
Journal of Scheduling **7:** 421–428, 2004.

THE ON-LINE MULTIPROCESSOR SCHEDULING PROBLEM WITH KNOWN SUM OF THE TASKS

E. ANGELELLI[1], Á.B. NAGY[2], M.G. SPERANZA[1], AND ZS. TUZA[3]

[1] *Department of Quantitative Methods, University of Brescia, C.da S. Chiara 50, I-25122 Brescia, Italy*
[2] *Department of Computer Science, University of Veszprém, Hungary. Research supported in part by the Hungarian Scientific Research Fund, grant OTKA T029309*
[3] *Comp. and Autom. Inst., Hungarian Academy of Sciences, Budapest, Hungary and Department of Computer Science, University of Veszprém, Hungary.* *Research supported in part by the Hungarian Scientific Research Fund, grant OTKA T032969*

ABSTRACT

In this paper we investigate a semi on-line multiprocessor scheduling problem. The problem is the classical on-line multiprocessor problem where the total sum of the tasks is known in advance. We show an asymptotic lower bound on the performance ratio of any algorithm (as the number of processors gets large), and present an algorithm which has performance ratio at most $\frac{\sqrt{6}+1}{2} < 1.725$ for any number of processors. When compared with known general lower bounds, this result indicates that the information on the sum of tasks substantially improves the performance ratio of on-line algorithms.

KEY WORDS: on-line scheduling, parallel processors, competitive analysis

1. INTRODUCTION

In the classical multiprocessor scheduling problem a set of tasks with given processing times is available and has to be assigned to a set of identical processors with the objective of minimizing the makespan, i.e. the maximum completion time on the processors. As the problem is NP-hard, exact algorithms, heuristic and approximation algorithms are known. The performance of an approximation algorithm is measured through the worst-case ratio between the value of the makespan of the algorithm and the minimum makespan.

In the on-line multiprocessor scheduling problem the tasks and their processing times are not known in advance and, when a task becomes available, it has to be immediately assigned to one of the processors before the next task becomes available, with the objective of minimizing the makespan at the end of the instance. No information is known on the tasks not yet available. An on-line algorithm assigns the incoming task to one of the already partially loaded processors until the end of the instance. The performance of an online algorithm is measured through the worst-case ratio between the value of the makespan of the algorithm and the minimum makespan, which might be obtained if all the tasks were known in advance, as in the classical multiprocessor scheduling problem. An additional information on the performance of an on-line algorithm is a lower bound on the performance of any on-line algorithm. When the lower bound coincides with the algorithm performance, the algorithm can be considered optimal. The currently best general upper bound for the on-line multiprocessor scheduling problem is due to Fleischer and Wahl (2000)

Correspondence to: E. Angelelli. E-mail: angele@eco.unibs.it. M.G. Speranza. E-mail: speranza@eco.unibs.it

who present an algorithm with a competitive ratio which tends to about 1.9201 as the number of processors tends to infinity. On the other hand, Albers has proved in Albers (1999) that no general algorithm, for $m \geq 80$, can perform better than 1.852. Faigle, Kern, and Turán (1989) proved that no algorithm can have a competitive ratio better than 1.707 for any $m \geq 4$.

The assumptions of pure on-line problems are often too pessimistic with respect to real problems where in many cases partial information is available and can be exploited by appropriate algorithms. We consider the on-line multiprocessor scheduling problem under the assumption that the sum of the processing times of the tasks is known in advance. No additional information is known on the tasks not yet available. As in the on-line multiprocessor scheduling problem an incoming task has to be immediately assigned to one of the processors with the objective of minimizing the makespan at the end of the instance. The problem has been studied, for the case of two processors, by Kellerer et al. (1997) where an optimal algorithm with performance 4/3 has been presented. For the case of three processors, an algorithm with performance $1 + \frac{8}{19} < 1.4211$ has been introduced in Angelelli, Speranza, and Tuza (submitted) together with a lower bound 1.3929 on the performance of any algorithm.

The problem attacked in this paper is in some sense similar to the one studied by Azar and Regev (2001) where the authors study an on-line bin-stretching problem which is equivalent to minimizing the makespan in a multiprocessor scheduling problem under the assumption that the off-line optimum is known in advance. Thanks to this strong assumption they obtain an algorithm with competitive ratio equal to 1.625.

Variants of the on-line multiprocessor scheduling problem with known sum of the processing times have also been studied for the case of two processors. In Angelelli (2000) an optimal algorithm for the problem has been obtained for the case where a lower bound on the processing times of the tasks is also known. The case where an upper bound, instead of a lower bound, is known on the processing times of the tasks has been investigated in Angelelli, Speranza, and Tuza (2003). He and Zhang (1999) studied the problem on two identical processor where the sum of the tasks is not given in advance, but both a lower and an upper bound on the size of the tasks are known. They proved that in this case the List Scheduling algorithm is optimal.

