Competitive distributed scheduling (draft: do not circulate)

Noga Alon [*]

Department of Mathematics

Raymond and Beverly Sackler Faculty of Exact Sciences

Tel Aviv University, Tel Aviv, Israel

Abstract

We consider a distributed scheduling problem in which each of n agents receives a list of tasks which he has to assign to n identical processors based only on the knowledge of the lengths of his own tasks, and without any communication with the other agents. It is known that the best possible competitive ratio of a deterministic algorithm to this problem is $\Theta(\sqrt{n})$. We show that the best possible competitive ratio of a randomized algorithm to this problem is $\Theta(\frac{\log n}{\log\log n})$. This improves an $O(\frac{(\log n)^2}{\log\log n})$ upper bound obtained by Deng and Papadimitriou and provides the first nontrivial lower bound for randomized algorithms to the problem.

1 Introduction

Suppose that each of n agents has an arbitrary number of tasks of arbitrary (integral) lengths, and suppose these tasks should all be scheduled to n identical processors. There is no communication between the agents, and each of them must schedule his own jobs among the processors based only on the local information he has, i.e, the lengths of his jobs. The objective is to minimize the *makespan*, defined as the completion time of the last job (or, equivalently, the maximum total length of jobs assigned to a single processor). We are interested in finding algorithms that optimize the worst case ratio of the makespan of the algorithm divided by the makespan of the best possible algorithm, that works under complete information and may choose the schedule based on the

[*]Research supported in part by a United States Israel BSF Grant

lengths of all the jobs in the system. More precisely, for a deterministic algorithm A for the above problem, and for an input $(I_1, \ldots, I_n)$, where I_i is the list of lengths of the tasks given to agent number i, let $A(I_1, \ldots, I_n)$ be the makespan when all the tasks are assigned to processors according to the algorithm A. Similarly, let $OPT(I_1, \ldots, I_n)$ be the makespan when the tasks are distributed among the processors optimally, i.e., when the tasks of all the lists together are partitioned into n sets, so that the maximum total length of tasks in a single set is minimized. The *competitive ratio of* A, denoted by $C(A)$, is defined as the supremum, over all possible input sequences $(I_1, \ldots, I_n)$ of the ratio

$$\frac{A(I_1, \ldots, I_n)}{OPT(I_1, \ldots, I_n)}. \tag{1}$$

The competitive ratio is defined for randomized algorithms in a similar fashion. If A is such an algorithm and $(I_1, \ldots, I_n)$ is a possible input, let $A(I_1, \ldots, I_n)$ denote the expected value of the makespan when the tasks are assigned according to A, where the expectation is taken over all coin-flips of the algorithm A. The competitive ratio $C(A)$ of A is now, again, the supremum of the ratio (1) under all possible inputs.

The question of designing deterministic and randomized algorithms with good competitive ratios for the above scheduling problem was considered by Deng and Papadimitriou in [2]. It arises naturally in the study of networks in which n processors wish to balance their loads by downloading tasks without exchanging load information. Deng and Papadimitriou proved that the best possible competitive ratio of *deterministic* algorithms for the above problem is $\Theta(\sqrt{n})$, where n is the number of agents and processors, and showed that there are randomized algorithms whose competitive ratio is $O(\frac{(\log n)^2}{\log \log n})$. Here we improve this upper bound by a factor of $\log n$, and show that our estimate is tight, up to a constant factor. This is stated in the following theorem.

Theorem 1.1 *The best possible competitive ratio of a randomized algorithm for the above problem with n agents and n processors is $\Theta(\frac{\log n}{\log \log n})$.*

The upper bound is obtained by analysing the most natural randomized algorithm for the problem; each agent simply distributes his tasks randomly between the processors, where each task is assigned randomly and independently to one of the processors according to a uniform distribution. The analysis is presented in Section 2. The lower bound, described in Section 3, is somewhat more complicated and combines the easy direction of the main result in [4] with some probabilistic and combinatorial ideas.

