ACADEMIC PRESS

Available at www.ComputerScienceWeb.com POWERED BY SCIENCE @ DIRECT•

Journal of Computer and System Sciences 68 (2004) 80–95

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

http://www.elsevier.com/locate/jcss

Average stretch without migration☆

Luca Becchetti,[a,*,1] Stefano Leonardi,[a] and S. Muthukrishnan[b]

[a] *Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", via Salaria 113, 00198 Rome, Italy*
[b] *Department of Computer and Information Science, Rutgers University, 319 Core Building, Freylinghuysen Road, Piscataway, NJ, USA*

Received 26 February 2001; revised 5 February 2003

Abstract

We study the problem of scheduling parallel machines online, allowing preemptions while disallowing migration of jobs that have been scheduled on one machine to another. For a given job, we measure the quality of service provided by an algorithm by the *stretch* of the job, defined as the ratio between the amount of time spent by the job in the system (the *response time*) and its processing time. For a sequence of jobs, we measure the performance of an algorithm by the *average stretch* achieved over all jobs. The scheduling goal is to minimize the average stretch. This problem is of relevance in many applications, e.g., wireless data servers and distributed server systems in wired networks. We prove an $O(1)$ competitive ratio for this problem. The algorithm for which we prove this result is the one proposed in Awerbuch et al. (Proceedings of the ACM Symposium on the Theory of Computing (STOC '99), 1999, pp. 198–205) that has (tight) logarithmic competitive ratio for minimizing the average response time. Thus, the algorithm in Awerbuch et al. (Proceedings of the ACM Symposium on the Theory of Computing (STOC '99), 1999, pp. 198–205) simultaneously performs well for average response time as well as average stretch. We prove the $O(1)$ competitive ratio against an adversary who not only knows the entire input ahead of time, but is also allowed to migrate jobs. Thus, our result shows that migration is not necessary to be competitive for minimizing average stretch; in contrast, we prove that preemption is essential, even if randomization is allowed.

Keywords: Scheduling; Average stretch; Migration

☆A preliminary version of this paper appeared in the Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00), pp. 548–557, 2000.

*Corresponding author.

E-mail addresses: becchett@dis.uniroma1.it (L. Becchetti), leon@dis.uniroma1.it (S. Leonardi), muthu@cs.rutgers.edu (S. Muthukrishnan).

[1]Partially supported by the IST Programme of the EU under Contract IST-1999-14186 (ALCOM-FT) and IST-2000-14084 (APPOL), and by the MURST Projects "Algorithms for Large Data Sets: Science and Engineering" and "Resource Allocation in Computer Networks".

0022-0000/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2003.06.001

1. Introduction

There is increasing wireless support for data networks because of the growth in mobile clients (e.g., palm tops) as well as "data-centric" applications (e.g., traffic information systems, stock or sports tickers, wireless internet access, etc.). Data servers in wireless networks receive a continuous stream of requests for documents from users, requests varying over several orders of magnitude in size. They feed data to the users over multiple channels[2]. Scheduling the delivery of data over multiple channels is an instance of the problem we study here. A similar scheduling problem arises in wired data networks as well. In order to increase the capacity of servers in wired networks, systems use distributed servers [10]. Requests for documents arrive at a central dispatcher which must schedule them to one of the multiple servers for delivery.

Scheduling data delivery as in the two motivating applications above is the classical multiprocessor scheduling problem where each channel or server may be thought of as a machine, and each request for a document as a job. Data must be delivered in a responsive manner—this is typically abstracted as the problem of scheduling jobs on the multiprocessor system to optimize some metric.

There are two issues to clarify further, namely, the model for scheduling jobs and the metric to be optimized.

In our model, jobs may be *preempted*, that is, a job that is being processed may be interrupted and resumed later after processing other jobs in the interim. Preemption improves the system performance, however there is a penalty. In multiprocessor computer systems, it requires a context switching at a processor but this cost is reasonably small[3]. In the case of the distributed server system we described, interrupting the delivery of a large file to service small ones is reasonable if it improves system performance because the end users see bursty data delivery in any case due to the network behaviour once the data leave the server. The cost of preemption here is the space needed to buffer partly delivered documents. There are additional costs in the wireless data server case, such as the energy spent by a mobile client to stay connected for the duration of the document delivery, which is typically larger on average if jobs get preempted. In current wireless systems, clients can go into doze (dormant) mode and wake up in time to receive the remainder of a transmission (there is a control channel to regulate this). This mechanism reduces the cost of preemption. To summarize, the cost of preemption is quite reasonable in existing applications.

In our model, jobs may not be *migrated*, that is, once a machine begins processing a job, that job must be processed only by that machine; in particular, if one job is preempted, no other machine may process that job even if it is idle. Like preemption, migration also improves the performance of the system but it is expensive. In multiprocessor computer systems and distributed server systems, moving a job from one machine to another calls for communication and data transfer between machines, which is time consuming. It requires clients to switch channels in

[2] Depending on the wireless technology, the term "channels" has different meanings. In frequency division multiple access (FDMA), it is frequency. In time division multiple access (TDMA), it is time slots. In code division multiple access (CDMA), it is codes. See [1,9] and references therein for details.

[3] This in particular holds in systems that support threads [7,11,18,20].

wireless data servers which may be expensive too. It is thus desirable to avoid migration but still guarantee reasonable performance.

The metric to be optimized depends on the notion of responsiveness. Traditionally, the time a job spends in the system, namely, the *response time* of a job defined as the difference between its completion time and its arrival time, has been considered a suitable measure. More recently, *stretch*, namely, the factor by which a request is slowed down relative to the time it takes on an unloaded system, has emerged as a more suitable measure. Formally, the stretch of a job is the ratio between its response time and its processing time. Intuitively, it relates the users' waiting time to their demands and measures the quality of service provided to a job in proportion to its requirements. The performance of a system is measured as the average performance per user. Average stretch (equivalently, total stretch) may be a better indicator of the system performance than average response time because it reflects per user expectation better, and it is a good indicator of the overall system load—a system with a large average stretch is necessarily overloaded while one with a large average response time need not be overloaded, since a single large job would result in large response times. Stretch, more precisely the average stretch, has been used to study the performance of several applications, e.g., in databases [17] and operating systems [12]. It is also the chosen metric in the context of Web servers and database servers [6,1].

