it has waited for at least $\alpha^k \cdot \beta^l$ time in the central pool, i.e., $t - r_j \geq \alpha^k \cdot \beta^l$. For any time t, we define a total order $\prec$ on the jobs in $W(t)$ as follows – $j \prec j'$ if either (i) class$(j) < class(j')$, or (ii) class$(j) =$ class(j') and $r_j > r_{j'}$ (in case class$(j) =$ class(j') and $r_j = r_{j'}$ we can order them arbitrarily).

Now we describe the actual details of the algorithm. Initially, at time 0, $A_i(0)$ is empty for each machine. At each time t, the algorithm considers machines in the order $1, 2, \ldots, m$ (recall that the machines have been arranged in the ascending order of their slowness). Let us describe the algorithm when it considers machine i. Let $M_i(t)$ be the jobs in $W(t)$ which are mature for machine i at time t. Let $j \in M_i(t)$ be the smallest job according to the total order $\prec$. If class$(j) <$ class(j') for all jobs $j' \in A_i(t)$, then we assign j to machine i (i.e., we delete j from $W(t)$ and add it to $A_i(t)$).

Once the algorithm has considered all the machines at time t, each machine i processes the job of smallest class in $A_i(t)$ at time t. This completes the description of our algorithm. It is also easy to see that the machines need to perform the above steps for only a polynomial number of time steps t (i.e., when a job finishes or matures for a class of machines).

We remark that both the conditions in the definition of $\prec$ are necessary. The condition (i) is clear because it prefers smaller jobs. Condition (ii) makes sure that we make old jobs wait longer so that they can mature for slower machines. It is easy to construct examples where if we do not obey condition (ii) then slow machines will never get used and so we will incur very high flow time.

4 Analysis

We now show that the flow time incurred by our algorithm is within poly-logarithmic factor of that of the optimal algorithm. The outline of the proof is as follows. We first argue that the total processing time incurred by our algorithm is not too large. Once we have shown this, we can charge the waiting time of all the jobs in $A_i(t)$ for all machines i and time t to the total processing time. After this, we show that the total waiting time of the jobs in the central pool is also bounded by poly-logarithmic factor times the optimum's flow time.

Let $\mathcal{A}$ denote our algorithm. For a job j, define the *dispatch time* $d_j^{\mathcal{A}}$ of j in $\mathcal{A}$ as the time at which it is assigned to a machine. For a job j and class l of machines, let $t_M(j,l)$ denote the time at which j matures for machines of class l, i.e., $t_M(j,l) = r_j + \alpha^k \beta^l$, where k is the class of job j. Let $F^{\mathcal{A}}$ denote the total flow time of our algorithm. For a job j let $P_j^{\mathcal{A}}$ denote the time it takes to process job j in $\mathcal{A}$ (i.e., the processing time of j in $\mathcal{A}$). Similarly, for a set of jobs

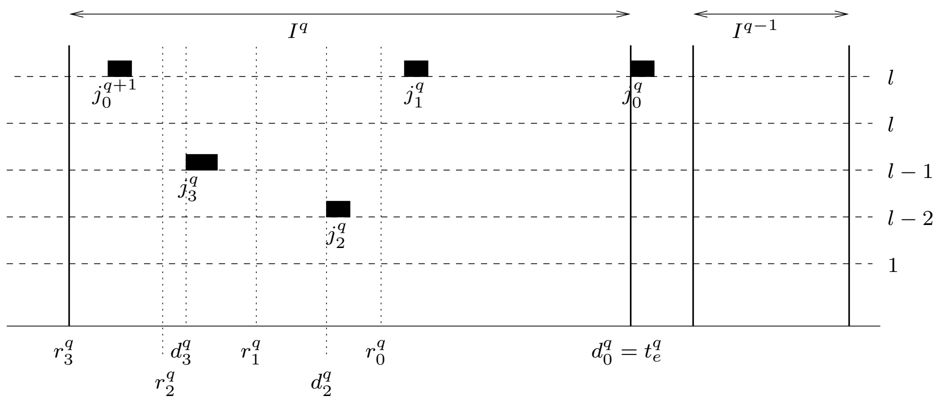

Figure 1: An illustration of the notation used

J define $P_J^{\mathcal{A}}$ as the total processing time incurred by $\mathcal{A}$ on these jobs. Let $P^{\mathcal{A}}$ denote the sum $\sum_j P_j^{\mathcal{A}}$, i.e., the total processing time incurred by $\mathcal{A}$. Define $F^{\mathcal{O}}$, $P_j^{\mathcal{O}}$, $P_J^{\mathcal{O}}$ and $P^{\mathcal{O}}$ similarly. Let m_l denote the number of machines of class l.

4.1 Bounding the Processing Time

We shall now compare $P^{\mathcal{A}}$ with $F^{\mathcal{O}}$. For each value of class k of jobs and class l of machines, let $J(k,l)$ be the jobs of class k which get processed by $\mathcal{A}$ on machines of class l. For each value of k and l, we shall bound the processing time incurred by $\mathcal{A}$ on $J(k,l)$. So fix a class k of jobs and class l of machines.

The idea of the proof is as follows. We shall divide the time line into intervals $I^1 = (t_b^1, t_e^1), I^2 = (t_b^2, t_e^2), \ldots$ so that each interval I^q satisfies the following property – $\mathcal{A}$ is *almost* busy in I^q processing jobs of class at most k on machines of class $l-1$ or less. Further these jobs have release time at least t_b^q. We shall denote these jobs by H^q. Now, if $\mathcal{O}$ schedules jobs in $J(k,l)$ on machines of class l or more, we have nothing to prove since $\mathcal{O}$ would incur at least as much processing time as we do for these jobs. If $\mathcal{O}$ schedules some jobs in $J(k,l)$ on machines of class $l-1$ or less during the interval I^q, then one of these two cases must happen – (i) some jobs in H^q need to be processed on machines of class l or more, or (ii) some jobs in H^q get processed after time t_e^q. We shall show that both of these cases are good for us and we can charge the processing times of jobs in $J(k,l)$ to the flow time of jobs in H^q in $\mathcal{O}$.

Let us formalize these ideas (see Figure 1). The starting points of the intervals $I^1, I^2, \ldots$ will be in decreasing order, i.e., $t_b^1 > t_b^2 > \cdots$. Suppose we have defined intervals $I^1, \ldots, I^{q-1}$ so far. Let $J^q(k,l)$ denote the set of jobs in $J(k,l)$ which get released before interval I^{q-1} begins, i.e., before t_b^{q-1} ($J^1(k,l)$ is defined as $J(k,l)$).