2 The simplest randomized algorithm is $O(\frac{\log n}{\log\log n})$-competitive

Let A be the randomized algorithm in which each agent simply distributes his tasks randomly between the processors, where each task is assigned randomly and independently to one of the processors according to a uniform distribution. Obviously in this manner all the tasks are distributed randomly, uniformly and independently among the processors. Given a possible input $I = (I_1, \ldots, I_n)$, let M denote the maximum length of a task (in all the lists I_j) and let S denote the total length of all te jobs in all the lists I_j. Two obvious lower bounds for the makespan of the optimal algorithm on the input I are M and S/n. Thus, to complete the proof of the upper bound it suffices to show that the expected makespan of the algorithm A on the input I is at most

$$c\frac{\log n}{\log\log n}Max\{M, S/n\},$$

for some absolute constant c (independent of I). We need the following two simple statements.

Lemma 2.1 *If m balls are distributed randomly, uniformly and independently into n cells, then the probability that there is a cell with at least p balls is at most*

$$n\binom{m}{p}n^{-p}.$$

Therefore, for every positive constant d there is a positive constant $c = c(d)$ such that the probability that some cell receives more than

$$c\frac{\log n}{\log\log n}Max\{1, m/n\}$$

balls is at most n^{-d}.

Proof The probability that a fixed cell receives some fixed set of p balls is precisely n^{-p} and since there are n cells and $\binom{m}{p}$ possibilities to choose p balls out of m the $n\binom{m}{p}n^{-p}$ estimate follows. If $m \leq n$ then

$$n\binom{m}{p}n^{-p} \leq n\frac{1}{p!} \leq n(\frac{e}{p})^p,$$

which is at most n^{-d} for $p \geq c\frac{\log n}{\log\log n}$, for an appropriate $c = c(d)$. For $m \geq n$, if $p = \frac{m}{n}x$ then

$$n\binom{m}{p}n^{-p} \leq n(\frac{em}{p})^p n^{-p} = n(e/x)^p \leq n(e/x)^x.$$

The last number is clearly at most n^{-d} when $x \geq c\frac{\log n}{\log\log n}$, i.,e, when $p \geq c\frac{\log n}{\log\log n}m/n$, completing the proof. □

Corollary 2.2 *Suppose that m balls, each labelled by a number between $L/2$ and L are distributed randomly, uniformly and independently among n cells. Let T denote the sum of all labells of the balls (hence $mL/2 \leq T \leq mL$). Then :*
(i) For every positive d there is a positive $c = c(d)$ such that the probability that there is a cell so that the sum of the labells on the balls it received exceeds

$$c\frac{\log n}{\log\log n}Max\{L, 2T/n\}$$

is at most n^{-d}.
(ii) The expected value of the maximum sum of labells in a cell is at most

$$c\frac{\log n}{\log\log n}Max\{L, 2T/n\},$$

for some absolute constant c.

Proof Part (i) follows immediately from Lemma 2.1. To prove part (ii) let X denote the random variable whose value is the maximum sum of labells in a cell. Observe that always $X \leq T$ and that by part (i) (with, say, $c' = c'(1)$),

$$Prob(X \geq c'\frac{\log n}{\log\log n}Max\{L, 2T/n\}) \leq n^{-1}.$$

Therfore, the expected value of X satisfies

$$E(X) \leq c'\frac{\log n}{\log\log n}Max\{L, 2T/n\} + \frac{1}{n}T \leq c\frac{\log n}{\log\log n}Max\{L, 2T/n\},$$

as needed. □

Returning to the analysis of the performance of the algorithm A on the input I, let us split the set of all tasks the algorithm has to assign into pairwise disjoint sets K_1, K_2, K_3 etc., where the set K_i consists of all the tasks whose length is in the interval $(\frac{M}{2^i}, \frac{M}{2^{i-1}}]$. Let S_i denote the sum of all the lengths of the tasks in K_i. The algorithm A assigns all the tasks randomly, independently and uniformly to the n processors, and consequently this is what it does to the tasks in each set K_i separately. Let X_i be the random variable whose value is the maximum total length of jobs from K_i in a single cell. Let X be the random variable whose value is the maximum total length of tasks in $\cup K_i$ in a single cell. Observe that $X \leq \sum_{i\geq 1} X_i$ and that X is precisely the makespan of the algorithmm A on the input I.