The aim of this paper is to study the performance of algorithms for the on-line multiprocessor scheduling problem with known sum of the processing times, when the number of processors is large. We show an asymptotic, with respect to the number of processors, lower bound on the performance of any algorithm and present an algorithm which has a performance $\frac{\sqrt{6}+1}{2} < 1.725$ for any number of processors. This performance considerably improves the 1.923 of the best known algorithm for the pure on-line multiprocessor problem and the performance of any algorithm for the on-line problem in the case of a large number of processors, as 1.852 is known to be a lower bound for all $m \geq 80$. This means that the knowledge of the sum of the processing times of the tasks, which is a simple aggregated information, is sufficient to substantially improve the performance of the on-line algorithms. For the sake of completeness we mention that a $\frac{5}{3}$-competitive algorithm is claimed by Girlich, Kotov, and Kovalev (1998) in a technical report dated 1998, where no lower bound is proposed. However, the proof in the original version (already submitted in 1998) is incomplete, and to our best knowledge, no complete version has been prepared since then. Thus, the $\frac{5}{3}$-competitiveness appears to be a conjecture and our paper contains the best upper bound proved so far.

In Section 2 the asymptotic lower bound is presented while Section 3 is devoted to the description and analysis of the on-line algorithm. Some conclusions are finally sketched.

2. PROBLEM DEFINITION AND AN ASYMPTOTIC LOWER BOUND

A set $\mathcal{M} = \{1, \ldots, m\}$ of m identical processors is available for the processing of the tasks. The tasks are labeled with natural numbers, in the order of arrival, and we denote by t_i the processing time of task i. Given an online algorithm H, the makespan of the algorithm is denoted by T, while the value of the optimal off-line makespan is denoted by $\tilde{T}(\tilde{T} \leq T)$. The load of processor j at any iteration of the on-line algorithm is denoted by P_j. In order to simplify numbers occurring in the proofs, we assume without loss of generality, that $\sum_i t_i = 6m$.

The performance of an on-line algorithm is measured by the *competitive ratio.* An on-line algorithm H for a minimization problem is said to be r-competitive if the inequality $T \leq r \cdot \tilde{T}$ holds for any instance. The competitive ratio R_H is defined as inf $\{r \mid H$ is r-competitive$\}$. An algorithm H is said to be *optimal* if no other algorithm has a better competitive ratio.

2.1. An asymptotic lower bound

Theorem 1. No algorithm can have a performance ratio asymptotically better than 1.565 *as* $m \to \infty$.

Proof. We can view the proof as a game between two players, an algorithm H and an adversary A. Player A sends out one task at a time and player H has to assign that task to a processor. Player H aims at minimizing the makespan, while player A tries to maximize it. We discuss a strategy for player A and show how, and to which extent, it prevents any player H from obtaining the optimal assignment.

Player A chooses some $x \in (0.5,1)$ and $q \in (x, 1)$, then define $\varepsilon = \frac{2x+3q}{m-2}$. Note that x and q are bounded quantities, thus, $\varepsilon \to 0$ when $m \to \infty$. In other words, ε can be made arbitrarily small if an adequately large number m of processors is taken.

When the game begins, player A first sends out $(m-2)$ tasks $p_1, p_2, \ldots, p_{m-2}$ of size $6(1 - \varepsilon)$ followed by 2 tasks p_{m-1}, p_m of size $6x$.

If player H assigns a pair of tasks to the same processor, we have basically two cases:

(a) p_1 and p_i for some $i \epsilon (2, m)$, are assigned to the same processor. Then player A sends out two tasks p_{m+1}, p_{m+2} of size $6(1 — x)$ and $(m — 2)$ tasks of size 6ε to complete the instance. In this case $T \geq 6(1 - \varepsilon) + 6x$, while $\tilde{T} = 6$. Thus, $T/\tilde{T} \geq 1 + x - \varepsilon$.

(b) p_{m-1} and p_m are assigned to the same processor. Then player A sends out two big tasks p_{m+1}, p_{m+2} of size $6(1 + q)$ and a set of small tasks to complete the instance. In this case at least one of the two big tasks has to be assigned to the same processor as p_1 and $T = 6(1 - \varepsilon) + 6(1 + q)$, while $\tilde{T} = 6(1 + q)$. Thus, $T/\tilde{T} = \frac{2+q-\varepsilon}{1+q} = 1 + \frac{1-\varepsilon}{1+q}$.