Now we define I^q. Let $j_0^q \in J^q(k,l)$ be the job with the highest dispatch time. Let r_0^q denote the release time of j_0^q. Let k_0^q denote k, the class of job j_0^q. Let d_0^q denote the dispatch time of j_0^q. The right end-point t_e^q of I^q is defined as d_0^q.

Consider the jobs in $J(k,l)$ which get dispatched during (r_0^q, d_0^q). Let j_1^q be such a job with the earliest release date. Define r_1^q, k_1^q, d_1^q similarly.

Let $H_0^q(l')$, $l' < l$, be the set of jobs of class at most k_0^q which are dispatched to a machine of class l' during the time interval $(t_M(j_0^q, l'), d_0^q)$. Note that the phrase "class at most k_0^q" in the previous sentence is redundant because we cannot dispatch a job of class greater than k_0^q during $(t_M(j_0^q, l'), d_0^q)$ on machines of class l'. Let H_0^q denote $\cup_{l'=1}^{l-1} H_0^q(l')$. Define $H_1^q(l'), H_1^q$ similarly.

If all jobs in $H_1^q \cup H_0^q$ get released after r_1^q, we stop the process here. Otherwise find a job in $H_1^q \cup H_0^q$ which has the earliest release date, let j_2^q denote this job. As before define r_2^q as the release date of j_2^q, and k_2^q as the class of j_2^q. Again define $H_2^q(l')$ as the set of jobs (of class at most k_2^q) which get dispatched on machines of class l' during $(t_M(j_2^q, l'), d_2^q)$. Define

H_2^q analogously. So now assume we have defined $j_0^q, j_1^q, \ldots, j_i^q$ and $H_0^q, H_1^q, \ldots, H_i^q$, $i \geq 2$. If all jobs in H_i^q are released after r_i^q, then we stop the process. Otherwise, define j_{i+1}^q as the job in H_i^q with the earliest release date. Define H_{i+1}^q in a similar manner (see Figure 1). We remark that the first two steps of this process differ from the subsequent steps. This we will see later is requierd to ensure that the intervals do not overlap too much. The following simple observation shows that this process will stop.

Claim 4.1 *For $i \geq 2$, $k_i^q < k_{i-1}^q$.*

Proof. Consider a job $j \in H_{i-1}^q(l')$. If class(j) = class(j_{i-1}^q), then $\mathcal{A}$ prefers j over j_{i-1}^q. So, release time of j must be at least r_{i-1}^q. Thus, class of j_i^q must be less than that of j_{i-1}^q. ■

Suppose this process stops after u_q steps. Define the beginning of I^q, i.e., t_b^q as $r_{u_q}^q$.

This completes our description of I^q. Let H^q denote $\cup_i H_i^q$. We are now ready to show that interval I^q is sufficiently long, and that for a large fraction of this time, all machines of class less than l are processing jobs of class less than k which are released in this interval itself. This would be valuable in later arguing that $\mathcal{O}$ could not have scheduked jobs of $J(k,l)$ on the lower class machines and thereby incurred small processing time.

Lemma 4.2 *Length of the interval I^q is at least $\alpha^k \beta^l$.*

Proof. Indeed, job j_0^q must wait for at least $\alpha^k \beta^l$ amount of time before being dispatched to a machine of class l. So $t_e^q - t_s^q \geq d_0^q - r_0^q \geq \alpha^k \beta^l$. ■

Lemma 4.3 *H^q consists of jobs of class at most k and all jobs in H^q are released after t_b^q.*

Proof. The first statement is clear from the definition of H^q. Let us look at H_i^q. As argued in proof of Claim 4.1 all jobs of class k_i^q in H_i^q are released after r_i^q. If all jobs of class less than k_i^q in H_i^q are released after r_i^q, then we are done. Otherwise, all such jobs are released after r_{i+1}^q (after r_2^q if $i = 0$). This completes the proof of the lemma. ■

Lemma 4.4 *A machine of class $l' < l$ processes jobs of H^q for at least $|I_q| - 6 \cdot \alpha^k \beta^{l'}$ amount of time during I_q.*

Proof. Let us fix a machine i of class $l' < l$. Let us consider the interval (r_p^q, d_p^q). Job j_p^q matures for i at time $t_M(j_p^q, l')$. So machine i must be busy during $(t_M(j_p^q, l'), d_p^q)$, otherwise we could have dispatched j_p^q earlier. We now want to argue that machine i is mainly busy during this interval processing jobs from H^q.

Let j be a job which is processed by i during $(t_M(j_p^q, l'), d_p^q)$. We have already noted that j can be of class at most k_p^q. If j gets dispatched after $t_M(j_p^q, l')$, then by definition it belongs to H^q. If j gets dispatched before $t_M(j_p^q, l')$, it must belong to $A_i(t')$. But $A_i(t')$ can contain at most one job of each class. So the total processing time taken by jobs of $A_i(t')$ during (t', d_p) is at most $\beta^{l'} \cdot (\alpha + \alpha^2 + \cdots + \alpha^{k_p^q}) \leq 3/2 \cdot \alpha^{k_p^q} \beta^{l'}$.

So during (r_p^q, d_p^q), i processes jobs from H^q except perhaps for a period of length $(t_M(j_p^q, l') - r_p^q) + 3/2 \cdot \alpha^{k_p^q} \beta^{l'} = 5/2 \cdot \alpha^{k_p^q} \beta^{l'}$. Since $\cup_p (r_p^q, d_p)$ covers I_q, the amount of time i does not process jobs from H^q is at most $5/2 \cdot \beta^{l'} (\alpha^k + \alpha^k + \cdots + 1)$, which proves the lemma. ■

We would like to charge the processing time for jobs in $J(k,l)$ in our solution to the flow time of jobs in H^q in $\mathcal{O}$. But to do this we require that the sets H^q be disjoint. We next prove that this is almost true.

Lemma 4.5 *For any q, I^q and I^{q+2} are disjoint. Hence H^q and H^{q+2} are also disjoint.*

Proof. Recall that $J^{q+1}(k,l)$ is the set of jobs in $J(k,l)$ which get released before t_b^q. However some of these jobs may get dispatched after t_b^q and hence I^q and I^{q+1} can intersect.