By applying Corollary 2.2 to the set K_i (where the lengths are here the labells of the balls) we conclude that for every $i \geq 1$:

$$E(X_i) \leq c\frac{\log n}{\log\log n} Max\{\frac{M}{2^{i-1}}, 2S_i/n\}.$$

Therefore, by linearity of expectation,

$$E(X) \leq \sum_{i\geq 1} E(X_i) \leq c\frac{\log n}{\log\log n}\sum_{i\geq 1} Max\{\frac{M}{2^{i-1}}, 2S_i/n\}$$

$$\leq c\frac{\log n}{\log\log n}\sum_{i\geq 1}(\frac{M}{2^{i-1}} + 2S_i/n) = c\frac{\log n}{\log\log n}(2M + 2S/n)$$

$$\leq 4c\frac{\log n}{\log\log n} OPT(I).$$

This completes the proof of the upper bound in Theorem 1.1. □

Remark It is not too difficult to modify the algorithm above slightly and obtain a version A' of it which will not only be $O(\frac{\log n}{\log\log n})$-competitive but will in fact have the stronger property that for every input I the probability that $A'(I) > c\frac{\log n}{\log\log n}OPT(I)$ will be at most n^{-d}. This version, considered already in [2] (without the tight analysis given here), is the following; each agent first splits his own tasks into n parts trying to minimize the maximum total length in a part and then assigns the parts randomly and uniformly to the processors. (Although finding the best possible splitting in this respect is NP-complete, the splitting can be done efficiently by any of the known approximation algorithms for this problem, like, e.g., the one in [3] that approximates the optimum by a factor of 2). One can easily check that this process reduces the general case to the case of at most n^2 tasks, and since the estimate for the failure probability in Coroallry 2.2 can be made smaller than n^{-d} for every d we can repeat the proof and get the stronger assertion mentioned above for this version of the algorithm. We omit the details.

3 No randomized algorithm is $o(\frac{\log n}{\log \log n})$-competitive

In this section we prove that no randomized algorithm can perform significantly better than the simple one described in the previous section. We need the following combinatorial lemma (in which we make no attempt to optimize the constants). From now on we assume, whenever this is needed, that n is sufficiently large.

Lemma 3.1 *Let $G = (V, E)$ be a (simple) bipartite graph with vertex classes C and B, where $|C| = n$ and $n/2 \leq |B| \leq n$. Suppose the degree $d(b)$ of every vertex $b \in B$ is at least $\log n$ and let U be a random subset of B obtained by choosing each vertex of B randomy and independently to be a member of U with probability $\frac{1}{(\log n)^2}$. Then, the probability that there is no vertex $c \in C$ that has at least $\frac{1}{10} \frac{\log n}{\log \log n}$ distinct neighbors in U is*

$$e^{-n^{\Omega(1)}} = o(1).$$

Proof Define $x = \lceil \frac{1}{10} \frac{\log n}{\log \log n} \rceil$. We claim that there is a family F of $f \geq \frac{n}{4x}$ edge disjoint subgraphs $H_1, \ldots, H_f$ of G, where each H_i is a star of x edges whose center is in C, and whose end-vertices are in B, so that no vertex of B is an end-vertex of more than one star in F. Indeed, such a family can be constructed greedily. Since there are at least $\frac{n}{2} \log n$ edges in $G = G_1$ the average degree of a vertex in C is larger than x and hence there is a vertex $c_1 \in C$ whose degree is at least x. Let H_1 be an arbitrary star of x edges with a center c_1 and let G_2 be the graph obtained from G_1 by deleting from it all the end-vertices of H_1 (but without deleting the center). Now G_2 has at least $(\frac{n}{2} - x) \log n$ edges and hence there is a vertex c_2 in C whose degree in G_2 is at least x. (Note that c_1 may be equal to c_2). Let H_2 be a star of x edges in G_2 with center c_2 and let G_3 be the graph obtained from G_2 by deleting all the end-vertices of H_2. This process can obviously be repeated for at least $n/4x$, since after $i < n/4x$ stars have been defined the total number of edges in G_i is still at least $(\frac{n}{2} - ix) \log n \geq \frac{n}{4} \log n$ and hence the average degree of a vertex in C is still at least $\log n/4 > x$ and the process can continue. Therefore there are f stars with the required properties, as claimed.