We study the problem of online scheduling to minimize the average stretch on multiple machines; we do not allow migration, but do allow preemption. Our main result is to show an $O(1)$ competitive algorithm for this problem.[4] In fact, the algorithm that achieves $O(1)$ competitive ratio for average stretch is the one proposed by Awerbuch et al. [2], which we refer to as AALR. AALR was previously shown to achieve logarithmic upper bound for average response time against a migratory adversary and this bound is tight [2]. Combined with our result, it follows that AALR performs well for both measures simultaneously. (When migration is allowed, a similar phenomenon occurs: the algorithm shortest remaining processing time (SRPT) simultaneously achieves the best asymptotic competitive ratio for average response time [16] as well as average stretch [8].)

Our main technical contribution is the competitive analysis of AALR for the average stretch metric. We show that this algorithm is $O(1)$ competitive even if the optimal offline adversary is allowed to migrate jobs; the algorithm of course does not migrate jobs. Thus migration is not necessary to be $O(1)$-competitive for minimizing average stretch. On the other hand, preemption is essential. We show an $\Omega(\Delta)$ lower bound on the competitive ratio of randomized online algorithms, where Δ is the ratio between the maximum and the minimum processing time of a job.

The role of migration in scheduling has been studied for a long time. There has been some progress recently. Following the work of [15], Kalyanasundaram and Pruhs [13] investigated the effect of migration in real-time multiprocessor scheduling with preemption, showing that non-migratory schedules can be competitive if they are allowed to use $6m - 5$ processors against an adversary who only uses m processors. As mentioned earlier, Awerbuch et al. [2] presented AALR for minimizing the total flow time without migration and its logarithmic competitive ratio is asymptotically tight.

[4] The competitive ratio of an online algorithm [19] is defined as the maximum over any input sequence of the ratio between the cost of the online algorithm and the cost of the adversary that knows the whole input sequence in advance and serves it optimally.

While average response time has been studied extensively [3,14,16,2], the analytical study of stretch was initiated only recently [4]. All results presented in [4], however, are for the *maximum stretch* metric only. Minimizing the maximum stretch is suitable for making worst case per job guarantees; however, it may overload the system and the average performance seen by users may suffer. The average stretch measure, although experimentally evaluated in many applications, was not formally studied until recently [8]. There the authors showed that the optimal competitive ratio for average stretch is greater than the optimal competitive ratio for average response time; in contrast, average stretch is easier than average response time on multiprocessor machines. Their results assume that jobs may be migrated (One can easily generate an input on which the SRPT algorithm analyzed in [8] induces $\Omega(n)$ migrations where n is the number of outstanding requests at any time).

Overview of our analysis: AALR holds a pool of jobs that have not yet been assigned to a machine and a stack for each machine comprising jobs that have already begun processing on that machine (and will be processed up to completion on that machine). The jobs in a machine's stack are ordered by increasing remaining processing time. The job on the top of the stack is scheduled for processing. Roughly speaking, a job is moved from the pool to a stack if its processing time is smaller than the remaining processing time of every other job in the stack by at least a factor 2 or if the stack is empty. There are two contributions to the stretch of a job. The first contribution is obtained from the time spent in a machine's stack. When a new job is assigned to a machine, all jobs in that machine's stack are further delayed by the processing time of the newly assigned job. AALR implies that the jobs in a stack have exponentially decreasing processing times (see Lemma 1); hence, the total increase in the stretch of the jobs in the stack is bounded by 2 (see Lemma 2) which proves the $O(1)$ bound for this first contribution by using the number of released jobs as a trivial lower bound on the optimum average stretch. The second contribution to the total stretch of the jobs is given by the time spent in the system's pool. Roughly speaking, Lemma 6 states that at any time, the difference between the number of jobs with about the same processing times waiting in the pool and the number of jobs with equal or smaller processing time uncompleted in the optimal solution is at most $6m$, where m is the number of machines. We then observe that the m smallest jobs in the pool are responsible for most of the additional contribution to the total stretch with respect to the optimal solution. Finally, Lemma 9 allows to infer that the total stretch of the m smallest jobs in the pool is bounded by $O(1)$ times the number of jobs that are released, which leads us to our main result.

The entire analysis we present is in some sense simpler than the one in [8] for migratory schedules. The analysis also follows quite different lines from [2,16]. The proof of $O(\log P)$ competitiveness for average flow time for AALR or SRPT relies on local competitiveness, i.e., at any time the number of uncompleted jobs is at most $O(\log P)$ times the optimum. We do not know of any proof based on local competitiveness for average stretch. We need to relate the difference in contribution to total stretch on the entire schedule to the total number of jobs released. On the other hand, the proof is much simpler than the proof of $O(\log n)$ competitiveness for AALR and SRPT with respect to average flow time.

Structure of the paper: In Section 2, we define our scheduling problem formally and state the algorithm we analyze. We define notation in Section 3. In Section 4, we analyze the AALR algorithm to show that it is $O(1)$-competitive. The constant here is 49. In Section 5, we present our randomized non-preemptive lower bound and in Section 6 concluding remarks.

2. Problem definition and algorithm

2.1. Problem definition

We are given a set $\mathcal{J}$ of n jobs and a set of m identical machines. Each job is described by a pair (r_j, p_j), where r_j is the *release time* of job j and p_j its *processing time*. The model we consider allows preemption, but not migration of jobs. Namely, a job running on a certain machine may be preempted and its execution may be resumed later, but once a job has been assigned to a machine it will be processed on that machine until completion, hence a preempted job must resume its execution on the same machine on which its processing started. Each machine can process at most one job at any time and a job cannot start being processed before its release time. Let C_j be the *completion time* of job j, $F_j = C_j - r_j$ be the *response time* (also called *flow time*) of job j, then the *stretch* of job j upon completion is $\frac{F_j}{p_j}$. The *average stretch* of a schedule is defined as $\frac{1}{n}\sum_{j \in \mathcal{J}} \frac{F_j}{p_j}$.

