Acta Cybernetica **15** (2001) 163-172.

An online scheduling algorithm for a two-layer multiprocessor architecture *

Cs. Imreh†

Abstract

In this paper we give online algorithms and competitive ratio bounds for a scheduling problem on the following two-layer architecture. The architecture consists of two sets of processors; within each set the processors are identical while both the processors themselves and their numbers may differ between the sets. The scheduler has to make an online assigment of jobs to one of the two processor sets. Jobs, assigned to a processor set, are then sceduled in an optimal offline preemptive way within the processor set considered. The scheduler's task is to minimize the maximum of the two makespans of the processor sets.

1 Introduction

In the most fundamental parallel machine scheduling model, we have a sequence of jobs, each of them has a processing time, and we have to process them on the available uniform machines. A schedule specifies for each job a machine and a time interval on the machine when the job is processed on it. The length of the time interval must be the processing time, the starting and ending points of the time interval are called the *starting and finishing time* of the job. A schedule is *feasible* if for each machine the time intervals do not overlap. Our goal usually is to minimize the maximal finishing time. Sometimes it is allowed to preempt the jobs. In this case, we have to specify for each job a sequence of machines with not overlapping time intervals (one machine can have more time intervals), and the total length of the time intervals must be the processing time. For more details on scheduling problems we refer to [6].

The most fundamental example of an online machine scheduling problem is the online problem of jobs arriving one by one. In this problem we have a fixed number m of identical machines. The jobs and their processing times are revealed to the online algorithm one by one. When a job is revealed, the online algorithm must irrevocably assign the job to a machine, without any information about the further

*This work has been supported by the Grant OTKA T030074

†Department of Informatics, University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary e-mail: cimreh@inf.u-szeged.hu

jobs. The starting time of the job is the finishing time of the previous job assigned to the machine considered. By the load of a machine, we mean the sum of the processing times of all jobs assigned to the machine. Then, the maximal completion time is the maximum load, thus the objective is to minimize the maximum load, often called the *makespan*. The first result for this online scheduling model is due to Graham [4], the best algorithm which is known for this problem can be found in [1], but the best possible competitive ratio is still unknown for $m \geq 3$. For details and results on other online scheduling problems we refer to the survey [8].

In this paper we consider a scheduling problem where the machines form a two-layer multiprocessor architecture. In this problem we have two sets $\mathcal{P}$ and $\mathcal{S}$ of identical machines containing k and m machines with $k \leq m$. The jobs arrive one by one. Each job j has two different processing times p_j and s_j, one for each set of machines. We assign the jobs online to one of the two machine sets. Finally, when the stream of jobs has come to an end, we schedule the jobs assigned to $\mathcal{P}$ (respectively, the jobs assigned to $\mathcal{S}$) on the machines of $\mathcal{P}$ (respectively, $\mathcal{S}$) in an offline way so as to minimize the preemptive makespan. Let C_P (respectively, C_S) denote this optimal makespan. The cost of the schedule, which we want to minimize, is the maximum of the makespans, $\max\{C_P, C_S\}$. We denote this problem by $LS(k,m)$ (*Layered Scheduling*). The general problem without fixing the numbers k and m is denoted by LS. It is worth noting that the $LS(1,1)$ problem is the online two-machine scheduling problem with unrelated machines which problem is investigated in [2].

A similar problem called classification with preemptive scheduling (or CPS for short) is studied in [5]. Just as in the LS problem, in CPS an online scheduling to one of two sets of identical processors is followed by an offline preemptive scheduling within the processor sets. However in CPS the objective to be minimized is the sum of the two makespans instead of their maximum. For the CPS problem two algorithms are developed the first one is a greedy algorithm. The competitive ratio of this greedy algorithm is linear in m/k, therefore this algorithm can be effective only in the cases when m/k is small. A more difficult algorithm with constant competitive ratio is also presented.

In this paper we study the algorithms which are presented in [5], and we determine their competitive ratios for the LS problem. Moreover, we prove a general lower bound, namely, we show that no online algorithm can have smaller competitive ratio than 1.781.