Otherwise H assigns the first m tasks to m distinct processors. Then player A sends out a task p_{m+1} of size $6q$.

If H assigns task p_{m+1} to the same processor as p_1 , then player A sends out a task p_{m+2} of size $6(1 - q)$ and a set of small tasks to complete the instance. In this case $T = 6(1 - \varepsilon) + 6q$, while $\tilde{T} = 12x$. Thus, $T/\tilde{T} = 1 + \frac{1+q-2x-\varepsilon}{2x}$.

Otherwise H assigns task p_{m+1} to the same processor as p_m. Then player A completes the instance with two big tasks $6(1 + q)$, $6(1 + q)$. In this case $T = 6(1 - \varepsilon) + 6(1 + q)$, while $\tilde{T} = 6(1 + q)$. Thus, $T/\tilde{T} = \frac{2+q-\varepsilon}{1+q} = 1 + \frac{1-\varepsilon}{1+q}$.

We have shown that no algorithm can guarantee a performance ratio better than $1 + \min(x - \varepsilon, \frac{1-\varepsilon}{1+q}, \frac{1+q-2x-\varepsilon}{2x})$. Since we are interested in the asymptotic behavior of the lower bound for m large, we consider ε as negligible and focus our attention on the bound $1 + \min(x, \frac{1}{1+q}, \frac{1+q-2x}{2x})$.

Now, let us choose $x = \frac{1}{1+q}$ and $x \leq \frac{1+q-2x}{2x}$ in order to keep the lower bound as large as possible.

From the latter conditions we obtain $q = \frac{1-x}{x}$ and $x \leq \frac{1-2x^2}{2x^2}$, which holds for $x \leq \frac{1}{6}(46 + 6\sqrt{57})^{\frac{1}{3}} + \frac{2}{3}(46 + 6\sqrt{57})^{-\frac{1}{3}} - \frac{1}{3}$.

In conclusion, given a fixed $\tilde{x} = \frac{1}{6}(46 + 6\sqrt{57})^{\frac{1}{3}} + \frac{2}{3}(46 + 6\sqrt{57})^{-\frac{1}{3}} - \frac{1}{3} > 0.565$ and a fixed $\tilde{q} = \frac{1-\tilde{x}}{\tilde{x}}$, the performance ratio of any algorithm is not better than $1 + \tilde{x} + O(\frac{1}{m})$. ■

3. AN ALGORITHM FOR ANY NUMBER OF PROCESSORS

In this section we prove that there exists a c-competitive on-line algorithm for any number of processors, with

$$c = \frac{\sqrt{6}+1}{2} < 1.725.$$

Observe that this c is just

$$c = \frac{5}{3}(1 + \delta)$$

where δ is the positive root of the equation

$$\frac{5}{3} \cdot 8 \cdot (1 + \delta)^2 = 14 + 8\delta.$$

Given a problem instance with $\sum_i t_i = 6m$, we define as *small tasks* all the tasks with $0 < t_i < 4(1 + \delta)$, and as *big tasks* all the tasks with $4(1 + \delta) \leq t_i$.

Theorem 2. There exists a $\frac{\sqrt{6}+1}{2}$*-competitive algorithm, for any number of processors.*

We prove this theorem through the following steps. First, we describe an on-line algorithm $\bar{H}$. Then, we prove the following stronger statement, for the case in which the number of big tasks does not exceed the number of processors:

Theorem 3. Under the assumption that there are at most m big tasks, the algorithm $\bar{H}$ *satisfies*

$$T \leq \frac{5}{3} \cdot 6 \cdot (1 + \delta) \leq c\tilde{T} \tag{1}$$

for all problem instances.

Finally, we will prove that the algorithm $\bar{H}$ has competitive ratio not worse than c also on instances where more than m big tasks occur.

3.1. The algorithm $\bar{H}$

Let task i be the incoming task, that is the task to be scheduled. Let us define the following sets:
$\mathcal{K} = \{j \in \mathcal{M}: P_j + t_i \leq 10(1 + \delta)$, a big task has already been assigned to processor $j\}$

$$\mathcal{F} = \{j \in \mathcal{M}: P_j + t_i \leq 4(1 + \delta)\}$$

The algorithm $\bar{H}$ is defined as follows.