The goal is to minimize the average stretch, or equivalently the total stretch $S = \sum_{j \in \mathcal{J}} \frac{F_j}{p_j}$ for every instance of the problem. In the off-line version of the problem all jobs (for each job, its release time and processing time) are known in advance, while in the on-line setting jobs arrive over time and at any instant, the algorithm has to make its decisions without knowledge of jobs that have not yet been released.

2.2. Algorithm

In this section, we describe the algorithm AALR that was first presented in [2]. We then introduce some notation and definitions which will be useful in the sequel. In the following, a job is called *alive* at time t if it has been released, but it has not yet been completed. A job j is of *class* k if p_j lies in the interval $[2^k, 2^{k+1})$. *Observe that the class of a job does not change with time.* We denote by $k_{\min}$ and $k_{\max}$ the minimum and maximum class, respectively.

The algorithm classifies jobs that are alive according to their remaining processing time. Given a job j, denote by $\rho_j(t)$ its remaining processing time at t. The *rank* of a job j is k at time t if $\rho_j(t)$ falls in $[2^k, 2^{k+1})$. Notice that while the class of a job is fixed, its rank changes during execution. AALR holds a pool of jobs that are alive but have not been processed at all (so their ranks are the same as their classes). In addition, for each machine the algorithm holds a stack of jobs. The stack of machine i holds jobs that are alive and have already been processed by machine i.

We observe that the definition of rank given here coincides with the definition of class given in [2], while in our paper the class only refers to the size of jobs when they are released. The algorithm works as follows:

- Each machine processes the job on top of its stack.
- When a new job arrives, the algorithm looks for a machine that is currently idle or is processing a job whose rank is higher than the class of the newly released one. If it finds one, the new job is pushed into that machine's stack and its processing begins. Otherwise the job is inserted into the pool.
- When a job is completed on some machine, it is popped from its stack. The algorithm then compares the rank of the job now on top of that machine's stack with the minimum class of a

job in the pool. If the minimum is in the pool then a job in the pool whose class achieves this minimum is pushed on top of the machine's stack and its processing begins.

When a job enters some machine's stack it remains there until completion, so no migration occurs.

3. Preliminaries

An alternative (but equivalent) definition of the total stretch of a set of jobs is as follows. Let $Q(t)$ be the set of alive jobs at time t for a generic algorithm. The total stretch $S = \sum_{j \in \mathcal{J}} \frac{F_j}{p_j}$ of a set $\mathcal{J}$ of jobs can be defined as

$$S = \int_{t \geqslant 0} \left(\sum_{j \in Q(t)} \frac{1}{p_j} \right) dt.$$

We consider OPT, the offline optimal algorithm, as our adversary; OPT is further allowed to migrate jobs. Let S^{A} and S^{OPT} denote the total stretch of the algorithms AALR and OPT, respectively. Trivially, $S^{\mathrm{OPT}} \geqslant n$, a lower bound we use later. Let $Q^{\mathrm{A}}(t)$ be the set of alive jobs in AALR's schedule at time t. $Q^{\mathrm{A}}(t)$ is partitioned between $Q^{\mathrm{A,P}}(t)$ and $Q^{\mathrm{A,S}}(t)$, respectively denoting the set of jobs in the pool and in the system stacks of AALR at time t. $\mathcal{T}$ denotes the set of time instants in which the pool is not empty. S^{A} can be seen as being the sum of two terms, $S^{\mathrm{A,S}}$ and $S^{\mathrm{A,P}}$, where

$$S^{\mathrm{A,S}} = \int_{t \geqslant 0} \sum_{j \in Q^{A,S}(t)} \frac{1}{p_j} dt$$

and

$$S^{\mathrm{A,P}} = \int_{t \geqslant 0} \sum_{j \in Q^{A,P}(t)} \frac{1}{p_j} dt.$$

In the following, let $\delta_{=k}^{\mathrm{A,P}}(t)$ $(\delta_{\leqslant k}^{\mathrm{A,P}}(t))$ denote the number of jobs of class k (at most k) in the pool at time t and let $V_{=k}^{\mathrm{A,P}}(t)$ $(V_{\leqslant k}^{\mathrm{A,P}}(t))$ denote their volume, i.e., their overall remaining processing time at time t. Furthermore, let $\delta_{=k}^{\mathrm{OPT}}(t)$ $(\delta_{\leqslant k}^{\mathrm{OPT}}(t))$ and $V_{=k}^{\mathrm{OPT}}(t)$ $(V_{\leqslant k}^{\mathrm{OPT}}(t))$, respectively, denote the number and the volume of the jobs of class k (at most k) that are in the system at time t in OPT. We also set $\Delta_{=k} V(t)$ $(\Delta_{\leqslant k} V(t))$ to denote the difference $V_{=k}^{\mathrm{A,P}}(t) - V_{=k}^{\mathrm{OPT}}(t)$ $(V_{\leqslant k}^{\mathrm{A,P}}(t) - V_{\leqslant k}^{\mathrm{OPT}}(t))$.

4. Analysis

We show in this section that AALR achieves an $O(1)$ competitive ratio with respect to OPT. More precisely, here we prove an upper bound 49 on the competitive ratio of AALR.

We separately bound the contribution to the total stretch given by the alive jobs in the system's stacks and the contribution to the total stretch given by the alive jobs in the system's pool. We start by bounding the contribution to the total stretch given by the alive jobs in the system's stacks, namely $S^{\mathrm{A,S}}$. The following lemma (a similar claim is also in [2]) states that, at any time and for any stack, no two jobs in the same stack may have the same rank or be in the same class.

Lemma 1. *In each stack the jobs are ordered in* (i) *strictly increasing rank order*; (ii) *strictly increasing class order*.

Proof. Consider a machine's stack. If the stack is empty, part (i) and (ii) of the claim hold. By induction, assume the claim is true for all jobs currently held by the stack. Three different events may happen: (a) a job is moved from the pool to the stack; (b) a job in the stack is completed; (c) the rank of the currently processed job is decreased by 1. In case (a), a job is pushed into the stack for execution. The rank of the job, which equals its class, is lower than the rank of any other job in the stack and the claim still holds. If case (b) occurs, the claim is obviously still true. In case (c), the rank of the job with lowest rank is decreased by 1, hence the claim is still true. □

Lemma 2. $S^{\mathrm{A,S}} \leqslant 3S^{\mathrm{OPT}}$.