The paper is organized as follows. In the following section we introduce some basic notation for the LS problem. Then, in Section 3, we present two algorithms for solving the problem and determine the competitive ratios of these algorithms. In Section 4 we present two lower bounds for LS. First, we construct input sequences for each fixed pair of values (k,m) to show that no online algorithm for the fixed (k,m) can be better than $(1+\sqrt{5})/2$-competitive. Then we construct a single input sequence to show that for all online algorithms there exists a pair (k,m) such that the algorithm cannot be better than $(3+\sqrt{17})/4$-competitive. Finally, we summarize the results and present some open questions regarding the problem in Section 5.

2 Notation and preliminaries

In the problem considered we have two sets of machines, $\mathcal{P}$ is the set of k identical machines and $\mathcal{S}$ is the set of m identical machines. In what follows we assume that $k \leq m$. Furthermore, each job j has two processing times, p_j if we schedule it on the machines of $\mathcal{P}$ and s_j if we schedule it on the machines of $\mathcal{S}$. Here we allow ∞ processing time, which means that it is not possible to process the job on the machines in the set. The vector (p_j, s_j) is called the *size* of the job. The algorithm has to decide in an online fashion on the arrival of each job where to assign it. When the sequence of jobs is finished, we schedule the jobs assigned to $\mathcal{P}$ respectively $\mathcal{S}$ on the machines of $\mathcal{P}$ respectively $\mathcal{S}$, in the way that we minimize the makespan preemptively in an offline way.

With preemption a job may be scheduled on multiple machines. In preemptive scheduling we have to assign time intervals to each job on one or more machines. The total length of the intervals must be the processing time of the job, and if time intervals $(q_1, t_1), \ldots, (q_i, t_i)$ are assigned to a job, then $t_j \leq q_{j+1}$ must be valid for $j = 1, \ldots, i-1$. Furthermore, no two jobs may have overlapping intervals on the same machine. It is well-known and one can easily see that for any set of jobs, the preemptive makespan of the optimal scheduling is the maximum of the maximal processing time and the load of the jobs. By the *load of a set of jobs* we mean the value obtained by dividing the sum of the processing times by the number of machines.

For any subset I of the jobs, we use the following notation

$$S_I = \sum_{j \in I} s_j, \quad P_I = \sum_{j \in I} p_j, \quad \text{Smax}_I = \max_{j \in I} s_j, \quad \text{Pmax}_I = \max_{j \in I} p_j.$$

Using these notation, the cost of a schedule SC can be written in the form

$$w(SC) = \max\{\frac{P_R}{k}, \text{Pmax}_R, \frac{S_Q}{m}, \text{Smax}_Q\},$$

where R and Q are the sets of jobs assigned to $\mathcal{P}$ and $\mathcal{S}$, respectively.

The optimal cost on a list L of jobs is denoted by $OPT(L)$. This is the minimum of the costs of the schedules which assign the jobs of L to the sets. We measure the performance of the presented algorithms by the competitive analysis. For this reason let A be an arbitrary online algorithm and let $A(L)$ denote the cost of the schedule produced by A on a list L of jobs. An algorithm A is called *c-competitive* if for every list L of jobs $A(L)$ is at most c times greater than OPT(L). The *competitive ratio* of an algorithm on a problem is c if the algorithm is c-competitive and it is not $\bar{c}$-competitive for any $\bar{c} < c$.

3 Upper bounds

There is a simple online algorithm for LS. The basic idea is to assign each job to the set of machines where the job has a smaller load. This algorithm is called load

greedy, (in short LG) and it is presented in [5] for the problem CPS. It has a similar analysis in our case. Formally, the algorithm is given as follows:

Algorithm *LG*: *When a job j arrives, then it is assigned to $\mathcal{P}$ if $\frac{p_j}{k} \leq \frac{s_j}{m}$, otherwise, it is assigned to $\mathcal{S}$.*

We can prove the following statement.

Theorem 1 *Algorithm LG has the competitive ratio* $\max\{2, m/k\}$ *on problem* $LS(k,m)$.