Consider some job $j \in J^{q+1}(k,l)$ is dispatched during I^q. Now, observe that j_0^{q+1} is released before t_b^q (by definition of $J^{q+1}(k,l)$) and dispatched after j. So, j gets dispatched in (r_0^q, d_0^q).

This means that release date of j_1^{q+1} must be before release date of j. But $t_b^{q+1} \leq r_1^{q+1}$ and so, j is released after t_b^{q+1}. But then $j \notin J^{q+2}(k,l)$. So all jobs in $J^{q+2}(k,l)$ get dispatched before I^q begins, which implies that I^q and I^{q+2} are disjoint. ■

Consider an interval I^q. Let $D^q(k,l)$ denote the jobs in $J(k,l)$ which get released after t_b^q but dispatched before t_e^q. It is easy to see that $D^q(k,l)$ is a superset of $J^q(k,l) - J^{q+1}(k,l)$. So $\cup_q D^q(k,l)$ covers all of $J(k,l)$.

Now we would like to charge the processing time of jobs in $D^q(k,l)$ to the flow time incurred by $\mathcal{O}$ on the jobs in H^q. Before we do this, let us first show that $\mathcal{O}$ incurs significant amount of flow time processing the jobs in H^q. This is proved in the technical theorem below whose proof we defer to the appendix.

Theorem 4.6 *Consider a time interval $I = (t_b, t_e)$ of length T. Suppose there exists a set of jobs J_I such that every job $j \in J_I$ is of class at most k and is released after t_b. Further $\mathcal{A}$ dispatches all the jobs in J_I during I and only on machines of class less than l. Further, each machine i of class $l' < l$ satisfies the following condition : machine i processes jobs from J_I for at least $T - 6 \cdot \alpha^k \cdot \beta^{l'}$ amount of time. Assuming $T \geq \alpha^k \cdot \beta^l$, the flow time $F_{J_I}^{\mathcal{O}}$ incurred by $\mathcal{O}$ on the jobs in J_I is at least $\Omega\left(P_{J_I}^{\mathcal{A}}\right)$.*

Substituting $t_b = t_b^q, t_e = t_e^q, I = I^q, T = |I^q|, J_I = H^q$ in the statement of Theorem 4.6 and using Lemmas 4.2, 4.3, 4.4, we get

$$P_{H^q}^{\mathcal{A}} \leq O(F_{H^q}^{\mathcal{O}}). \tag{1}$$

We are now ready to prove the main theorem of this section.

Theorem 4.7 *The processing time incurred by $\mathcal{A}$ on $D^q(k,l)$, namely $P_{D^q(k,l)}^{\mathcal{A}}$, is $O(F_{H^q}^{\mathcal{O}} + F_{D^q(k,l)}^{\mathcal{O}})$.*

Proof. Let V denote the volume of $D^q(k,l)$. By Lemma 4.4, machines of class l' do not process jobs from H^q for at most $6 \cdot \alpha^k \beta^{l'}$ units of time during I^q. This period translates to a job-volume of at most $6\beta \cdot \alpha^k$. If V is sufficiently small then it can be charged to the processing time (or equivalently the flow time in $\mathcal{O}$) of the jobs in H^q.

Lemma 4.8 *If $V \leq c \cdot \beta \cdot \alpha^k (m_0 + \ldots + m_{l-1})$ where c is a constant, then $P_{D^q(k,l)}^{\mathcal{A}}$ is $O(F_{H^q}^{\mathcal{O}})$.*

Proof. The processing time incurred by $\mathcal{A}$ on $D^q(k,l)$ is at most $\beta^l V = O(\alpha^k \beta^{l+1} \cdot (m_0 + \ldots + m_{l-1}))$. Now, Lemmas 4.2 and 4.4 imply that machines of class $l' < l$ process jobs from H^q for at least $\alpha^k \beta^l / 2$ amount of time. So $P_{H^q}^{\mathcal{A}}$ is at least $\alpha^k \beta^l (m_0 + \ldots + m_{l-1})/2$. Using equation (1), we get the result. ■

So from now on we shall assume that $V \geq c \cdot \beta \cdot \alpha^{k+1} (m_0 + \ldots + m_{l-1})$ for a sufficiently large constant c. We deal with easy cases first :

Lemma 4.9 *In each of the following cases $P_{D^q(k,l)}^{\mathcal{A}}$ is $O(F_{D^q(k,l)}^{\mathcal{O}} + F_{H^q}^{\mathcal{O}})$:*

(i) At least $V/2$ volume of $D^q(k,l)$ is processed on machines of class at least l by $\mathcal{O}$.

(ii) At least $V/4$ volume of $D^q(k,l)$ is finished by $\mathcal{O}$ after time $t_e^q - \alpha^k \beta^l / 2$.

(iii) At least $V/8$ volume of H^q is processed by $\mathcal{O}$ on machines of class l or more.

Proof. Note that $P_{D^q(k,l)}^{\mathcal{A}}$ is at most $V\beta^l$. If (i) happens, $\mathcal{O}$ pays at least $V/2 \cdot \beta^{l-1}$ amount of processing time for $D^q(k,l)$ and so we are done. Suppose case (ii) occurs. All jobs in $D^q(k,l)$ get dispatched by time t_e^q. So they must get released before $t_e^q - \alpha^k \beta^l$. So at least 1/4 volume of these jobs wait for at least $\alpha^k \beta^l / 2$ amount of time in $\mathcal{O}$. This means that $F_{D^q(k,l)}^{\mathcal{O}}$ is at least $V/8 \cdot \beta^l$, because each job has size at most α^k. This again implies the lemma. Case (iii) is similar to case (i). ■

So we can assume that none of the cases in Lemma 4.9 occur. Now, consider a time t between $t_e^q - \alpha^k\beta^l/2$ and t_e^q. Let us look at machines of class 1 to $l-1$. $\mathcal{O}$ finishes at least $V/4$ volume of $D^q(k,l)$ on these machines before time t. Further, at most $V/8$ volume of H^q goes out from these machines to machines of higher class. The volume that corresponds to time slots in which these machines do not process jobs of H^q in $\mathcal{A}$ is at most $V/16$ (for a sufficiently large constant c). So, at least $V/16$ amount of volume of H^q must be waiting in $\mathcal{O}$ at time t. So the total flow time incurred by $\mathcal{O}$ on H^q is at least $V/(16\alpha^k)\cdot\alpha^k\beta^l/2$ which again is $\Omega(V\beta^l)$. This proves the theorem. ■

Combining the above theorem with Lemma 4.5, we get

Corollary 4.10 *$P^{\mathcal{A}}_{J(k,l)}$ is $O(F^{\mathcal{O}})$, and so $P^{\mathcal{A}}$ is $O(\log S \log P \cdot F^{\mathcal{O}})$.*

4.2 Bounding the flow time

We now show that the average flow time incurred by our algorithm is within poly-logarithmic factor of that incurred by the optimal solution. We shall say that a job j is *waiting* at time t in $\mathcal{A}$ if it is in the central pool at time t in $\mathcal{A}$ and has been released before time t.

