Improved Scheduling in Rings

Dekel Tsur *

Deptartment of Computer Science and Engineering, University of California, San Diego, CA 92093-0114, USA

Abstract

We study the problem of scheduling unit size jobs on n processors connected by a ring. We show a distributed algorithm for this problem with an approximation ratio of $\frac{3}{2} + \sqrt{2}$.

Key words: Scheduling, Approximation Algorithms

1 Introduction

The problem of scheduling a set of jobs on parallel machines is a well studied problem in computer science. In this paper we study the following *distributed* model, which was introduced in [4]: There are n processors with communication links between some of the processors. Initially, each processor has several unit size jobs (the number of jobs at each processor is specified by an input I to the problem). At each time step, a processor can process one job from its queue, and send some of its jobs to its neighbors. Moreover, each processor can send additional information (e.g. its initial load) to its neighbors in each time step. The goal is to process all the jobs in minimum time.

Since initially the processors do not have information on the other processors, it is impossible to give an algorithm that achieves the the optimal schedules for all inputs. We say that an algorithm A for the problem has an approximation ratio r, if there is a constant c such that for every input I, $A(I) \leq r \cdot \mathrm{OPT}(I) + c$, where $A(I)$ denotes the completion time (also called makespan) of algorithm A on the input I, and $\mathrm{OPT}(I)$ denotes the completion time of the optimal schedule for the input I.

* Fax: 1 858 534 7029.
Email address: dtsur@cs.ucsd.edu (Dekel Tsur).

Preprint submitted to Elsevier Science

Awerbuch et al. [3] gave an algorithm for the problem whose approximation ratio is $O(\log n)$. A lower bound of $\Omega(\log n/\log\log n)$ on the approximation ratio of any algorithm was given by Alon et al. [1]. For some special cases of the problem, better approximation algorithms exist. When the processors are connected by a ring, the algorithm of Awerbuch et al. has a constant approximation ratio, for some large constant. Fizzano et al. [5] presented an algorithm for rings with an approximation ratio of 4.22. They also gave a lower bound of 1.06 on the approximation ratio of any algorithm for rings. In this paper, we give an algorithm for rings with an approximation ratio of $\frac{3}{2}+\sqrt{2}\approx 2.91$.

Related Work Deng et al. [4] gave an optimal *centralized* algorithm for scheduling on arbitrary graphs, whose running time is polynomial. They also gave an $O(\log n)$-approximation centralized algorithm for the case when the jobs have weights. Phillips et al. [6] gave a 2-approximation algorithm for the latter problem.

Awerbuch et al. [3] studied the online version of the scheduling problem. In this problem, jobs arrive to the system at different times, and the goal is to minimize the maximum response time (i.e. the maximum time a job waits before it is processed). Awerbuch et al. gave an algorithm for this problem with a competitive ratio of $O(\log^2 n)$ for general graphs. An $\Omega(\log n)$ lower bound for the online problem was given by Alon et al. [1]. Anagnostopoulos et al. [2] showed a scheduling algorithm which is stable against an adversary.

2 Algorithm for the fractional problem

We first give some definitions. Denote the processors by $1,\ldots,n$, where processor i is connected to processor $i+1$ for all i (throughout the paper, processor numbers are assumed to be modulo n). An input to the scheduling problem will be denoted by $I=\{x_1,\ldots,x_n\}$ when x_i is the initial load of processor i. The *clockwise* direction for processor i is the direction of processor $i+1$. The *load* on processor i at time t is the number of jobs that are at processor i at the beginning of step t. The *completion time* of processor i for an input I is the last step in which the load of i is nonzero.

We first consider a variant of the scheduling problem, called the *fractional problem*, in which a processor can split a job into fractional parts, and each part can either be sent to other processors or processed (at every time step, the total amount of jobs processed in every processor is at most 1). We will show an approximation algorithm for the fractional problem, and later we will show

how to obtain an approximation algorithm for the original problem (which is called the *integral problem*).