Proof. Consider an arbitrary list L of jobs, and denote by a and b the makespans obtained by LG on $\mathcal{P}$ and $\mathcal{S}$, respectively. Suppose that $a \geq b$. If the makespan is defined on $\mathcal{P}$ by the maximal completion time, then denote the job with this maximal processing time by j. Then $p_j = a$, and since our algorithm assigns this job to $\mathcal{P}$, we get that, $s_j > \frac{m}{k}a$. Hence, the optimal cost is at least a, and this yields that we have an optimal solution. Now, suppose that the makespan is defined by the load of the jobs. Let P denote the set of the jobs assigned by LG to $\mathcal{P}$, and let R and Q denote the sets of the jobs from P which are assigned to $\mathcal{P}$ and $\mathcal{S}$ in an optimal solution, respectively. Then, the optimal cost is at least $\max\{\frac{P_R}{k}, \frac{S_Q}{m}\}$. Furthermore, by the definition of LG, we get that $\frac{P_Q}{k} \leq \frac{S_Q}{m}$. This yields that the cost of the optimal solution is at least $\max\{\frac{P_R}{k}, \frac{P_Q}{k}\} \geq a/2$, and our statement follows.

Let us assume that $a < b$. Now, consider two cases depending on the makespan on $\mathcal{S}$. If the makespan is the load, then in the same way as above, we obtain that the optimal cost is at least $b/2$, which yields that the algorithm is 2-competitive. Suppose that the makespan is defined by a maximal processing time. Denote the job with this maximal processing time by j. Then, $s_j = b$ and since our algorithm assigns this job to $\mathcal{S}$, we obtain that $p_j > \frac{k}{m}b$. Hence, the optimal cost is at least $\frac{k}{m}b$, and our statement follows.

To prove that the above analysis is tight, consider one job of size $(1, \frac{m}{k} - \varepsilon)$. Algorithm *LG* assigns this job to $\mathcal{S}$ with cost $\frac{m}{k} - \varepsilon$, hence, since the optimal cost is 1, by choosing a sufficiently small ε, the competitive ratio on this job is arbitrarily close to $\frac{m}{k}$. To prove that the bound 2 is tight, we have to consider a sequence of jobs where the load of each job is the same in the two sets.

□

Algorithm LG works well only in the cases when m/k is small. Here we study an algorithm which is also efficient when m/k is large. This algorithm is defined in [5] and can be considered as a generalization of the reject total penalty type algorithms which are presented in [3] and [7]. The algorithm has two parameters $0 < \alpha \leq 1$ and $0 < \gamma \leq 1$.

Algorithm $A(\alpha, \gamma)$

- 1. *Initialization.* Let $R := \emptyset$.
- 2. When job j arrives:
  - (i) If $\frac{p_j}{k} \leq \frac{\gamma}{m} \cdot s_j$, then assign j to $\mathcal{P}$.
  - (ii) Let r be the cost of the optimal offline preemptive scheduling of the set $R \cup \{j\}$ on $\mathcal{P}$. Formally, $r = \max\{\frac{P_{R\cup\{j\}}}{k}, \text{Pmax}_{R\cup\{j\}}\}$. If $r \leq \alpha \cdot s_j$, then
    - (a) Assign j to $\mathcal{P}$,
    - (b) Set $R = R \cup \{j\}$.
  - (iii) Otherwise, assign j to $\mathcal{S}$.