1. The task i is small
   (a) If $\mathcal{K} \neq \emptyset$, then
   $k = \text{argmax}_{j \in \mathcal{K}}(P_j + t_i)$
   (b) If $\mathcal{K} = \emptyset$, and $\mathcal{F} \neq \emptyset$, then
   $k = \text{argmax}_{j \in \mathcal{F}}(P_j + t_i)$
   (c) Else
   $k = \text{argmax}_{j \in \mathcal{M},\, P_j + t_i \leq 10(1+\delta)}(P_j + t_i)$
2. The task i is big
   (a) If $\mathcal{K} \neq \emptyset$, then
   $k = \text{argmax}_{j \in \mathcal{K}}(P_j + t_i)$
   (b) If there exists j with $(P_j + t_i) \leq 10(1 + \delta)$, then
   $k = \text{argmax}_{j \in \mathcal{M},\, P_j + t_i \leq 10(1+\delta)}(P_j + t_i)$
   (c) Else
   $k = \text{argmin}_{j \in \mathcal{M}}(P_j + t_i)$
3. Assign task i to processor k.

3.2. The case with at most m big tasks

As we assumed $\sum_i t_i = 6m$, then $\tilde{T} \geq 6$ holds.
Therefore, in order to have competitive ratio at most c, it would suffice to prove

$$T \leq 10(1 + \delta). \tag{2}$$

We shall prove this under the following two conditions:

1. $t_i \leq 6(1 + \delta)$ for all i
2. there are at most m big tasks.

It will be shown at the end why the condition 1 may be assumed without loss of generality, hence completing the proof of Theorem 3. The situation where the condition 2 is dropped will be considered in the next section, where we prove that c is a general upper bound on the competitive ratio, proving Theorem 2.

Lemma 4. Algorithm $\bar{H}$ can assign any small task to a processor without violating the bound $10(1 + \delta)$.

Proof. Since the average load of processors in a complete schedule is 6, in each step there exists a processor with load less than $6(1 + \delta)$. A small item can always be assigned there in step (1c). On the other hand, steps (1a) and (1b) obviously respect the bound. ■

Lemma 5. Under the conditions 1 *and* 2 *above, the algorithm can assign any big task to a processor without violating the bound* $10(1 + \delta)$.

Proof. Suppose for a contradiction that the algorithm cannot assign some big task t_i without violating the bound:

$$P_j + t_i > 10(1+\delta) \quad \forall j \in \mathcal{M}.$$

Because the number of big tasks is not greater than m, there exist some processors (may be more than one) loaded with small tasks only. Let processor k be the one among them with minimum current load. We have assumed

$$P_k + t_i > 10(1+\delta)$$

and therefore

$$P_k > 4(1+\delta)$$

as no task exceeds $6(1+\delta)$.

Processor k was loaded with small tasks and so the bound $4(1+\delta)$ was violated. Let us consider the small task i' that violated the bound $4(1+\delta)$ for P_k. The task i' has been assigned by rule (1c). Now focus on the step when task i' was assigned. Denote by $\mathcal{M}_1 \subseteq \mathcal{M}\backslash\{k\}$ the set of processors with loads exceeding $4(1+\delta)$. Since task i' was assigned by rule (1c) to the most loaded processor with load no more than $10(1+\delta) - t_{i'}$ and task i' is small, the following hold:

$$\begin{aligned} t_{i'} + P_l &> 10(1+\delta) \\ P_l &> 6(1+\delta) \quad \forall l \in \mathcal{M}_1. \end{aligned} \tag{3}$$

On the other hand, for the set $\mathcal{M}_2 = \mathcal{M}\backslash\mathcal{M}_1\backslash\{k\}$ of processors (corresponding to the cases of $P_l < 4(1+\delta)$) we have

$$t_{i'} + P_l > 4(1+\delta) \quad \forall l \in \mathcal{M}_2, \tag{4}$$

because the rule (1b) was not applied. Let m_1 and m_2 denote the cardinality of $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively.

Let us consider the processors in $\mathcal{M}_2$. We will show the following inequality

$$\sum_{l \in \mathcal{M}_2} P_l > 2(1+\delta)m_2 - 2(1+\delta). \tag{5}$$

Let us notice that there is at most one processor with $P_{l'} \leq 2(1+\delta)$, otherwise we would assign the tasks of two such processors to one single processor, in step (1b).

If $P_l \geq 2(1+\delta)$ for all $l \in \mathcal{M}_2$, then (5) holds. Otherwise there is a processor, say l', with $P_{l'} < 2(1+\delta)$. Since its tasks have not been assigned to any other $l \in \mathcal{M}_2$, we have $P_l > 2(1+\delta)$ for all $l \in \mathcal{M}_2\backslash\{l'\}$, so that (5) follows.