Let algorithm A_1 be as follows:

For each j, the x_j jobs at processor j are moved into a bucket, denoted B_j. In each step, the bucket B_j moves clockwise along the ring. When the bucket B_j reaches processor i, it leaves

$$\min(k, c\sqrt{x_j + \cdots + x_i} - c\sqrt{x_{j+1} + \cdots + x_i})$$

jobs at processor i, where k is the current number of jobs in the bucket and c is a constant to be determined later. In particular, at the first step, the bucket B_j leaves $\min(x_j, c\sqrt{x_j})$ jobs at processor j.

Additionally, each processor i sends messages containing the value of x_i in both directions of the ring. If the algorithm has not finished within $\lfloor n/2 \rfloor$ steps, then at the beginning of step $\lfloor n/2 \rfloor + 1$, each processor knows the values of $x_1, \dots, x_n$. Then, each processor checks whether there is some bucket B_i that would contain jobs after traversing the entire ring. If there is such a bucket, then the buckets process is stopped, and the remaining jobs are evenly distributed among the n processors.

We note that algorithm A_1 is very similar to the algorithm of Fizzano et al. [5]. However, there are few differences that are crucial to our improved analysis.

Theorem 1 *The approximation ratio of algorithm A_1 is at most $\frac{3}{2} + \sqrt{2}$.*

PROOF. Consider some input $\hat{I} = \{\hat{x_1}, \dots, \hat{x}_n\}$, and let $L = \mathrm{OPT}(\hat{I})$. Our goal is to show that $A(\hat{I}) \leq (\frac{3}{2} + \sqrt{2})L + O(1)$. We use the following two lemmas from [5]:

Lemma 2 *For every j and k, $\hat{x}_j + \cdots + \hat{x}_{j+k-1} \leq L(L + k - 1)$.*

Lemma 3 *The maximum distance a bucket can travel before it becomes empty is at most $(2/c + 1/c^2)L$.*

We note that Lemma 3 is proved in [5] for the algorithm presented in that paper, but it is easy to verify that the lemma also holds for our algorithm.

Suppose first that no bucket would contain jobs after n steps. Due to the symmetry of the algorithm, it suffices to bound the completion time of every processor for which B_1 is the last bucket that leaves jobs at the processor. Let r be such a processor. To simplify the analysis, we define algorithm A_2 to be an algorithm that acts like algorithm A_1, except that the buckets $B_2, \dots, B_r$ have an infinite supply of jobs until they leave processor r. In other words, the bucket

B_i leaves exactly $c\sqrt{x_i + \cdots + x_j} - c\sqrt{x_{i+1} + \cdots x_i}$ jobs at processor j, for $i = 2, \ldots, r$ and $j = i, \ldots, r$. Moreover, if the bucket B_1 is not empty when it arrives to processor r, then it leaves exactly $c\sqrt{x_1 + \cdots + x_i} - c\sqrt{x_2 + \cdots + x_i}$ jobs at processor r. Finally, the jobs at processors $r+1, \ldots, n$ are not moved into buckets. Clearly, algorithm A_2 is not a valid algorithm for the problem. However, it is easy to verify that on the input $\hat{I}$, the completion time of processor r in algorithm A_1 is less than or equal to its completion time in algorithm A_2. Therefore, in the following we will bound the completion time of processor r in algorithm A_2, which will also bound the completion time in algorithm A_1.

An input $I = \{x_1, \ldots, x_n\}$ will be called an L-*valid* input if (1) When algorithm A_2 runs on I, the bucket B_1 is not empty when it arrives to processor r, and (2) $x_j + \cdots + x_{j+k-1} \leq L(L + k - 1)$ for all j and k.

To bound the completion time of processor r on the input $\hat{I}$, we will look for an L-valid input that maximizes the completion time of processor r. For this end, we give several lemmas which show simple transformations on the input which do not decrease the completion time of processor r, except perhaps by a constant.