Consider, now, the randomly chosen set U. For each fixed star H_i in F, let E_i be the event that all the end-vertices of H_i lie in U. Clearly,

$$Prob(E_i) = (\frac{1}{(\log n)^2})^x = \frac{1}{n^{0.2+o(1)}} \geq \frac{1}{n^{0.3}}.$$

Moreover, the events E_i are *mutually independent*, since no two stars share a common end-vertex. It follows that the probability that none of the events E_i occurs is at most

$$(1-\frac{1}{n^{0.3}})^f \leq e^{-n^{0.7}/4x} = o(1).$$

However, if E_i occurs then c_i has at least x neighbors in U, completing the proof. $\Box$

We can now prove that no randmized algorithm for the scheduling problem with n agents and n processors is, say, $x/4$ competitive, where $x = \frac{1}{10}\frac{\log n}{\log\log n}$. To do so we exhibit a probability distribution on the set of possible inputs I and show that the expected ratio of the makespan of every *deterministic* algorithm A for the problem divided by the makespan of the optimum (where the expectation is computed over this distribution) is at least $x/4$. By the easy direction of the argument of Yao ([4]) this implies the desired result. The probability distribution we consider is the following; we choose each agent, randomly and independently, with probability $\frac{1}{(\log n)^2}$, and if it is chosen we let its list consist of $\lfloor(\log n)^2\rfloor$ tasks of length 1 each (the agents that are not chosen do not get any tasks). Let S denote the sum of lengths of all the tasks in the system (which is here the number of tasks). By the standard estimates for Binomial distributions (cf. e.g., [1]) $Prob(S > 2n) < e^{-n^{\Omega(1)}} = o(1)$ and since on every input I of the above form the makespan of the optimum is precisely $\lceil S/n\rceil$ this means that with extremely high probability $OPT(I) \leq 2$.

Next we show that for every deterministic algorithm A for the problem the probability that for the random input I chosen according to our distribution $A(I) \geq x$ is $1 - o(1)$. To see this, let N be the set of all agents that assign, according to the algorithm A, at least x tasks to the same processor if they get a list of $\lfloor(\log n)^2\rfloor$ unit jobs. Consider two possible cases.

Case 1: $|N| \geq n/2$. In this case, almost surely at least one of the members of N received in the input I $\lfloor(\log n)^2\rfloor$ unit jobs, and in this case he will assign at least x of them to the same processor, hence making the makespan at least x, as needed.

Case 2: $|N| \leq n/2$. In this case, there are at least $n/2$ agents (the ones not in N) so that each of them, when receiving a list of $\lfloor(\log n)^2\rfloor$ unit jobs never assigns x of them to a single processor, and hence assigns some tasks to at least $\lfloor(\log n)^2\rfloor/x \geq \log n$ processors. Let G be the bipartite graph whose classes of vertices are C and B, where C is the set of n processors and B is the set of all agents not in N. An agent $b \in B$ is adjacent to a processor $c \in C$ if b assigns one or more tasks to c according to the algorithm A if he gets a list of $\lfloor(\log n)^2\rfloor$ unit jobs. The degree of every vertex

$b \in B$ in this graph is at least $\log n$. Therfore, we can apply Lemma 3.1 and conclude that when the random input I is chosen as above then almost certainly some processor will receive tasks from at least x distinct agents, and hence almost certainly the makespan will be at least x.

We have thus shown that for every deterministic algorithm A for the problem, if inputs I are chosen according to the above probability distribution then almost certainly $OPT(I) \leq 2$ while $A(I) \geq x$ and hence almost certainly the ratio $A(I)/OPT(I)$ is at least $x/2$. Since this ratio is always positive (in fact, it is always at least 1) this clearly implies that the average, over the distribution of the inputs I, of this ratio, is at least $(1 - o(1)x/2 > x/4$. This completes the proof of the lower bound. $\square$

References

[1] N. Alon and J. H. Spencer, *The Probabilistic Method*, Wiley, 1991.

[2] X. Deng and C. H. Papadimitriou, *Competitive distributed decision-making*, to appear.

[3] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell Systems Technical Journal 45 (1966), 1563-1581.

[4] A. C. C. Yao, *Probabilistic computation: towards a unified measure of complexity*, Proc. 18th Annual IEEE FOCS, Providence, RI (1977), pp. 222-227.