Theorem 2 *The competitive ratio of algorithm $A(\alpha, \gamma)$ is c on problem LS, where*

$$c = \max\{1 + \frac{1}{\alpha}, 1 + \alpha + \gamma, 1 + \frac{1}{\gamma}\}.$$

Proof. We prove this statement in two parts. First, we show that the algorithm is c-competitive, and later we prove that it is not better than c-competitive. Let us consider an arbitrary sequence of jobs, and denote the list of the jobs by L. Fix an optimal schedule of the jobs. Denote by P_{opt} the set of jobs assigned to $\mathcal{P}$ in the optimal schedule. Let P_0 be the set of jobs with $\frac{p_j}{k} \leq \frac{\gamma}{m} \cdot s_j$, and P be the set of jobs assigned to $\mathcal{P}$ by our algorithm. Let us observe that, by the definition of the algorithm, we have $P_0 \subseteq P$. Define the following sets

$$X = L \setminus (P_{opt} \cup P), \quad Y = P_{opt} \setminus P,$$

$$Z = P_{opt} \cap (P \setminus P_0), \quad U = P_{opt} \cap P_0,$$

$$V = P_0 \setminus P_{opt}, \quad W = (P \setminus P_0) \setminus P_{opt}.$$

Then, the algorithm gives the following cost on L:

$$A(L) = \max\{\frac{P_Z + P_U + P_V + P_W}{k}, \text{Pmax}_Z, \text{Pmax}_U,$$

$$\text{Pmax}_V, \text{Pmax}_W, \frac{S_X + S_Y}{m}, \text{Smax}_X, \text{Smax}_Y\}.$$

Furthermore, the optimal cost is

$$OPT(L) = \max\{\frac{P_Y + P_Z + P_U}{k}, \text{Pmax}_Y \text{Pmax}_Z,$$

$$\mathrm{Pmax}_U, \frac{S_X + S_V + S_W}{m}, \mathrm{Smax}_X, \mathrm{Smax}_V, \mathrm{Smax}_W\}.$$

To prove the first part of the theorem, we have to show that $A(L) \leq c \cdot OPT(L)$. In the proof we will use the following lemma which is proved in [5]. For the reader's convenience, we recall here this result and also sketch the proof of the statement.

Lemma 3 *([5]) The following inequalities are valid:*

$$(1) \quad \frac{P_W}{k} \leq \alpha \cdot \mathrm{Smax}_W,$$

$$(2) \quad \mathrm{Pmax}_W \leq \alpha \cdot \mathrm{Smax}_W,$$

$$(3) \quad \alpha \cdot \mathrm{Smax}_Y \leq \max\{\frac{P_Z + P_W + P_Y}{k}, \mathrm{Pmax}_{Z \cup W \cup Y}\}.$$

Proof. We first prove (1). Let j be the last job from W. At the time when it was assigned to $\mathcal{P}$ by the algorithm, we had $r \leq \alpha \cdot s_j$. On the other hand, j is the last job in W, thus $\frac{P_W}{k} \leq r$. Furthermore, obviously $s_j \leq \mathrm{Smax}_W$, and the validity of (1) follows. We can prove (2) in the same way as (1). Indeed, let j be a job from W with $p_j = \mathrm{Pmax}_W$. When it was assigned to $\mathcal{P}$, we had $p_j \leq r \leq \alpha \cdot s_j$. This inequality yields (2).

There exists a job $j \in Y$ with $s_j = \mathrm{Smax}_Y$. At the time when it was assigned to $\mathcal{S}$, we had $r > \alpha s_j$. On the other hand, $R \cup \{j\} \subseteq Z \cup W \cup Y$ was valid for the considered set R, thus $r \leq \max\{\frac{P_Z + P_W + P_Y}{k}, \mathrm{Pmax}_{Z \cup W \cup Y}\}$ was also valid by definition. Therefore, the required inequality holds.

□

Using this lemma, we can prove the desired upper bound. For this reason consider the following cases.

Case 1: Suppose that $A(L) = \max\{\mathrm{Pmax}_Z, \mathrm{Pmax}_U, \mathrm{Smax}_X\}$. In this case $A(L) \leq OPT(L)$, and thus, the algorithm results in an optimal solution.

Case 2: Suppose that $A(L) = \frac{P_Z + P_U + P_V + P_W}{k}$. In this case the definition of the set V yields that $\frac{P_V}{k} \leq \frac{\gamma}{m} \cdot S_V \leq \gamma OPT(L)$. On the other hand $\frac{P_Z + P_U}{k} \leq OPT(L)$. Furthermore, by Lemma 3, we have that $P_W/k \leq \alpha\ \mathrm{Smax}_W \leq \alpha OPT(L)$. Therefore, $A(L) \leq (1 + \alpha + \gamma)OPT(L)$.