Claim 4 *Let I and I' be two inputs such that*

(1) There is a set $S \subseteq \{1, \ldots, r-1\}$ such that for every $t \in \{1, \ldots, r-1\} \setminus S$, the total amount of jobs left by buckets $B_{r-t+1}, \ldots, B_r$ in processor r in the run on I is ***greater*** *than or equal to the corresponding amount in the run on I'.*

(2) The total amount of jobs left by buckets $B_1, \ldots, B_r$ in processor r in the run on I is less than or equal to the corresponding amount in the run on I'.

Then, the completion time of processor r on the input I is less than or equal to its completion time on the input I' plus $|S|$.

For the following lemmas, let $I = \{x_1, \ldots, x_n\}$ be some L-valid input.

Lemma 5 *If $x_{i+1} > 0$ for some $2 \leq i < r$, then the completion time of processor r on I is less than or equal to its completion time on $I' = \{x_1, \ldots, x_i + 1, x_{i+1} - 1, \ldots, x_n\}$.*

PROOF. The difference between the inputs I and I' has two effects on the run of algorithm A_2: First, the buckets B_i and B_{i+1} leave different amount of jobs at processor r in the runs on I and I' (note that every other bucket from $B_2, \ldots, B_r$ leaves the same amount of jobs at processor r). More precisely, when algorithm A_2 runs on I, B_{i+1} leaves $a = c\sqrt{x_{i+1} + \cdots + x_r} -$

$c\sqrt{x_{i+2}+\cdots+x_r}$ jobs at processor r, and B_i leaves $b = c\sqrt{x_i+\cdots+x_r} - c\sqrt{x_{i+1}+\cdots+x_r}$ jobs. When the algorithm runs on I', B_{i+1} and B_i leave $a' = c\sqrt{x_{i+1}+\cdots+x_r-1} - c\sqrt{x_{i+2}+\cdots+x_r}$ and $b' = c\sqrt{x_i+\cdots+x_r} - c\sqrt{x_{i+1}+\cdots+x_r-1}$ jobs at processor r, respectively. Note that $a+b = a'+b'$ and $a' < a$, so the first condition of Claim 4 is satisfied with $S = \phi$.

The second difference is that bucket B_1 leaves different amount of jobs at processors $1,\ldots,r-1$: In the run on I, B_1 leaves $a_i = c\sqrt{x_1+\cdots+x_i} - c\sqrt{x_2+\cdots+x_i}$ jobs at processor i (for every $i < r$), and in the run on I', B_1 leaves $c\sqrt{x_1+\cdots+x_i+1} - c\sqrt{x_2+\cdots+x_i+1} < a_i$ jobs at processor i. It follows that in the run on I', bucket B_1 always has more jobs than in the run on I. In particular, since B_1 is not empty when it arrives to processor r in the run on I, it is also not empty when it arrives to processor r in the run on I'. By the definition of algorithm A_2, B_1 leaves the same number of jobs at processor r in the runs on I and I'. Thus, the second condition of Claim 4 is satisfied and the lemma follows. □

Lemma 6 *The completion time of processor r on I is less than or equal to its completion time on $I' = \{x_1+1, x_2, \ldots, x_n\}$.*

PROOF. Let

$$f(x_1,\ldots,x_i) = \frac{c(\sqrt{x_1+\cdots+x_i} - \sqrt{x_2+\cdots+x_i})}{x_1} = \frac{c}{\sqrt{x_1+\cdots+x_i} + \sqrt{x_2+\cdots+x_i}}$$

be the fraction of the x_1 jobs initially in B_1 that bucket B_1 leaves at processor i. Since $f(x_1+1, x_2, \ldots, x_i) < f(x_1,\ldots,x_i)$ for all i, and since B_1 is not empty when it arrives to processor r in the run on I, it follows that B_1 is not empty when it arrives to r in the run on I'. Therefore, in the run on I the bucket B_1 leaves $a = c(\sqrt{x_1+\cdots+x_r} - \sqrt{x_2+\cdots+x_r})$ jobs at processor r, while in the run on I it leaves $c(\sqrt{x_1+1+\cdots+x_i} - \sqrt{x_2+\cdots+x_i}) > a$ jobs at processor r. The buckets $B_2,\ldots,B_r$ leave the same amount of jobs in processor r in the runs on I and I'. Therefore, the lemma follows from Claim 4. □