Let $V^{\mathcal{A}}_{\le k}(t)$ denote the volume of jobs of class at most k which are waiting at time t in $\mathcal{A}$. Define $V^{\mathcal{O}}_{\le k}$ as the remaining volume of jobs of class at most k which are active at time t in $\mathcal{O}$. Note the slight difference in the definitions for $\mathcal{A}$ and $\mathcal{O}$ — in case of $\mathcal{A}$, we are counting only those jobs which are in the central pool, while in $\mathcal{O}$ we are counting all active jobs. Our goal is to show that for all values of k, $\sum_t V^{\mathcal{A}}_{\le k}(t)$ is bounded from above by $\sum_t V^{\mathcal{O}}_{\le k}(t)$ plus small additive factors, i.e., $O(\alpha^k \cdot (P^{\mathcal{O}} + P^{\mathcal{A}}))$. Once we have this, the desired result will follow from standard calculations.

Let us develop some more notation. $x^{\mathcal{A}}(t)$ will denote the number of machines busy at time t in $\mathcal{A}$. Define $x^{\mathcal{O}}(t)$ similarly. $x^{\mathcal{O}}_l(t)$ denotes the number of machines of class l busy at time t in $\mathcal{O}$. $P^{\mathcal{O}}_l(t_1,t_2)$ will denote the total amount of processing time incurred by machines of class l during the time period (t_1,t_2) by $\mathcal{O}$. Define $P^{\mathcal{A}}_l(t_1,t_2)$ similarly. Recall that m_l denotes the number of machines of class l. Define $m_{\le l}$ as the number of machines of class at most l. Similarly define $m_{<l}$. Let $m_{(l_1,l_2)}$ denote the number of machines of class between l_1 and l_2 (inclusive of l_1 and l_2).

Given a class k of jobs and time t, let $J_k(t)$ be the set of jobs of class at most k which are waiting at time t in $\mathcal{A}$. Let $j_k(t) \in J_k(t)$ be the job with the earliest release date. Let $r_k(t)$ denote the release date of job $j_k(t)$, and $c_k(t)$ denote the class of $j_k(t)$. Let $l_k(t)$ denote the largest l such that $j_k(t)$ has matured for machines of class l at time t. In other words, $l_k(t)$ is the highest value of l such that $t_M(j,l) \le t$. Observe that all machines of class $l_k(t)$ or less must be busy processing jobs of class at most $c_k(t)$ at time t, otherwise our algorithm should have dispatched j by time t. Our proof will use the following lemma.

Lemma 4.11 *For any class k of jobs and time t,*

$$V^{\mathcal{A}}_{\le k}(t) - V^{\mathcal{O}}_{\le k}(t) \le 2\cdot\beta\cdot\alpha^{c_k(t)}\cdot m_{\le(l_k(t)-1)} + V^{\mathcal{A}}_{\le(c_k(t)-1)}(r_k(t)) - V^{\mathcal{O}}_{\le(c_k(t)-1)}(r_k(t)) + \sum_{l \ge l_k(t)} \frac{P^{\mathcal{O}}_l(r_k(t),t)}{\beta^{l-1}}.$$

Proof. Let $V^{\mathcal{A}}$ denote the volume of jobs of class at most k which are processed by $\mathcal{A}$ on machines of class $l_k(t)-1$ or less during $(r_k(t),t)$. Define $U^{\mathcal{A}}$ as the volume of jobs of class at most k which are processed by $\mathcal{A}$ on machines of class $l_k(t)$ or more during $(r_k(t),t)$. Define $V^{\mathcal{O}}$ and $U^{\mathcal{O}}$ similarly. Clearly, $V^{\mathcal{A}}_{\le k}(t) - V^{\mathcal{O}}_{\le k}(t) = V^{\mathcal{A}}_{\le k}(r_k(t)) - V^{\mathcal{O}}_{\le k}(r_k(t)) + (V^{\mathcal{O}} - V^{\mathcal{A}}) + (U^{\mathcal{O}} - U^{\mathcal{A}})$.

Let us look at $V^{\mathcal{O}} - V^{\mathcal{A}}$ first. Any machine of class $l' \le l_k(t)-1$ is busy in $\mathcal{A}$ during $(t_M(j_k(t),l'),t)$ processing jobs of class at most $c_k(t)$. The amount of volume $\mathcal{O}$ can process on such machines during $(r_k(t), t_M(j_k(t),l'))$ is at most $\frac{m_{l'}\cdot\alpha^{c_k(t)}\beta^{l'}}{\beta^{l'-1}}$, which is at most $m_{l'}\cdot\beta\cdot\alpha^{c_k(t)}$. So we get $V^{\mathcal{O}} - V^{\mathcal{A}} \le m_{\le(l_k(t)-1)}\cdot\beta\cdot\alpha^{c_k(t)}$.

Let us now consider $V^{\mathcal{A}}_{\leq k}(r_k(t)) - V^{\mathcal{O}}_{\leq k}(r_k(t))$. Let us consider jobs of class $c_k(t)$ or more which are waiting at time $r_k(t)$ in $\mathcal{A}$ (so they were released before $r_k(t)$). By our definition of $j_k(t)$, all such jobs must be processed by time t in $\mathcal{A}$. If $l' \leq l_k(t)-1$, then such jobs can be done on machines of class l' only during $(t_M(j_k(t), l'), t)$. So again we can show that the total volume of such jobs is at most $m_{\leq(l_k(t)-1)} \cdot \beta \cdot \alpha^{c_k(t)} + U^{\mathcal{A}}$. Thus we get $V^{\mathcal{A}}_{\leq k}(r_k(t)) - V^{\mathcal{O}}_{\leq k}(r_k(t)) \leq m_{\leq(l_k(t)-1)} \cdot \beta \cdot \alpha^{c_k(t)} + U^{\mathcal{A}} + V^{\mathcal{A}}_{(\leq c_k(t)-1)}(r_k(t)) - V^{\mathcal{O}}_{\leq(c_k(t)-1)}(r_k(t))$, because $V^{\mathcal{O}}_{\leq(c_k(t)-1)}(r_k(t)) \leq V^{\mathcal{O}}_{\leq k}(r_k(t))$.