Case 3: Suppose that $A(L) = \mathrm{Pmax}_V$. Then, by the definition of the set V, $\mathrm{Pmax}_V/k \leq \gamma\ \mathrm{Smax}_V/m$. Since $k \leq m$, this yields that $\mathrm{Pmax}_V \leq \gamma\ \mathrm{Smax}_V \leq \gamma OPT(L)$. This is possible only when $\gamma = 1$, and in this case the algorithm results in an optimal solution.

Case 4: Suppose $A(L) = \mathrm{Pmax}_W$. Then by Lemma 3, $\mathrm{Pmax}_W \leq \alpha\ \mathrm{Smax}_W \leq \alpha \cdot OPT(L)$, therefore, this case is possible only if $\alpha = 1$, and the algorithm gives an optimal solution.

Case 5: Suppose that $A(L) = \frac{S_X+S_Y}{m}$. In this case, by the definition of the set Y we have that $\frac{S_Y}{m} \leq \frac{P_Y}{\gamma k} \leq OPT(L)/\gamma$. Hence, we obtain that $A(L) \leq (1+1/\gamma)OPT(L)$.

Case 6: Suppose that $A(L) = \text{Smax}_Y$. By Lemma 3 this yields that

$$A(L) \leq \frac{1}{\alpha}\max\{\frac{P_Z+P_W+P_Y}{k}, \text{Pmax}_{Z\cup W\cup Y}\}.$$

Consider now three subcases. If $A(L) \leq \frac{P_Z+P_W+P_Y}{\alpha k}$, then since $\frac{P_W}{\alpha k} \leq \text{Smax}_W \leq OPT(L)$ (by Lemma 3), we get that $A(L) \leq (1+1/\alpha)OPT(L)$. If $A(L) \leq 1/\alpha\ \text{Pmax}_{Z\cup Y}$, then we obtain immediately that $A(L) \leq OPT(L)/\alpha$. Finally, if $A(L) \leq 1/\alpha\ \text{Pmax}_W$, then by Case 4, we have that $A(L) \leq OPT(L)$.

Since we considered all the possible cases, we proved that the algorithm is c-competitive. We can prove that the bound c is tight by the following examples.

First, assume that $k = 1$ and $m > \gamma \cdot (1+\alpha)/\alpha$. Consider the following sequence of two jobs. The first job has size $(\alpha \cdot M, M)$, the second job has size $(M+\varepsilon, M(1+\alpha)/\alpha)$ for some large M and small ε. By the definition of m, we have that $\alpha M > \gamma M/m$, and thus, the first job is assigned to $\mathcal{P}$ in Step (ii). The second job is assigned to $\mathcal{S}$. Therefore, the cost of the algorithm is $M(1+\alpha)/\alpha$ on this sequence. The optimal cost is $M+\varepsilon$, we assign the first job to $\mathcal{S}$ and the second to $\mathcal{P}$. As M tends to ∞ the ratio of these costs tends to $1+1/\alpha$, hence we proved that the first bound is tight.

To prove the tightness of the second bound, fix the value of k and let m be much greater than k. Consider the following sequence of jobs. First consider $M(m-k)$ jobs of size $(\gamma \cdot k/m, 1)$. The second part of the sequence contains k jobs of size (M, ∞), finally, the third part contains k jobs of size $(\alpha \cdot M, M)$. Then the first and second parts are assigned to $\mathcal{P}$ in Step (i), the third part is also assigned to $\mathcal{P}$ in Step (i) or in Step (ii). Therefore, the cost of the algorithm is

$$\frac{M(m-k)\gamma k/m + Mk + \alpha Mk}{k}.$$

The optimal solution assigns the first and the third parts to $\mathcal{S}$, and the second part to $\mathcal{P}$ and its cost is M. As m tends to ∞, the ratio of these costs tends to $1+\alpha+\gamma$, hence, we proved that the second bound is tight.