Lemma 7 *For every $0 \le a \le x_1$, the completion time of processor r on I is at most the completion time of processor r on $I' = \{x_1-a, x_2+a, \ldots, x_n\}$ plus one.*

PROOF. Consider the definition of f from the previous lemma. As $f(x_1-a, x_2+a, \ldots, x_i) < f(x_1,\ldots,x_i)$ for all i, we have that B_1 is not empty when it arrives to processor r in I', and therefore the total number of jobs that is left by the buckets $B_1,\ldots,B_r$ at processor r in the run on I' is $c\sqrt{x_1+\cdots+x_r}$, which is equal to the amount of jobs left at processor r in the run on I. Therefore, the conditions of Claim 4 are satisfied with $S = \{r-1\}$. $\Box$

Lemma 8 *Suppose that processor r is not idle between step $r-i+1$ (i.e., after B_i arrives to r) and its completion time in the run on I. Then, for every $a \geq 0$, the completion time of processor r on I is at most its completion time on $I' = \{x_1, \ldots, x_i + a, \ldots, x_n\}$.*

PROOF. In the run on I', the bucket B_1 leaves $\sqrt{x_1 + \cdots + x_j + a} - \sqrt{x_2 + \cdots + x_j + a} < \sqrt{x_1 + \cdots + x_j} - \sqrt{x_2 + \cdots + x_j}$ jobs on processor j, for every $j \geq i$. Therefore, B_1 is not empty when it arrives to processor r, and the total amount of jobs left at processor r is $\sqrt{x_1 + \cdots + x_r + a}$, which is larger then the total amount of jobs left at processor r in the run on I. $\Box$

Note that in Lemmas 5–8, the input I' satisfies the first requirement in the definition of L-valid input. For the input $\hat{I}$, we assume w.l.o.g. that $\hat{x}_i = 0$ for $i = r+1, \ldots, n$. We repeatedly apply the transformation of Lemma 5 for every index i for which the input after the transformation is L-valid. Then, we apply the transformation of Lemma 6 or Lemma 7 in order to make x_1 equal to L. The new input $I = \{x_1, \ldots, x_n\}$ satisfies either

$$x_1, \ldots, x_n = L, L^2, \overbrace{L, \ldots, L}^{s-1 \text{ times}}, b, 0, \ldots, 0$$

where $b \leq L$ and $s \leq r-2$, or

$$x_1, \ldots, x_n = L, b, 0, \ldots, 0$$

where $b < L^2$. Then, we apply the transformation of Lemma 8 with $i = s+2$ and $a = L - b$ in the first case, and with $i = 2$ and $a = L^2 - b$ in the second case. We will later show that the conditions of Lemma 8 are satisfied. In both cases, the new input $I' = \{x'_1, \ldots, x'_n\}$ satisfies

$$x'_1, \ldots, x'_n = L, L^2, \overbrace{L, \ldots, L}^{s \text{ times}}, 0 \ldots, 0.$$