Consider the tasks between task i' and task i. We claim that among those tasks, each processor in $\mathcal{M}_2$ has been loaded with at least one big task. Indeed, each $l \in \mathcal{M}_2$ has been loaded with a (new) task violating the bound $4(1+\delta)$, for otherwise the current task i could be assigned there in (2b); and the new load could not be a small task, since it would be assignable to processor k (or maybe a larger one) in step (1c).

Let us now estimate the sum of tasks.

$$\sum_{l\in\mathcal{M}_1} P_l + \sum_{l\in\mathcal{M}_2} P_l + (P_k + t_i) > 6(1+\delta)m_1 + (2(1+\delta)m_2 - (1+\delta)2 + 4(1+\delta)m_2)$$

$$+ (P_k + t_i) > 6(m_1 + m_2) - 2 + 10 > 6m \tag{6}$$

This contradiction proves the assertion. ■

Finally, we explain why the condition 1 given at the beginning of this section may be assumed without loss of generality. If the longest task is larger, say $\max t_i = t = 6(1+\delta) + x$ for some $x > 0$, then $\tilde{T} \geq t$ while $T \leq 10(1+\delta) + x$, i.e. the algorithm is even better than c-competitive.

3.3. *The case with more than m big tasks*

Theorem 6. Under the assumption that there are more than m big tasks, algorithm $\bar{H}$ satisfies

$$T \leq c\tilde{T} \tag{7}$$

for all feasible problem instances.

Proof. In this case any optimal schedule assigns at least two big tasks to the same processor, therefore

$$\tilde{T} \geq 8(1+\delta)$$

Suppose first that $t_i \leq 8(1+\delta)$ holds for all tasks i. Since in each step there is a processor loaded less than 6, we obtain also in (2c) that

$$T < 6 + \max t_i \leq 14 + 8\delta.$$

Hence, by the choice of δ, the makespan does not exceed

$$14 + 8\delta = \frac{5}{3} \cdot 8 \cdot (1+\delta)^2 \leq c\tilde{T}.$$

Finally, if the largest task has length $t = 8(1+\delta) + x$ for some $x > 0$, then the off-line optimum is larger by at least x, while the on-line makespan is larger by at most x, hence the c-competitiveness remains valid also in this case. ■

Proof of Theorem 2. The assertion is a direct corollary of Theorems 3 and 6.

4. CONCLUSIONS

We have shown that, if the sum of the processing times of the tasks is known in advance, an algorithm exists for the on-line multiprocessor scheduling problem with competitive ratio $\frac{\sqrt{6}+1}{2} < 1.725$ for any number of processors. An asymptotic lower bound slightly bigger than 1.565 has also been obtained on the performance of any on-line algorithm. The reduction of the gap between these two values remains an open problem. The ratio 1.725 substantially improves the best known competitive ratio 1.923 for the pure on-line multiprocessor problem. The improvement is obtained thanks to the information on the total processing time of the tasks of the instance. No detailed

information on the tasks is assumed to be known in advance. It would be interesting to discover whether other types of information on the instance may further improve the performance of an algorithm for the on-line multiprocessor scheduling problem.

REFERENCES

1. Albers, S., "Better bounds for online scheduling," *SIAM J. Comput.*, **29**, 459–473 (1999).
2. Angelelli, E., "Semi on-line scheduling on two parallel processors with known sum and lower bound on the size of the tasks," *Central Europ. J. of Oper. Res.*, **8**, 285–295 (2000).
3. Angelelli, E., M. G. Speranza, and Zs. Tuza, "Semi on-line scheduling on two parallel processors with upper bound on the items," *Algorithmica*, **37**, 243–262 (2003).
4. Angelelli, E., M. G. Speranza, and Zs. Tuza, "Semi on-line scheduling on three processors with known sum of the tasks," submitted.
5. Azar, Y. and O. Regev, "On-line bin-stretching," *Theoretical Computer Science*, **268**, 17–41 (2001).
6. Faigle, U., W. Kern, and Gy. Turán, "On the performance of on-line algorithms for particular problems," *Acta Cybernetica*, **9**, 107–119 (1989).
7. Fleischer, R. and M. Wahl, "On-Line scheduling revisited," *Journal of Scheduling*, **3**, 343–353 (2000).
8. Girlich, E., V. Kotov, and M. Kovalev, "Semi on-line algorithm for multiprocessor scheduling problem with a given total processing time," Technical Report 98-05, The Mathematical Department, University Magdeburg, Germany, 1998.
9. He, Y. and G. Zhang, "Semi on-line scheduling on two identical machines," *Computing*, **62**, 179–187(1999).
10. Kellerer, H., V. Kotov, M. G. Speranza, and Zs. Tuza, "Semi on-line algorithms for the partition problem," *Oper. Res. Letters*, **21**, 235–242 (1997).