To prove that the third bound is tight, consider such k and m that satisfy the inequality $\alpha/k > \gamma/m$ and the following sequence of jobs. The first part of the sequence is one job of size $(\alpha(m/(\gamma k)+2\varepsilon), (m/(\gamma k)+2\varepsilon))$. The second part contains Mk jobs of size $(1, m/(\gamma k)+\varepsilon)$, and the third part contains m jobs of size (∞, M). Then the algorithm assigns the first job to $\mathcal{P}$ in Step (ii), and assigns the other jobs to $\mathcal{S}$. Therefore, its cost is

$$\frac{Mk(m/(\gamma k)+\varepsilon) + mM}{m}.$$

There is a feasible schedule which assigns the second part of the jobs to $\mathcal{P}$ and the third part to $\mathcal{S}$, therefore, the optimal cost is no more than $M + m/(\gamma k) + 2\varepsilon$. As M tends to ∞ and ε tends to 0, the ratio of these costs tends to $1 + 1/\gamma$, hence, we proved that the third bound is tight.

□

To find the best values of α, γ, we have to choose $\alpha = \gamma = 1/\sqrt{2}$. By substituting these values into Theorem 2, we obtain the following result.

Corollary 4 *Algorithm $A(1/\sqrt{2}, 1/\sqrt{2})$ is $1 + \sqrt{2}$-competitive on LS.*

4 Lower bounds

In this section we present the following lower bounds.

Theorem 5 *Let (k, m) be an arbitrary pair of positive integers. If an online algorithm is c-competitive for the $LS(k, m)$ problem, then $c \geq (1 + \sqrt{5})/2 \approx 1.618$.*

Proof. We prove this statement by contradiction. Suppose there is a pair (k, m) and an algorithm A which is $c < (1 + \sqrt{5})/2$-competitive for the problem $LS(k, m)$. Consider the following list L of jobs. The first part contains k jobs of size $((\sqrt{5} - 1)/2, 1)$, and the next part contains k jobs of size $(1, \infty)$. In this case, the optimal offline algorithm assigns the first k jobs to $\mathcal{S}$, and the next k jobs to $\mathcal{P}$, hence $OPT(L) = 1$. On the other hand, since A is c-competitive it must assign the first k jobs to $\mathcal{P}$, otherwise, we omit the next k jobs, and the offline optimum is $(\sqrt{5} - 1)/2$, while the cost of the algorithm is 1. Therefore, the online algorithm must assign all jobs to $\mathcal{P}$, and thus, it has a makespan $(1 + \sqrt{5})/2$. This yields that $A(L)/OPT(L) = (1 + \sqrt{5})/2$, which is a contradiction.

□

We can obtain a sharper, general lower bound as follows.

Theorem 6 *Let k be a fixed constant. If an online algorithm is c-competitive for every m on the problem $LS(k, m)$, then $c \geq (3 + \sqrt{17})/4 \approx 1.781$*

Proof. We prove this statement by contradiction. Suppose that there exist such k and an online algorithm A, that A is c-competitive for every m on the problem $LS(m, k)$, where $c < (3 + \sqrt{17})/4$. For the sake of simplicity, in the rest of the proof we denote the number $(3 + \sqrt{17})/4$ by b. Let m be greater than $5k$ and consider the following sequence of jobs. The first part contains k jobs of size $(1/b, 1)$, and the following $m - k$ jobs have size $(\frac{1}{m-k}, 1)$. Finally, depending on the decisions made by A, we finish the sequence with k jobs of size $(1, \infty)$, this is list L_1, or we finish the list with $m - k$ jobs of size $(\infty, 1)$, this is list L_2.

Consider first the offline optimum. In the first case we can assign the first m jobs to $\mathcal{S}$ and the last k jobs to $\mathcal{P}$, this schedule has cost 1. In the second case,

we can assign the first k jobs and the last $m-k$ jobs to $\mathcal{S}$, and the other $m-k$ jobs to $\mathcal{P}$, and thus, we can obtain a makespan of 1. Therefore, $OPT(L_1) \leq 1$, and $OPT(L_2) \leq 1$.