The input I' is L-valid. Therefore, when algorithm A_2 runs on I', the bucket B_1 leaves $c(\sqrt{L^2 + (i-1)L} - \sqrt{L^2 + (i-2)L})$ jobs at processor i for $i = 2, \ldots, s+2$, and

$$c\left(\sqrt{L^2 + (s+1)L} - \sqrt{L^2 + sL}\right) = \frac{cL}{\sqrt{L^2 + (s+1)L} + \sqrt{L^2 + sL}}$$
$$\geq \frac{cL}{2\sqrt{L^2 + (s+1)L}}$$

jobs for $i = s+3, \ldots, r-1$. Thus, the number of jobs bucket B_1 leaves at processor $2, \ldots, r-1$ is at least

$$\sum_{i=2}^{s+2} c\left(\sqrt{L^2+(i-1)L} - \sqrt{L^2+(i-2)L}\right) + \frac{(r-s-3)cL}{2\sqrt{L^2+(s+1)L}}$$
$$= c\sqrt{L^2+(s+1)L} - c\sqrt{L^2} + \frac{(r-s-3)cL}{2\sqrt{L^2+(s+1)L}}.$$

Clearly, this number is at most the number of jobs that are initially in B_1, which is $x'_1 = L$. Setting $\alpha = (s+1)/L$ and $\beta = (r-s-3)/L$, we obtain that

$$\sqrt{1+\alpha} - 1 + \frac{\beta}{2\sqrt{1+\alpha}} \le \frac{1}{c}.$$

This yields that $\beta \le \frac{1}{2}(1+1/c)^2$ and $\sqrt{1+\alpha} \le z(c,\beta)$, where

$$z(c,\beta) = \frac{1+\frac{1}{c}+\sqrt{\left(1+\frac{1}{c}\right)^2 - 2\beta}}{2}.$$

The total number of jobs that the buckets $B_1, \ldots, B_r$ leave at processor r is $c\sqrt{L^2+(s+1)L} = cL\sqrt{1+\alpha}$. Therefore, the completion time of processor r on the input I' is at most

$$r-s-2+cL\sqrt{1+\alpha} = 1+\beta L + cL\sqrt{1+\alpha} \le 1 + (\beta + cz(c,\beta))L.$$

By Lemmas 5–8, $A_1(\hat{I}) \le A_2(I) + 1 \le 2 + (\beta + cz(c,\beta))L$. Therefore,

$$\max_I \frac{A_1(I)-2}{\mathrm{OPT}(I)} \le \max_{\beta \le \frac{1}{2}(1+1/c)^2} (\beta + cz(c,\beta)).$$

Using simple calculus, we obtain that $\max_I((A_1(I)-2)/\mathrm{OPT}(I)) \le c^2/8 + c/2 + 1 + 1/c + 1/2c^2$. We minimize the last expression by choosing $c = \sqrt{2}$, and obtain that $\max_I((A_1(I)-2)/\mathrm{OPT}(I)) \le \frac{3}{2} + \sqrt{2}$.

We now show that the conditions of Lemma 8 are satisfied, namely, that processor r is not idle from step $r-s-1$ until its completion time on the input I. We first show that processor r is not idle from step $r-s-1$ to step $t-2$. For $i = 3, \ldots, s$, bucket B_i leaves less jobs in processor r than bucket B_{i+1}. Therefore, it suffices to show that the average number of jobs that is left by a bucket from $B_3, \ldots, B_{s+2}$ is at least 1. That is, we need to show that $c\sqrt{b+(s-1)L} \ge s$. This inequality is indeed satisfied since we have from Lemma 3 that $s+2 \le r \le (\sqrt{2}+\frac{1}{2})L$. Moreover, we have that the amount of jobs left by B_2 and $B_3, \ldots, B_{s+2}$ at processor r is $c\sqrt{b+(s-1+L)L} \ge s+2$, and therefore processor r is not idle from step $r-s-1$ until its completion.