Proof. The release of a job increases the total stretch by at least 1. We prove the claim by showing that the assignment of a job to a machine determines an increase of $S^{\mathrm{A,S}}$ of at most 3. Assume job j is assigned to machine i. The corresponding increase of $S^{\mathrm{A,S}}$ is given by the stretch that will be paid by job j if processed non-preemptively up to completion on machine i plus the increase in the stretch of all jobs previously in stack i whose completion will be delayed by p_j. Consider the stack of the ith machine. Assume job j moved from the pool to the ith stack at time t. Assume q jobs are in stack i at time t prior to the arrival of job j. Order those jobs from 1 to q by increasing class, and denote by $k_1, \ldots, k_q$ the classes of those jobs. By Lemma 1, $k_1 < \cdots < k_q$. The increase in the stretch of the q jobs is then bounded

$$p_j\left(\frac{1}{2^{k_1}} + \frac{1}{2^{k_2}} + \cdots + \frac{1}{2^{k_q}}\right) \leqslant 2$$

since job j is of class strictly less than k_1. We should also consider an additional contribution of 1 for the stretch of job j for which the claim is proved. □

In the following we will turn our attention to the contribution to the total stretch due to the jobs in the pool of AALR, namely $S^{\mathrm{A,P}} = \int_{t \in \mathscr{T}} \sum_{j \in Q^{\mathrm{A,P}}(t)} \frac{1}{p_j}\, dt$.

Since the size of a job of class k is in $[2^k, 2^{k+1})$, we bound $S^{\mathrm{A,P}}$ by

$$S^{\mathrm{A,P}} \leqslant \int_{t \in \mathscr{T}} \sum_{k=k_{\min}}^{k_{\max}} \frac{\delta_{=k}^{\mathrm{A,P}}(t)}{2^k}\, dt.$$

Consider a time instant $t \in \mathscr{T}$ and let $(t_0, t]$ denote the maximum left open interval ending at time t and contained in $\mathscr{T}$. Let t_k, $t_0 \leqslant t_k \leqslant t$ denote the last time a job of rank bigger than k was executed by AALR on a machine. Only jobs of rank at most k are executed on the m machines in the interval $(t_k, t]$.

Claim 3. *At any time $t \in \mathcal{T}$ no machine is idle.*

Proof. It follows by the definition of the algorithm since the pool is not empty. □

Lemma 4. $\Delta_{\leqslant k} V(t_k) \leqslant 0$.

Proof. Consider any $\varepsilon > 0$ sufficiently small. Then $V^{\mathrm{A,P}}_{\leqslant k}(t_k - \varepsilon) = 0$ from the definition of t_k. Two cases may arise:

- $t_k = t_0$. In this case there is an idle machine at $t_k - \varepsilon$, hence the pool is empty at t_k.
- $t_k > t_0$. A job of rank strictly greater than k is processed at $t_k - \varepsilon$, for any ε sufficiently small, hence the pool contains no job of class k or less at any instant sufficiently close to t_k. All jobs of class at most k released at time t_k contribute to both $V^{\mathrm{OPT}}(t_k)$ and $V^{\mathrm{A,P}}(t_k)$ then implying the claim. □

Lemma 5. $\forall t \in \mathcal{T} : \Delta_{\leqslant k} V(t) \leqslant m 2^{k+2}$.

Proof. No job of *rank* higher than k is processed by AALR in $(t_k, t]$, hence no job of class higher than k is pushed into a stack in $(t_k, t]$. The only jobs of class higher than k that are processed in $(t_k, t]$ are those in the stacks at time t_k. $\Delta_{\leqslant k} V(t)$ may grow only because these jobs are processed in $(t_k, t]$ by the algorithm, while OPT may only work on jobs of class less than or equal to k. However, the overall volume of these jobs that is processed in $(t_k, t]$ is at most $m2^{k+2}$ over all m machines, i.e., by Lemma 1, at most one job for each rank less or equal than k. The claim follows since no machine in AALR schedule is idle in $(t_k, t]$ and from the claim of Lemma 4. □

In the following, we omit t when clear from the context.

Lemma 6. *For any $k_{\min} \leqslant k_1 \leqslant k_2 \leqslant k_{\max}$ and for any $t \in \mathcal{T}$ we have*

$$\sum_{k=k_1}^{k_2} \frac{\delta^{\mathrm{A,P}}_{=k}(t)}{2^k} \leqslant \frac{6m}{2^{k_1}} + 2 \sum_{k \leqslant k_2} \frac{\delta^{\mathrm{OPT}}_{=k}(t)}{2^k}.$$

Proof. We can write

$$\sum_{k=k_1}^{k_2} \frac{\delta^{\mathrm{A,P}}_{=k}(t)}{2^k} \leqslant \sum_{k=k_1}^{k_2} \frac{\Delta_{\leqslant k} V(t) - \Delta_{\leqslant k-1} V(t) + V^{\mathrm{OPT}}_{=k}(t)}{4^k}$$

$$\leqslant \frac{\Delta_{\leqslant k_2} V(t)}{4^{k_2}} + 3 \sum_{k=k_1}^{k_2-1} \frac{\Delta_{\leqslant k} V(t)}{4^{k+1}} - \frac{\Delta_{\leqslant k_1-1} V(t)}{4^{k_1}} + 2 \sum_{k=k_1}^{k_2} \frac{\delta^{\mathrm{OPT}}_{=k}(t)}{2^k}.$$

Observing

$$-\Delta_{\leqslant k_1-1} V(t) \leqslant V^{\mathrm{OPT}}_{\leqslant k_1-1}(t)$$

and

$$\frac{V^{\mathrm{OPT}}_{\leqslant k_1-1}(t)}{4^{k_1}} = \frac{\sum_{k=k_{\min}}^{k_1-1} V^{\mathrm{OPT}}_{=k}(t)}{4^{k_1}} \leqslant \sum_{k=k_{\min}}^{k_1-1} \frac{V^{\mathrm{OPT}}_{=k}(t)}{4^k}$$
$$= 2\sum_{k=k_{\min}}^{k_1-1} \frac{V^{\mathrm{OPT}}_{=k}(t)}{2^k \times 2^{k+1}} \leqslant 2\sum_{k=k_{\min}}^{k_1-1} \frac{\delta^{\mathrm{OPT}}_{=k}(t)}{2^k},$$

we obtain

$$\sum_{k=k_1}^{k_2} \frac{\delta^{\mathrm{A,P}}_{=k}(t)}{2^k} \leqslant \frac{\Delta_{\leqslant k_2} V(t)}{4^{k_2}} + 3\sum_{k=k_1}^{k_2-1} \frac{\Delta_{\leqslant k} V(t)}{4^{k+1}} + 2\sum_{k=k_{\min}}^{k_2} \frac{\delta^{\mathrm{OPT}}_{=k}(t)}{2^k}.$$