Consider the algorithm A. Since A is an online algorithm, it cannot see any difference between L_1 and L_2 before it gets the $m+1$-th job, and thus, it has the same behaviour in both cases. Furthermore, A is c-competitive with $c < b$, therefore it must assign the first k jobs to $\mathcal{P}$, otherwise we get a contradiction in the same way as in the proof of Theorem 5. From the next $m-k$ jobs A can assign x to $\mathcal{P}$ and $m-k-x$ to $\mathcal{S}$. Therefore, in the case of list L_1 we have that $A(L_1) \geq \frac{1}{b} + \frac{x}{m-k} + 1$ and in the case of list L_2 we get that $A(L_2) \geq \frac{2m-2k-x}{m}$. The algorithm can choose x to be any integer between 1 and $m-k$, and we can choose the list which yields the greater makespan. Therefore, since the offline optimum is at most 1 for both lists and A is c-competitive, we have that

$$c \geq \min_{1 \leq x \leq m-k} \max\{\frac{1}{b} + \frac{x}{m-k} + 1, \frac{2m-2k-x}{m}\}.$$

Here we omitted the condition that x is an integer. This does not cause any problem since it decreases the right side of the inequality. It can easily be seen that the function of x, which is on the right side of the inequality, is minimal for

$$x = \frac{m(m-k)}{2m-k}(\frac{m-2k}{m} - \frac{1}{b}).$$

If we substitute this value into the bound for c, we obtain that

$$c \geq \frac{1}{b} + \frac{3m-3k}{2m-k} - \frac{m}{(2m-k)b}.$$

Since this inequality is valid for arbitrary m, it is also valid if we let m to tend to infinity. Therefore,

$$c \geq \frac{3}{2} + \frac{1}{2b}.$$

On the other hand, $b = \frac{3}{2} + \frac{1}{2b}$, and thus, we obtain that $c \geq b$ which is a contradiction.

□

5 Conclusions

In this paper we investigated a particular scheduling problem on a two-layer multiprocessor architecture. We showed that the greedy approach works well only in the cases where the layers contain around the same number of machines. We also presented a better algorithm for the general case, which has a constant competitive ratio for arbitrary number of machines. It was also proved that there exists no online algorithm with smaller competitive ratio than 1.781.

In relation with the problem considered, some further questions arise.

Concerning the *LS* problem there is a gap between our lower and upper bounds. It would be nice to decrease this gap by finding more efficient algorithms or better lower bounds.

In the problem considered the objective is the maximum of the two makespans, which function is the l_∞ norm of the vector constructed from the makespans. In [5] the l_1 norm is investigated. It would be also interesting to study other l_p norms.

We considered the problem with two sets of machines. A straightforward generalization is to consider a problem with more sets of machines.

References

[1] S. Albers, Better Bounds for Online Scheduling, *SIAM Journal of Computing*, 29 (1999) 459–473.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts, On-line load balancing with applications to machine scheduling and virtual circuit routing, *J. of the ACM*, 44 (1997) 486–504.

[3] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall and L. Stougie, Multiprocessor scheduling with rejection, *SIAM Journal of Discrete Mathematics*, 13 (2000) 64–78.

[4] R. L. Graham, Bounds for certain multiprocessor anomalies, *Bell System Technical Journal*, 45 (1966) 1563–1581.

[5] Cs. Imreh, Online classification with offline scheduling, *Algoritmica*, submitted for publication.

[6] S. A. Roosta, *Parallel Processing and Parallel Algorithms*, Springer-Verlag, New York, 1999.

[7] S. S. Seiden, Preemptive Multiprocessor Scheduling with Rejection, *Theoret. Comput. Sci.*, to appear.

[8] Sgall J. On-line scheduling, In *Online algorithms: The State of the Art*, Lecture Notes in Computer Science, Vol. 1442, G. Woeginger and A. Fiat (Eds.), Springer-Verlag, 1998, 196–231.