Finally note that $U^{\mathcal{O}}$ is at most $\sum_{l \geq l_k(t)} \frac{P^{\mathcal{O}}_l(r_k(t),t)}{\beta^{l-1}}$. Combining everything, we get the result. ■

The rest of the proof is really about unraveling the expression in the lemma above. To illustrate the ideas involved, let us try to prove the special case for jobs of class 1 only. The lemma above implies that $V^{\mathcal{A}}_{\leq 1}(t) - V^{\mathcal{O}}_{\leq 1}(t) \leq 2 \cdot \beta \cdot m_{\leq(l_1(t)-1)} + \sum_{l \geq l_1(t)} \frac{P^{\mathcal{O}}_l(r_1(t),t)}{\beta^{l-1}}$. We are really interested in $\sum_t (V^{\mathcal{A}}_{\leq 1}(t) - V^{\mathcal{O}}_{\leq 1}(t))$. Now, the sum $\sum_t m_{\leq(l_1(t)-1)}$ is not a problem because we know that at time t all machines of class $l_1(t)$ or less must be busy in $\mathcal{A}$. So, $x^{\mathcal{A}}(t) \geq m_{\leq(l_1(t)-1)}$. So $\sum_t m_{\leq(l_1(t)-1)}$ is at most $\sum_t x^{\mathcal{A}}(t)$, which is the total processing time of $\mathcal{A}$. It is little tricky to bound the second term. We shall write $\sum_{l \geq l_1(t)} \frac{P^{\mathcal{O}}_l(r_1(t),t)}{\beta^{l-1}}$ as $\sum_{l \geq l_1(t)} \sum_{t'=r_1(t)}^{t} \frac{x^{\mathcal{O}}_l(t')}{\beta^{l-1}}$. We can think of this as saying that at time time t', $r_1(t) \leq t' \leq t$, we are charging $1/\beta^{l-1}$ amount to each machine of class l which is busy at time t' in $\mathcal{O}$. Note that here l is at least $l_1(t)$.

Now we consider $\sum_t \sum_{l \geq l_1(t)} \sum_{t'=r_1(t)}^{t} \frac{x^{\mathcal{O}}_l(t')}{\beta^{l-1}}$. For a fixed time t' and a machine i of class l which is busy at time t' in $\mathcal{O}$, let us see for how many times t we charge to i. We charge to i at time t' if t' lies in the interval $(r_1(t), t)$, and $l \geq l_1(t)$. Suppose this happens. We claim that $t - t'$ has to be at most $\alpha\beta^{l+1}$. Indeed otherwise $t - r_1(t) \geq \alpha\beta^{l+1}$ and so $j_1(t)$ has matured for machines of class $l+1$ as well. But then $l < l_1(t)$. So the total amount of charge machine i gets at time t' is at most $\alpha\beta^{l+1} \cdot 1/\beta^{l-1} = O(1)$. Thus, the total sum turns out to be at most a constant times the total processing time of $\mathcal{O}$.

Let us now try to prove this for the general case. We build some notation first. Fix a time t and class k. We shall define a sequence of times $t^0, t^1, \cdots$ and a sequence of jobs $j^0(t), j^1(t), \ldots$ associated with this sequence of times. $c^i(t)$ shall denote the class of the job $j^i(t)$. Let us see how we define this sequence. First of all $t^0 = t$, and $j^0(t)$ is the job $j_k(t)$ (as defined above). Recall the definition of $j_k(t)$ – it is the job with the earliest release date among all jobs of class at most k which are waiting at time t in $\mathcal{A}$. Note that $c^0(t)$, the class of this job, can be less than k. Now suppose we have defined $t^0, \ldots, t^i$ and $j^0(t), \ldots, j^i(t)$, $i \geq 0$. t^{i+1} is the release date of $j^i(t)$. $j^{i+1}(t)$ is defined as the job $j_{c^i(t)-1}(t^{i+1})$, i.e., the job with the earliest release date among all jobs of class less than $c^i(t)$ waiting at time t^{i+1} in $\mathcal{A}$. We shall also define a sequence of classes of machines $l^0(t), l^1(t), \ldots$ in the following manner – $l^i(t)$ is the highest class l of machines such that job $j^i(t)$ has matured for machines of class l at time t^i. Figure 2 illustrates these definitions. The vertical line at time t^i denotes $l^i(t)$ – the height of this line is proportional to $l^i(t)$.

We note the following simple fact.

Claim 4.12 $\beta^{l^i(t)} \cdot \alpha^{c^i(t)} \leq t^i - t^{i+1} \leq \beta^{l^i(t)+1} \cdot \alpha^{c^i(t)}$.

Proof. Indeed t^{i+1} is the release date of $j^i(t)$ and $j^i(t)$ matures for machines of class $l^i(t)$ at time t^i, but not for machines of class $l^{i+1}(t)$ at time t^i. ■

The statement in Lemma 4.11 can be unrolled iteratively to give the following inequality :

$$\begin{aligned} V^{\mathcal{A}}_{\leq k}(t) - V^{\mathcal{O}}_{\leq k}(t) \quad \leq \quad & 2 \cdot \beta \cdot \left(\alpha^{c^0(t)} \cdot m_{<l^0(t)} + \alpha^{c^1(t)} \cdot m_{<l^1(t)} + \cdots\right) + \\ & \left(\sum_{l \geq l^0(t)} \frac{P^{\mathcal{O}}_l(t^1, t^0)}{\beta^{l-1}} + \sum_{l \geq l^1(t)} \frac{P^{\mathcal{O}}_l(t^2, t^1)}{\beta^{l-1}} + \cdots\right) \end{aligned} \tag{2}$$

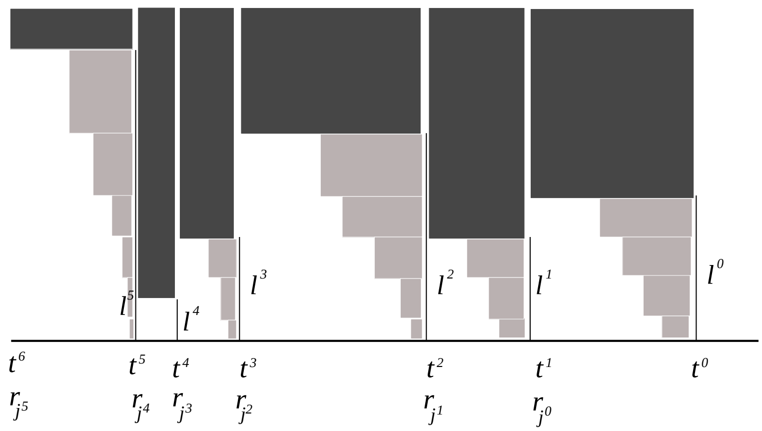

Figure 2: Illustrating the definitions of $t^0, \ldots$ and $c^0(t), \ldots$.