Finally, from Lemma 5, we conclude after straightforward manipulations

$$\frac{\Delta_{\leqslant k_2} V(t)}{4^{k_2}} + 3\sum_{k=k_1}^{k_2-1} \frac{\Delta_{\leqslant k} V(t)}{4^{k+1}} \leqslant \frac{4m}{2^{k_2}} + \sum_{k=k_1+1}^{k_2} \frac{6m}{2^k}$$
$$\leqslant \frac{6m}{2^{k_1}} - \frac{2m}{2^{k_2}} \leqslant \frac{6m}{2^{k_1}},$$

which gives the desired claim. □

Let $k(t)$ be the lowest class of a job in the pool at time $t \in \mathcal{T}$. Let $k'(t)$ be the minimum class of a job in the pool such that $\sum_{k=k(t)}^{k'(t)} \delta^{\mathrm{A,P}}_{=k}(t) \geqslant m$. If $\sum_{k=k_{\min}}^{k_{\max}} \delta^{\mathrm{A,P}}_{=k}(t) < m$ we assume by convention $k'(t) = k_{\max} + 1$.

For the contribution to the total stretch of the jobs in the pool of the algorithm we have the following expression:

$$S^{\mathrm{A,P}} \leqslant \int_{t \in \mathcal{T}} \sum_{k=k(t)}^{k_{\max}} \frac{\delta^{\mathrm{A,P}}_{=k}(t)}{2^k}\, dt$$
$$= \int_{t \in \mathcal{T}} \sum_{k=k(t)}^{k'(t)-1} \frac{\delta^{\mathrm{A,P}}_{=k}(t)}{2^k}\, dt$$
$$+ \int_{t \in \mathcal{T}} \sum_{k=k'(t)}^{k_{\max}} \frac{\delta^{\mathrm{A,P}}_{=k}(t)}{2^k}\, dt \qquad (1)$$

$$\begin{aligned}
&\leqslant \int_{t\in\mathscr{T}} \sum_{k=k(t)}^{k'(t)-1} \frac{\delta^{\mathrm{A,P}}_{=k}(t)}{2^k}\,dt \\
&\quad + \int_{t\in\mathscr{T}} \left(\frac{6m}{2^{k'(t)}} + 2 \sum_{k=k_{\min}}^{k_{\max}} \frac{\delta^{\mathrm{OPT}}_{=k}(t)}{2^k} \right) dt \\
&\leqslant \int_{t\in\mathscr{T}} \sum_{k=k(t)}^{k'(t)-1} \frac{\delta^{\mathrm{A,P}}_{=k}(t)}{2^k}\,dt + \int_{t\in\mathscr{T}} \frac{6m}{2^{k'(t)}}\,dt + 4S^{\mathrm{OPT}},
\end{aligned}$$

where the third inequality follows from Lemma 6.

To prove our $O(1)$ bound we will separately limit the first two terms of Eq. (1). For any time t, consider the m jobs of smallest class in the on-line pool, denoted in the following as the *bottom m jobs* of the schedule. If less than m jobs are in the pool, assume that the remaining $m - \delta^{\mathrm{A,P}}_{\leqslant k_{\max}}(t)$ jobs are of class $k_{\max}+1$. The bottom m jobs are of class between $k(t)$ and $k'(t)$. Let $k_1(t), \dots, k_m(t)$ be the classes of these jobs, where $k_1(t) = k(t)$ and $k_m(t) = k'(t)$. Now define

$$S^{\mathrm{l}}(t) = \sum_{j=1}^{m} \frac{1}{2^{k_j(t)}}.$$

Lemma 7.

$$\int_{t\in\mathscr{T}} \sum_{k=k(t)}^{k'(t)-1} \frac{\delta^{\mathrm{A,P}}_{=k}(t)}{2^k}\,dt \leqslant \int_{t\in\mathscr{T}} S^{\mathrm{l}}(t)\,dt.$$

Proof. From the definition of $S^{\mathrm{l}}(t)$, it follows that

$$S^{\mathrm{l}}(t) = \sum_{k=k(t)}^{k'(t)-1} \frac{\delta^{\mathrm{A,P}}_{=k}(t)}{2^k} + \frac{m(t)}{2^{k'(t)}},$$

where $m(t) = m - \sum_{k=k(t)}^{k'(t)-1} \delta^{\mathrm{A,P}}_{=k}(t) \geqslant 0$. This implies the thesis. $\square$

Lemma 8.

$$\frac{m}{2^{k'(t)}} \leqslant S^{\mathrm{l}}(t).$$

Proof. The proof follows since $k_m(t) = k'(t)$ and $k_1(t) \leqslant k_2(t) \leqslant \cdots \leqslant k_m(t)$. $\square$

To put an upper bound on the terms of Eq. (1) we will show $\int_{t\in\mathscr{T}} S^{\mathrm{l}}(t)\,dt = O(1)S^{\mathrm{OPT}}$.

We partition every maximal continuous subset of time instants contained in $\mathscr{T}$ (set of instants when the pool is not empty) into a collection of disjoint intervals $I_s = [t_s, r_s)$, $s = 1, \dots$. Interval I_1 starts with the beginning of the first maximal subset. Interval I_s starts with the end of I_{s-1}. Interval

I_s ends with the end of the maximal subset or when one among the bottom m jobs, let us say of class k_{I_s}, remains in the pool within interval I_s for a continuous subinterval of length $2^{k_{I_s}+1}$. Let I be the generic interval contained in $\mathscr{T}$. Denote by n_I the number of jobs that are either released, moved to execution or completed in interval I. In the following we will prove a bound on

$$\int_{t\in I} S^1(t)\, dt = \int_{t\in I} \sum_{j=1}^{m} \frac{1}{2^{k_j(t)}}\, dt.$$

More precisely, we prove the following.