Now, consider the case when there is a bucket that would not become empty after n steps. After the first $\lfloor n/2 \rfloor$ steps, the remaining jobs are evenly distributed among the processors in $\lfloor n/2 \rfloor$ steps, and then processed in at most $\lceil \frac{1}{n} \sum_{i=1}^{n} x_i \rceil$ steps. Clearly, $L = \mathrm{OPT}(I) \geq \lceil \frac{1}{n} \sum_{i=1}^{n} x_i \rceil$, so $A_1(I) \leq 2\lfloor n/2 \rfloor + \lceil \frac{1}{n} \sum_{i=1}^{n} x_i \rceil \leq n + L$. By Lemma 3, $n \leq (2/c + 1/c^2)L = (\sqrt{2} + \frac{1}{2})L$. Therefore, $A_1(I) \leq (\frac{3}{2} + \sqrt{2})L$. □

3 Algorithm for the integral problem

In the previous section we gave an approximation algorithm for the fractional problem. In this section, we will show an approximation algorithm, denoted A_1', for the integral problem. Recall that algorithm A_1 consists of two stages. In the first stage, the processors send jobs only clockwise. Algorithm A_1' will simulate the first stage of algorithm A_1 with a constant loss of performance. In the second stage of algorithm A_1' the remaining jobs are evenly distributed among the processors.

Theorem 9 *For every algorithm A for the fractional problem in which jobs are sent only clockwise, there is an algorithm A' for the integral problem such that $A'(I) \leq A(I) + 2$ for every integral input I.*

PROOF. Consider some input $I = \{x_1, \ldots, x_n\}$. Let $p_i(t)$ be the total amount of jobs processes by processor i during the first t steps of algorithm A, and let $s_i(t)$ be the total amount of jobs sent by processor i to processor $i+1$ during the first t steps. For the algorithm A' which will be defined shortly, we define $p_i'(t)$ and $s_i'(t)$ in a similar manner.

Algorithm A' is as follows: At step $t \leq A(I)$, each processor i computes the values of $p_i(t)$ and $s_i(t)$. Then, processor i processes one job if it has available jobs and $\lfloor p_i(t) \rfloor - p_i'(t-1) \geq 1$. Finally, processor i sends $\min(k, \lfloor s_i(t) \rfloor - s_i'(t-1))$ jobs to processor $i+1$, where k is the number of jobs in processor i at time t. At each step $t \geq A(I)+1$, each processor processes one of remaining jobs.

Lemma 10 *For every i and $t \leq A(I)$, $p_i'(t) = \lfloor p_i(t) \rfloor$ and $s_i'(t) = \lfloor s_i(t) \rfloor$.*

PROOF. We prove the lemma using induction on t. The base $t = 0$ is trivial. Suppose that we have proved the lemma for $t-1$, and consider some processor i at step t.

Let $p = p_i(t) - p_i(t-1)$ be the amount of jobs that is processed by processor i during step t of algorithm A. Clearly, p is less than or equal to the amount of jobs at processor i at the beginning of step t, namely

$$p_i(t) - p_i(t-1) = p \leq x_i + s_{i-1}(t-1) - s_i(t-1) - p_i(t-1).$$

From the induction hypothesis we have that $s'_i(t-1) = \lfloor s_i(t-1) \rfloor \leq s_i(t-1)$, so

$$p_i(t) \leq x_i + s_{i-1}(t-1) - s_i(t-1) \leq x_i + s_{i-1}(t-1) - s'_i(t-1).$$

Since x_i and $s'_i(t-1)$ are integers, and $s'_{i-1}(t-1) = \lfloor s_{i-1}(t-1) \rfloor$ by the induction hypothesis, we obtain that

$$\begin{aligned}\lfloor p_i(t) \rfloor &\leq \lfloor x_i + s_{i-1}(t-1) - s'_i(t-1) \rfloor = x_i + \lfloor s_{i-1}(t-1) \rfloor - s'_i(t-1) \\ &= x_i + s'_{i-1}(t-1) - s'_i(t-1).\end{aligned}$$

Therefore,

$$\lfloor p_i(t) \rfloor - p'_i(t-1) \leq x_i + s'_{i-1}(t-1) - s'_i(t-1) - p'_i(t-1).$$

The right side of the inequality above is the number of jobs in processor i at the beginning of step t of algorithm A'. Moreover, $\lfloor p_i(t) \rfloor - p'_i(t-1) = \lfloor p_i(t) \rfloor - \lfloor p_i(t-1) \rfloor \in \{0, 1\}$. It follows that in algorithm A', processor i processes $\lfloor p_i(t) \rfloor - p'_i(t-1)$ jobs during step t. Hence, $p'_i(t) = \lfloor p_i(t) \rfloor$.