Let us try to see what the inequality means. First look at the term $\alpha^{c^i(t)} \cdot m_{<l^i(t)}$. Consider a machine of class $l < l^i(t)$. It is busy in $\mathcal{A}$ during $(t_M(j^i(t), l), t^i)$. Now $t^i - t_M(j^i(t), l) = (t^i - t^{i+1}) - (t_M(j^i(t), l) - t^{i+1}) \geq \beta^{l^i(t)} \cdot \alpha^{c^i(t)} - \alpha^{c^i(t)} \cdot \beta^l \geq \alpha^{c^i(t)} \cdot \beta^l$, as $l < l^i(t)$. Hence machine l is also busy in $\mathcal{A}$ during $(t^i - \alpha^{c^i(t)} \cdot \beta^l, t^i)$, So the term $\alpha^{c^i(t)} \cdot m_{<l^i(t)}$ is essentially saying that we charge $1/\beta^l$ amount to each machine of class $l < l^i(t)$ during each time in $(t^i - \alpha^{c^i(t)} \cdot \beta^l, t^i)$. These intervals are shown in figure 2 using light shade. So a machine i of class l which is busy in $\mathcal{A}$ during these lightly shaded regions is charged $1/\beta^l$ units.

Let us now look at the term $\sum_{l \geq l^i(t)} \frac{P_l^{\mathcal{O}}(t^{i+1}, t^i)}{\beta^{l-1}}$. This means the following – consider a machine of class $l \geq l^i(t)$ which is busy in $\mathcal{O}$ at some time t' during (t^{i+1}, t^i) – then we charge it $1/\beta^{l-1}$ units. Figure 2 illustrates this fact – we charge $1/\beta^{l-1}$ to all machines of class $l \geq l^i(t)$ which are busy in $\mathcal{O}$ during the darkly shaded regions.

Let us see how we can simplify the picture. We say that the index i is *suffix maximum* if $l^i(t) > l^{i-1}(t), \ldots, l^0(t)$ ($i = 0$ is always suffix maximum). In Figure 2, t^0, t^2 and t^5 are suffix maximum. Let the indices which are suffix maximum be $i_0 = 0 < i_1 < i_2 < \ldots$. The following lemma says that we can consider only suffix maximum indices in (2). We defer its proof to the appendix.

Lemma 4.13

$$V_{\leq k}^{\mathcal{A}}(t) - V_{\leq k}^{\mathcal{O}}(t) \leq 4\beta^2 \cdot \sum_{i_u} \alpha^{c^{i_u}(t)} \cdot m_{(l^{i_{u-1}}(t), l^{i_u}(t)-1)} + \sum_{i_u} \sum_{l \geq l^{i_u}(t)} \frac{P_l^{\mathcal{O}}(t^{i_{u+1}}, t^{i_u})}{\beta^{l-1}},$$

where i_u varies over the suffix maximum indices (define $l^{i_{-1}}(t)$ as 1). Recall that $m_{(l_1, l_2)}$ denotes the number of machines of class between l_1 and l_2.

Figure 3 shows what Lemma 4.13 is saying. In the lightly shaded region, if there is a machine of class l which is busy at some time t' in $\mathcal{A}$, we charge it $1/\beta^l$ units at time t'. In the darkly shaded region if there is a machine i of class l which is busy at time t' in $\mathcal{O}$ we charge it $1/\beta^{l-1}$ units.

Now we try to bound the charges on the darkly shaded region for all time t. Let us fix a machine h of class l. Suppose h is busy in $\mathcal{O}$ at time t'. We are charging $1/\beta^{l-1}$ amount to h at time t' if the following condition holds : for all i such that $t^i \geq t'$, $l^i(t)$ is at most l. Now we ask, if we fix t', for how many values of t do we charge h at time t' ? The following claim shows that this can not be too large.

Claim 4.14 *Given a machine h of class l, we can charge it for the darkly shaded region at time t' for at most $2 \cdot \beta^{l+1} \cdot \alpha^k$ values of t.*

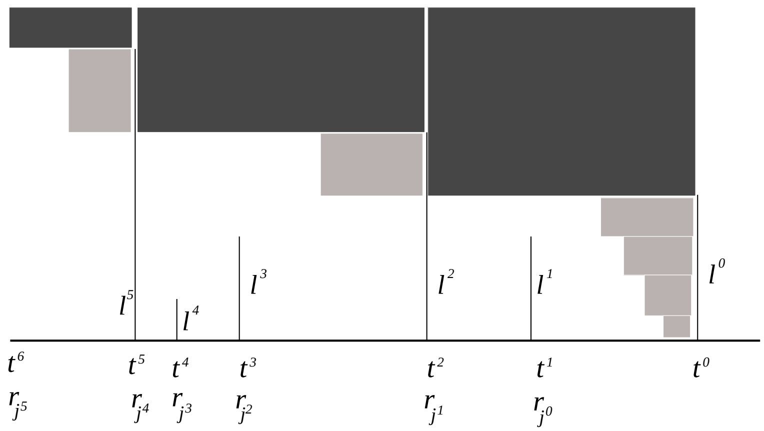

Figure 3: Illustrating Lemma 4.13

Proof. Suppose we charge h at time t' for some value of t. Fix this t. Clearly $t \geq t'$. Let i be the largest index such that $t' \leq t^i$. So $t - t' \leq t - t^{i+1}$. Now consider any $i' \leq i$. Lemma 4.12 implies that $t^{i'} - t^{i'+1} \leq \beta^{l^{i'}(t)+1} \cdot \alpha^{\alpha^{i'}(t)} \leq \beta^{l+1} \cdot \alpha^{c^{i'}(t)}$. Since $c^i(t)$ decrease as i increases, $\sum_{i'=0}^{i}(t^{i'} - t^{i'+1}) \leq 2 \cdot \beta^{l+1} \cdot \alpha^k$. This implies the claim. ∎