Lemma 9.

$$\int_{t\in I} S^1(t)\, dt \leqslant 2n_I.$$

Proof. We first consider the contribution to $\int_{t\in I} S^1(t)\, dt$ of those jobs that are among the bottom m jobs for the whole I. (Observe that there are no such jobs if I is the last interval of a maximal subset of $\mathscr{T}$.) At most m jobs are among the bottom m jobs for the whole I. The total stretch of these jobs is upper bounded by $2m$, since none of these jobs has been in the pool during I for more than twice its processing time. By the definition of interval I, during a subinterval of I of size 2^{k_I+1}, a job of class k_I was always in the pool. All jobs in execution in this subinterval are of rank at most k_I. It follows that at least m jobs must be completed on the m machines during interval I. We then have a contribution to n_I of at least m due to those jobs that are completed during I.

To complete the proof, we still have to consider the contribution to $\int_{t\in I} S^1(t)\, dt$ of those jobs that have entered or left the bottom m jobs during interval I. All these jobs have not been in the pool for a continuous interval bigger than twice their processing time, otherwise interval I would have been interrupted. Their contribution to the total stretch in I for every continuous interval of time in which they have been among the bottom m jobs is then less than 2. Such jobs have left the bottom m jobs either because they have been pushed onto a stack or because they have been replaced among the bottom m jobs by other jobs that have been released. Then, we match every continuous interval in which a job is among the bottom m jobs with a job that is either released or with a job that is pushed onto a stack. The claim is then proved. □

Corollary 10.

$$\int_{t\in \mathscr{T}} S^1(t)\, dt \leqslant 6n.$$

Proof. Partition $\mathscr{T}$ into a set of intervals as described above. Let I be the generic interval, let n_I be the set of jobs that are either released, pushed onto a stack or completed during interval I. Then:

$$\sum_I \int_{t\in I} S^1(t)\, dt \leqslant 2 \sum_I n_I \leqslant 6n$$

since every job is considered at most three times, when released, pushed onto a stack or completed. □

Theorem 11. $S^{A} \leqslant 49 S^{OPT}$.

Proof. Eq. (1), together with Lemmas 2, 7, 8 and Corollary 10 imply

$$\begin{aligned} S^{A} &= S^{A,S} + S^{A,P} \leqslant 7 \int_{t \in \mathcal{T}} S^{l}(t)\,dt + 7S^{OPT} \\ &\leqslant 42n + 7S^{OPT} \leqslant 49 S^{OPT}. \qquad \square \end{aligned}$$

5. Randomized lower bound for non-preemptive scheduling

In this section we prove that, while migration is not a fundamental aspect in order to obtain asymptotic good competitive ratios, an algorithm that is not allowed to employ preemption can yield very poor performance in the worst case even if randomization is allowed.

In the following, we assume that the scheduling algorithm cannot preempt a job, but it may delay the service of a job as long as necessary. Let Δ denote the ratio between the biggest and the smallest size of a job. We prove that any non-preemptive randomized on-line algorithm cannot behave but very poorly. The following theorem holds.

Theorem 12. *The competitive ratio of any non-preemptive, on-line randomized algorithm for the minimization of the average stretch on m parallel machines is* $\Omega(\Delta)$.

Proof. We use the application of Yao's lemma to prove lower bounds for randomized on-line algorithms [5]. We give a lower bound on the ratio between the expected cost of any deterministic algorithm and the optimal off-line cost for a specific distribution one the input sequences for the problem. The adversary presents a sequence drawn from the following probability distribution, where s is a parameter that will be fixed equal to Δ^2:

```
for $i = 1, 2, \ldots, s$
    1. Release $m$ jobs of size 1 at time $\frac{5}{2}(i-1)$.
    2. With equal probability release
    [2.1] either $m$ jobs of size $1/\Delta$ at any time
        $\frac{5}{2}(i-1) + 1/2 + (k-1)/\Delta$, $k = 1, \ldots, \Delta/2$.
    [2.2] or $m$ jobs of size $1/\Delta$ at any time
        $\frac{5}{2}(i-1) + 1 + (k-1)/\Delta$, $k = 1, \ldots, \Delta/2$.
    3. Release $m$ jobs of size $1/\Delta$ at any time
        $\frac{5}{2}(i-1) + 2 + (k-1)/\Delta$, $k = 1, \ldots, \Delta/2$.
    endfor
```

Consider the ith interval $[\frac{5}{2}(i-1), \frac{5}{2}i)$, $i = 1, \ldots, s$. We denote as jobs of *type j* the jobs presented in step j $(j = 1, 2.1, 2.2, 3)$ of the sequence above.

Let x_i be the number of jobs of size 1 that are started in interval i. Assume that within interval i, a set $\mathscr{J}_i^1$ of x_i^1 jobs of size 1 are started before time $\frac{5}{2}(i-1)+\frac{1}{2}$. These jobs are scheduled for the whole interval $[\frac{5}{2}(i-1)+\frac{1}{2},\frac{5}{2}(i-1)+1)$. A set $\mathscr{J}_i^2$ of x_i^2 jobs of size 1 are started in interval $[\frac{5}{2}(i-1)+\frac{1}{2},\frac{5}{2}(i-1)+1)$. These are scheduled for the whole interval $[\frac{5}{2}(i-1)+1,\frac{5}{2}(i-1)+\frac{3}{2})$. A set $\mathscr{J}_i^3$ of x_i^3 jobs of size 1 are started in the interval $[\frac{5}{2}(i-1)+1,\frac{5}{2}(i-1)+2)$. At least $x_i^3/2$ jobs of $\mathscr{J}_i^3$ are scheduled either for the whole second half of $[\frac{5}{2}(i-1)+1,\frac{5}{2}(i-1)+\frac{3}{2})$ or first half of $[\frac{5}{2}(i-1)+2,\frac{5}{2}(i-1)+\frac{5}{2})$. Finally a set $\mathscr{J}_i^4$ of $x_i^4=x_i-x_i^1-x_i^2-x_i^3$ jobs of size 1 are started in the interval $[\frac{5}{2}(i-1)+2,\frac{5}{2}(i-1)+\frac{5}{2})$. At least $x_i^4/2$ jobs of $\mathscr{J}_i^4$ are scheduled either in the whole second half of $[\frac{5}{2}(i-1)+2,\frac{5}{2}(i-1)+\frac{5}{2})$ or in the first half of $[\frac{5}{2}i+\frac{1}{2},\frac{5}{2}i+1)$.