Showing that $s'_i(t) = \lfloor s_i(t) \rfloor$ is done in a similar manner: Clearly,

$$s_i(t) - s_i(t-1) \leq x_i + s_{i-1}(t-1) - s_i(t-1) - p_i(t),$$

so

$$s_i(t) \leq x_i + s_{i-1}(t-1) - p_i(t) \leq x_i + s_{i-1}(t-1) - p'_i(t).$$

Thus,

$$\lfloor s_i(t) \rfloor \leq x_i + \lfloor s_{i-1}(t-1) \rfloor - p'_i(t) = x_i + s'_{i-1}(t-1) - p'_i(t).$$

Hence

$$\lfloor s_i(t) \rfloor - s'_i(t-1) \leq x_i + s'_{i-1}(t-1) - s'_i(t-1) - p'_i(t).$$

The right side of the last inequality is the number of jobs at processor i at step t (after possibly assigning a job to be processed). Therefore, in algorithm A', processor i sends$\lfloor s_i(t) \rfloor - s'_i(t-1)$ jobs during step t, and $s'_i(t) = \lfloor s_i(t) \rfloor$. □

Let $l_i(t)$ (resp., $l'_i(t)$) be the load on processor i at the beginning of step t of algorithm A (resp., A'). We have that for every i and $t \leq A(I) + 1$.

$$\begin{aligned} l'_i(t) &= x_i + s'_{i-1}(t-1) - s'_i(t-1) - p'_i(t-1) \\ &= x_i + \lfloor s_{i-1}(t-1) \rfloor - \lfloor s_i(t-1) \rfloor - \lfloor p_i(t-1) \rfloor \\ &\leq x_i + s_{i-1}(t-1) - s_i(t-1) - p_i(t-1) + 2 = l_i(t) + 2. \end{aligned}$$

In particular, $l'_i(A(I)+1) \leq 2$. It follows that $A'(I) \leq A(I) + 2$. $\square$

From Theorem 1 and Theorem 9 we obtain that the approximation ratio of algorithm A'_1 is at most $\frac{3}{2}+\sqrt{2}$ (the analysis of the second stage of algorithm A'_1 is identical to the analysis of the second stage of algorithm A_1).

Acknowledgements

We thank Yossi Azar for helpful discussions.

References

[1] N. Alon, g. Kalai, M. Ricklin, and L. J. Stockmeyer. Lower bounds on the competitive ratio for mobile user tracking and distributed job scheduling. *Theoretical Computer Science*, 130(1):175–201, 1994.

[2] A. Anagnostopoulos, A. Kirsch, and E. Upfal. Load balancing in arbitrary network topologies with stochastic adversarial input. *SIAM J. on Computing*, 34(3):616–639, 2005.

[3] B. Awerbuch, S. Kutten, and D. Peleg. Competetive distributed job scheduling. In *Proc. 24th Symposium on the Theory of Computing (STOC 92)*, pages 571–580, 1992.

[4] X. Deng, H. Liu, J. Long, and B. Xiao. Competitive analysis of network load balancing. *J. of Parallel and Distributed Computing*, 40(2):162–172, 1997.

[5] P. Fizzano, D. R. Karger, C. Stein, and J. Wein. Distributed job scheduling in rings. *J. of Parallel and Distributed Computing*, 45(2):122–133, 1997.

[6] C. A. Phillips, C. Stein, and J. Wein. Task scheduling in networks. *SIAM J. Discrete Math.*, 10(4):573–598, 1997.