So the total amount of charge to machine h at time t' is at most $2 \cdot \beta^2 \cdot \alpha^k$. Thus we get the following fact :

$$\sum_t \left(\sum_{i_u} \sum_{l \geq l^{i_u}(t)} \frac{P_l^{\mathcal{O}}(t^{i_u+1}, t^{i_u})}{\beta^{l-1}} \right) \leq 2 \cdot \beta^2 \cdot \alpha^k \cdot P^{\mathcal{O}} \tag{3}$$

Now we look at the charges on the lightly shaded region. Let h be a machine of class l which is busy in $\mathcal{A}$ at time t'. As argued earlier, we charge $1/\beta^l$ units to h at time t' if the following condition holds – there exists a suffix maximum index i_u such that t' lies in the interval $(t^{i_u} - \alpha^{c^{i_u}(t)}\beta^l, t^{i_u})$. Further for all suffix maximum indices $i' < i_u$, it must be the case that $l^{i'}(t) < l$. Now we want to know for how many values of t do we charge h at time t'.

Claim 4.15 *Given a machine h of class l, we can charge it for the lightly shaded region at time t' for at most $3 \cdot \beta^{l+1} \cdot \alpha^k$ values of t.*

Proof. Fix a time t such that while accounting for $V^{\mathcal{A}}_{\leq k}(t)$ we charge h at time t'. So there is a maximum index i_u such that $t^{i_u} - t' \leq \alpha^{c^{i_u}(t)}\beta^l$. Further, if i is an index less than i_u, then $l^i(t)$ must be less than l. We can argue as in the proof of Claim 4.14 that $t - t^{i_u}$ can be at most $2 \cdot \beta^{l+1} \cdot \alpha^k$. So $t - t' \leq 3 \cdot \beta^{l+1} \cdot \alpha^k$. ∎

So we get

$$\sum_t \left(\sum_{i_u} \alpha^{c^{i_u}(t)} \cdot m_{(l^{i_u-1}(t), l^{i_u}(t)-1)} \right) \leq 3 \cdot \beta \cdot \alpha^k \cdot P^{\mathcal{A}} \tag{4}$$

Putting everything together, we see that Lemma 4.13, and equations (3) and (4) imply that there is a constant c such that

$$\sum_t V^{\mathcal{A}}_{\leq k}(t) \leq \sum_t V^{\mathcal{O}}_{\leq k}(t) + c \cdot \alpha^k \cdot (P^{\mathcal{O}} + P^{\mathcal{A}}). \tag{5}$$

The final result now follows from standard calculations.

Theorem 4.16 *$F^{\mathcal{A}}$ is $O(\log S \cdot \log^2 P \cdot F^{\mathcal{O}})$.*

Proof. We have already bounded the processing time of $\mathcal{A}$. Once a job gets dispatched to a machine i, its waiting time can be charged to the processing done by i. Since at any time t, there are at most $\log P$ active jobs dispatched to a machine, the total waiting time of jobs after their dispatch time is at most $O(\log P \cdot P^{\mathcal{A}})$. So we just need to bound the time for which jobs are waiting in the central pool.

Let $n_k^{\mathcal{A}}(t)$ be the number of jobs of class k waiting in the central pool at time t in our algorithm. Let $n_k^{\mathcal{O}}(t)$ be the number of jobs of class k which are active at time t in $\mathcal{O}$ (note the difference in the definitions of the two quantities). Since jobs waiting in the central pool in $\mathcal{A}$ have not been processed at all, it is easy to see that $n_k^{\mathcal{A}}(t) \leq \frac{V_{\leq k}^{\mathcal{A}}(t)}{\alpha^k}$. Further, $V_{\leq k}^{\mathcal{O}}(t) \leq \alpha^k n_k^{\mathcal{O}}(t) + \cdots + \alpha n_1^{\mathcal{O}}(t)$. Combining these observations with equation (5), we get for all values of k,

$$\sum_t n_k^{\mathcal{A}}(t) \leq \sum_t \left(n_k^{\mathcal{O}}(t) + \frac{n_{k-1}^{\mathcal{O}}(t)}{\alpha} + \cdots + \frac{n_1^{\mathcal{O}}(t)}{\alpha^{k-1}} \right) + c \cdot (P^{\mathcal{O}} + P^{\mathcal{A}}).$$

We know that total flow time of a schedule is equal to the sum over all time t of the number of active jobs at time t in the schedule. So adding the equation above for all values of k and using Corollary 4.10 implies the theorem. ■

5 Open Problems

We leave open the problem of finding an online algorithm for this problem whose competitive ratio is independent of the speed of the machines.

References

[1] N. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. In *Proc. 15th Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 11–18. ACM, 2003.

[2] Baruch Awerbuch, Yossi Azar, Stefano Leonardi, and Oded Regev. Minimizing the flow time without migration. In *ACM Symposium on Theory of Computing*, pages 198–205, 1999.

[3] N. Bansal and K. Pruhs. Server scheduling in the l p norm: A rising tide lifts all boats. In *ACM Symposium on Theory of Computing*, pages 242–250, 2003.

[4] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk R. Pruhs. Online weighted flow time and deadline scheduling. *Lecture Notes in Computer Science*, 2129:36–47, 2001.

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

[6] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *ACM Symposium on Theory of Computing*, pages 363–372, 2004.

[7] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Ann. Discrete Math.*, 5:287–326, 1979.

[8] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. In *IEEE Symposium on Foundations of Computer Science*, pages 214–221, 1995.

[9] Hans Kellerer, Thomas Tautenhahn, and Gerhard J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *ACM Symposium on Theory of Cmputing*, pages 418–426, 1996.

[10] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *ACM Symposium on Theory of Computing*, pages 110–119, 1997.

[11] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *ACM Symposium on Theory of Computing*, pages 140–149, 1997.

Appendix

Proof of Theorem 4.6. Let M' denote the set of machines of class less than l. First observe that the processing time incurred by $\mathcal{A}$ on J_I is at most twice of $|M'| \cdot T$ (the factor twice comes because there may be some jobs which are dispatched during I but finish later — there can be at most one such job for a given class and a given machine). So we will be done if we can show that $F^{\mathcal{O}}_{J_I}$ is $\Omega(|M'| \cdot T)$.