We will now compute a lower bound on the average increase of the stretch of jobs of type 2 and 3 of size $1/\Delta$ due to the scheduling of jobs of size 1 in interval i. We will first give the following general claim.

Lemma 13. *Assume a set of ℓ jobs of size 1 is scheduled during an interval $[t,t+k/\Delta)$, $k\leqslant\Delta$, while at each time $t_h=t+h/\Delta$, $h=0,\dots,k-1$, m jobs of size $1/\Delta$ are released. The increase of the total stretch of the jobs of size $1/\Delta$ due to the scheduling of the ℓ jobs of size 1 is at least $\ell k^2/2$.*

Proof. The increase in the stretch due to the schedule of the jobs of size 1 is equal to the overall time spent by the jobs of size $1/\Delta$ while not assigned to a machine divided by $1/\Delta$. At each time t_h, $h=0,\dots,k-1$, ℓ additional jobs of size $1/\Delta$ are released and not scheduled. Hence, the overall increase in the stretch due to the ℓ jobs of size 1 is at least

$$\sum_{h=0}^{k-1}\frac{\ell(h+1)1/\Delta}{1/\Delta}=\ell\left(\frac{k^2}{2}+\frac{k}{2}\right). \qquad \square$$

Lemma 14. *Assume ℓ jobs of size 1 are released by time t and scheduled starting at time t or later. Then the overall stretch of these jobs is at least $\ell^2/4m$.*

Proof. We consider two cases:

- $\ell<2m$. In this case, the overall stretch of the jobs is at least ℓ. The thesis then follows by observing that

$$\ell\geqslant\frac{\ell^2}{2m}\geqslant\frac{\ell}{2}\lfloor\ell/m\rfloor.$$

- $\ell\geqslant 2m$. Let $d=\lfloor\ell/m\rfloor$. Then, for $j=1,2,\dots,d$, at least m jobs are in the system at least j time units. As a consequence, the overall stretch of the ℓ jobs is at least

$$m\sum_{j=1}^{d}j=m\frac{d}{2}(d+1)\geqslant\frac{\ell}{2}\lfloor\ell/m\rfloor$$

since $\ell\geqslant m$. This proves the thesis. $\square$

Assume a deterministic on-line algorithm A schedules $x = \sum_i x_i$ jobs of size 1 before time $(\frac{5}{2})s$. Let X_i be a random variable indicating the number of jobs of size 1 that are started in interval i and scheduled during a time interval $[t, t + k/\Delta)$, $k \geqslant \Delta/4$, where m jobs of size $1/\Delta$ are released at each time $t + h/\Delta$, $h = 0, \ldots, k-1$. Let $X = \sum_{i=1}^{s} X_i$.

Lemma 15. $E[X] \geqslant \frac{x}{2}$.

Proof. Consider interval i. Jobs of size 1 started in interval i are partitioned as described above between the sets $\mathcal{J}_i^1, \mathcal{J}_i^2, \mathcal{J}_i^3$ and $\mathcal{J}_i^4$. Observe that all jobs of $\mathcal{J}_i^1$ are scheduled during an interval of size $\frac{1}{2}$ where all jobs of type 2.1 in interval i are released with probability $\frac{1}{2}$. Jobs of $\mathcal{J}_i^2$ are scheduled during an interval of size $\frac{1}{2}$ where all jobs of type 2.2 in interval i are released with probability $\frac{1}{2}$. Jobs of $\mathcal{J}_i^3$ are scheduled during an interval of size $\frac{1}{4}$ where either half of the jobs of type 2.2 in interval i are released with probability $\frac{1}{2}$ or half of the jobs of type 3 in interval i are released. Finally, jobs of set $\mathcal{J}_i^4$ are scheduled during an interval of size $\frac{1}{4}$ where either half of the jobs of type 3 in interval i are released or half of the jobs of type 2.1 in interval $i+1$ are released with probability $\frac{1}{2}$. It then follows $E[X_i] \geqslant (\frac{1}{2})(x_i^1 + x_i^2 + x_i^3 + x_i^4) = (\frac{1}{2})x_i$ and $E[X] = \sum_{i=1}^{s} E[X_i] \geqslant (\frac{1}{2}) \sum_{i=1}^{s} x_i = x/2$. □

Lemmas 15 and 13 imply that the expected value of the overall stretch for jobs that are scheduled before time $(\frac{5}{2})s$ is at least $(\frac{1}{2})\frac{x}{2}(\Delta/4)^2$. On the other hand, $ms - x$ jobs of size 1 are scheduled after time $(\frac{5}{2})s$. Lemma 14 implies that their cost be at least

$$\frac{(ms - x)^2}{4m}.$$

Hence, the expected value of the overall stretch for algorithm A is at least

$$E[A] \geqslant \frac{1}{4} x (\Delta/4)^2 + \frac{(ms - x)^2}{4m}.$$

The above expression is minimized when $x = \frac{m}{2}(2s - \Delta^2/16)$, which yields

$$E[A] \geqslant \frac{\Delta^2 ms}{64} - \frac{\Delta^2 m}{4096}.$$

An off-line algorithm can process all jobs of type 1 within time $(\frac{5}{2})s$. By delaying type 1 jobs at most by 1, all jobs of types 2.1, 2.2 or 3 will undergo no delay. Hence

$$\mathrm{OPT} \leqslant 2ms + \frac{3m\Delta s}{2}.$$

By choosing $s = \Delta^2$ we have $E[A] = \Omega(m\Delta^4)$. On the other side $\mathrm{OPT} = O(m\Delta^3)$. This implies $E[A]/\mathrm{OPT} = \Omega(\Delta)$. □

6. Concluding remarks

We studied the problem of on-line scheduling to minimize average stretch on multiprocessor machines with preemptions, but no migrations. We proved the AALR algorithm proposed in [2] to be $O(1)$ competitive against an optimal, offline adversary who is allowed to migrate jobs. In Section 5, we proved that no non-preemptive on-line randomized algorithm can be better than $\Omega(\Delta)$-competitive. These results show that: (i) migration is not crucial to be competitive on-line; (ii) minimizing on-line average stretch is easier than minimizing average response time, for which an $\Omega(\max\{\log P, \log n/m\})$ lower bound is known [16], where $P = \frac{\max_i p_i}{\min_i p_i}$, and n and m denote the number of jobs and machines, respectively; (iii) differently from migration, preemption is crucial to be competitive.