Let V be the volume of jobs in J_I which are done by $\mathcal{O}$ on machines of class l or more. If $V \geq \frac{|M'| \cdot T}{\beta^{l+1}}$, then we are done because then the processing time incurred by $\mathcal{O}$ on J_I is at least $V \cdot \beta^l$. So we will assume in the rest of the proof that $V \leq \frac{|M'| \cdot T}{\beta^{l+1}}$.

Let i be a machine of class less than l. We shall say that i is *good* is i processes jobs from J_I for at least $T/4$ units of time during I in the optimal solution $\mathcal{O}$. Otherwise we say that i is *bad*. Let G denote the set of good machines. If $|G| \geq \frac{|M'|}{\beta}$, then we are done again, because $P^{\mathcal{O}}_{J_I}$ is at least $|G| \cdot T/4$. Let B denote the set of bad machines.

So we can assume that the number of good machines is at most $1/\beta$ fraction of the number of machines of class less than l. Now consider a time t in the interval $(t_b + T/2, t_e)$.

Claim 5.1 *At time t, at least $\alpha^k |M'|$ volume of jobs from J_I is waiting in $\mathcal{O}$.*

Proof. Let V_1 denote the volume of jobs from J_I which is done by $\mathcal{A}$ during (t_b, t). Let V_2 denote the volume of jobs from J_I which is done by $\mathcal{O}$ on machines of class less than l during (t_b, t). Recall that for a machine i, s_i denotes the slowness of i.

Since a machine i of class $l' < l$ does not perform jobs from J_I for at most $6\alpha^k \beta^{l'}$ amount of time during I, we see that $V_1 \geq \sum_{i \in M'} \frac{t - t_b - 6\alpha^k \cdot \beta^{c_i}}{s_i}$, where c_i denotes the class of i. Let us look at V_2 now. In $\mathcal{O}$ all bad machines do not process jobs from J_I for at least $3T/4$ units of time during I. So they do not process jobs from J_I for at least $T/4$ units of time during $(t_b, t_b + T/2)$. So $V_2 \leq \sum_{i \in M'} \frac{(t - t_b)}{s_i} - \sum_{i \in B} \frac{T}{4 s_i}$.

This shows that $V_1 - V_2 \geq \sum_{i \in B} \frac{T}{4 s_i} - \sum_{i \in M'} \frac{6\alpha^k \beta^{c_i}}{s_i}$. For a bad machine i, $T/4 - 6\alpha^k \beta^{c_i} \geq T/8$, since $c_i \leq l - 1$ (assuming β is large enough). So, we can see that this difference is at least $\sum_{i \in B} \frac{T}{8 s_i} - \sum_{i \notin B} 6\alpha^k \beta$. Since $T \geq \beta^l \cdot \alpha^k$, we see that this difference is at least $\frac{T}{\beta^{l-1}} (|B|/8 - 6|G|)$, which is at least $\frac{T|M'|}{10\beta^{l-1}}$, because we have assumed that $|B|$ is larger than $|G|$ by a sufficiently high constant factor. Recall that V is the volume of jobs in J_I which is done by $\mathcal{O}$ on machines of class l or more. Clearly, the volume of jobs from J_I which is waiting at time t in $\mathcal{O}$ is at least $V_1 - V_2 - V$. But V is at most $\frac{T|M'|}{\beta^{l+1}}$. Hence the volume waiting at time t is at least $\frac{T|M'|}{\beta^l}$. This proves the lemma. ■

Since each job in J_I is of size at most α^k, we see that at least $\Omega(|M'|)$ jobs are waiting at time t. Summing over all values of t in the range $(t_b + T/2, t_e)$ implies the theorem. ■

Proof of Lemma 4.13. Consider an i, $i_{u+1} < i \leq i_u$. Then $l^i(t) \leq l^{i_u}(t)$, otherwise we should have another suffix maximal index between i_{u+1} and i_u. So $\sum_{i=i_u}^{i_{u+1}-1} \alpha^{c^i(t)} \cdot m_{<l^i(t)} \leq m_{<l^{i_u}(t)} \cdot \sum_{i=i_u}^{i_u - 1} \alpha^{c^i(t)} \leq 2 \cdot m_{<l^{i_u}(t)} \cdot \alpha^{c^{i_u}(t)}$. So we get $\sum_i \alpha^{c^i(t)} \cdot m_{<l^i(t)} \leq 2 \cdot \sum_{i_u} m_{<l^{i_u}(t)} \cdot \alpha^{c^{i_u}(t)}$.

Now we consider the sum $\sum_i \sum_{l \geq l^i(t)} \frac{P_l^{\mathcal{O}}(t^{i+1},t^i)}{\beta^{l-1}}$. Fix an i, $i_{u+1} < i \leq i_u$. Using Claim 4.12, we see that $P_l^{\mathcal{O}}(t^{i+1}, t^i) \leq m_l \cdot \beta^{l^i(t)+1} \cdot \alpha^{c^i(t)}$. So we get

$$\sum_{l \geq l^i(t)} \frac{P_l^{\mathcal{O}}(t^{i+1}, t^i)}{\beta^{l-1}} \leq \sum_{l \geq l^{i_u}(t)} \frac{P_l^{\mathcal{O}}(t^{i+1}, t^i)}{\beta^{l-1}} + \sum_{l=l^i(t)}^{l^{i_u}(t)-1} \frac{m_l \cdot \beta^{l^i(t)+1} \cdot \alpha^{c^i(t)}}{\beta^{l-1}}$$

Now the second term on the right hand side above is at most $\beta^2 \cdot m_{<l^{i_u}(t)} \alpha^{c^i(t)}$. So we get

$$\sum_{i=i_u}^{i_{u+1-1}} \sum_{l \geq l^i(t)} \frac{P_l^{\mathcal{O}}(t^{i+1}, t^i)}{\beta^{l-1}} \leq \sum_{l \geq l^{i_u}(t)} \frac{P_l^{\mathcal{O}}(t^{i_{u+1}}, t^{i_u})}{\beta^{l-1}} + 2\beta^2 \alpha^{c^{i_u}(t)} m_{<l^{i_u}(t)},$$

because $\alpha^{c^i(t)}$ scales down geometrically as i increases.

Finally note that $\sum_{i_u} \alpha^{c^{i_u}(t)} m_{<l^{i_u}(t)}$ is at most twice of $\sum_{i_u} \alpha^{c^{i_u}(t)} \cdot m_{(l^{i_{u-1}}(t), l^{i_u}(t)-1)}$, because $\alpha^{c^i(t)}$ scale down geometrically. This proves the lemma (using (2)). ■