At the beginning, we briefly described two motivating applications: wireless data servers and distributed server systems. There are many details involved in modeling the precise scheduling problem in these scenarios. In particular, a crucial issue is how to "guess" the processing time needed to service a request. See [1,6,9,21] for more details on the applications. The algorithm in [2] is certainly applicable and simple, hence it will be useful in these applications; our analysis provides insight into why it is desirable in terms of its performance in minimizing the average stretch.

References

[1] S. Acharya, S. Muthukrishnan, G. Sundaram, Scheduling data delivery over multiple channels, Bell Labs Technical Memo, 1999. Available from acharya@research.bell-labs.com.
[2] B. Awerbuch, Y. Azar, S. Leonardi, O. Regev, Minimizing the flow time without migration, in: Proceedings of the ACM Symposium on the Theory of Computing (STOC '99), Atlanta, GA, 1999, pp. 198–205.
[3] K.R. Baker, Introduction to Sequencing and Scheduling, Wiley, New York, 1974.
[4] M. Bender, S. Chakrabarti, S. Muthukrishnan, Flow and stretch metrics for scheduling continuous job streams, in: Proceedings of Ninth ACM-SIAM Annual Symposium on Discrete Algorithms (SODA '98), San Francisco, CA, 1998, pp. 270–279.
[5] A. Borodin, Ran El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, Cambridge, 1998.
[6] M. Crovella, R. Frangioso, M. Harchol-Balter, Connection scheduling in web servers, Proceedings of the Second USENIX Symposium on Internet Technologies and Systems, Boulder, CO, 1999, pp. 243–254.
[7] T.W. Doeppner, Threads, a system for the support of concurrent programming, Technical Report CS-87-11, Brown University, 1987.
[8] J.E. Gehrke, S. Muthukrishnan, R. Rajaraman, A. Shaheen, Scheduling to minimize average stretch online, in: Proceedings of the Annual Symposium on Foundations of Computer Science, New York, NY, 1999, pp. 433–442.
[9] S. Hameed, N. Vaidya, Log-time algorithms for scheduling single and multiple channel data broadcast, Proceedings of the International Conference on Mobile Computing and Networking (MOBICOM), Budapest, Hungary, 1997, pp. 90–99.
[10] M. Harchol-Balter, M. Crovella, C. Murta, On choosing a task assignment policy for a distributed server system. Lecture Notes in Computer Science, Vol. 1469, Springer, Berlin, 1998, pp. 231–242.
[11] Institute for Electrical and Electronic Engineers, POSIX P1003.4a, Threads Extensions for Portable Operating Systems, 1994.
[12] R. Jain, The Art of Computer Systems Performance Analysis, Wiley, New York, 1991.
[13] B. Kalyanasundaram, K. Pruhs, Eliminating migration in multi-processor scheduling, Journal of Algorithms, Special issue for the Ninth ACM-SIAM Symposium on Discrete Algorithms (SODA '99), 38:1, 2–24, 2001.

[14] H. Kellerer, T. Tautenhahn, G.J. Woeginger, Approximability and nonapproximability results for minimizing total flow time on a single machine, Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, 1996, pp. 418–426.
[15] G. Koren, A. Amir, E. Dar, The power of migration in multi-processor scheduling of real time systems, Proceedings of Ninth ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 226–235.
[16] S. Leonardi, D. Raz, Approximating total flow time on parallel machines, in: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 110–119.
[17] M. Mehta, D.J. De Witt, Dynamic memory allocation for multiple-query worloads, in: S. Baker, D. Bell (Eds.), very large data bases, VLDB 1993: Proceedings of the 19th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers, Los Altos, CA, 1993, pp. 354–367.
[18] F. Mueller, A library implementation of POSIX threads under UNIX, in: Proceedings of the Winter 1993 USENIX Technical Conference, San Diego, CA, January 1993, pp. 29–41.
[19] D. Sleator, R.E. Tarjan, Amortized efficiency of list update and paging rules, Comm. ACM 28 (1985) 202–208.
[20] Sun Microsystems, SunOS 5.3 System Services, November 1993.
[21] H. Zhu, B. Smith, T. Yang, Hierarchical resource management for web server clusters with dynamic content, in: Proceedings of SIGMETRICS 1999, 1999, pp. 198–199.

Further reading

L. Becchetti, S. Leonardi, S. Muthukrishnan, Scheduling to minimize average stretch without migration, in: Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA 00), San Francisco, CA, 2000, pp. 548–557.
M. Crovella, M. Harchol-Balter, C. Murta, Task assignment in a distributed system: improving performance by unbalancing load, *Technical Report 1997-018*, Boston University, 1997, URL: http://www.cs.bu.edu/techreports/1997-018-unbalancing-load.ps.Z.
J. Du, J.Y.T. Leung, G.H. Young, Minimizing mean flow time with release time constraint, Theoret. Comput. Sci. 75 (3) (1990) 347–355.
L.A. Hall, Approximation algorithms for scheduling, in: D.S. Hochbaum (Ed.), Approximation Algorithms for NP-Hard Problems, PWS publishing company, Boston, 1997, pp. 1–45.
L. Hall, D. Shmoys, J. Wein, Scheduling to minimize average completion time: off-line and on-line algorithms, in: Proceedings of Seventh ACM-SIAM Symposium on Discrete Algorithms, Atlanta, GA, 1996, pp. 142–151.
E.L. Lawler, J.K. Lenstra, A.H.G.R. Kan, D.B. Shmoys, Sequencing and scheduling: algorithms and complexity, in: Handbooks in Operations Research and Management Science, Vol. 4, North-Holland, Amsterdam, 1993, pp. 445–522.