An efficient heuristic for on-line scheduling in system with one fast machine

Natallia Kokash

Department of Information and Communication Technology, University of Trento,
Via Sommarive, 14, 38050 Trento, Italy
email: natallia.kokash@dit.unitn.it

Abstract. In this paper, the problem of on-line scheduling of independent jobs on m uniform machines $(M_1, M_2, ..., M_m)$ is considered. Each machine M_i, $1 \leq i \leq m-1$, has a normalized processing speed $s_i = 1$ and the machine M_i has speed $s_m = s > 1$. We present an efficient on-line algorithm and prove that its worst-case ratio is 2, which coincides with the previously established lower bound for this problem for an arbitrary $s > 1$ and $m \geq 3$.

1 Introduction

A uniform machine system consists of m, $m \geq 1$ machines $(M_1, M_2, ..., M_m)$. A processing speed s_i, $s_i \geq 1$ is associated with each machine. In one unit of time M_i can carry out s_i units of processing. A list $(l_1, l_2, ..., l_n)$ of n jobs is given on-line, i.e., the jobs arrive one by one. Neither the total number of jobs that need to be scheduled nor the size of the jobs are previously known. The processing time of job l_i becomes known only when l_{i-1} has already been scheduled. As soon as job l_i appears, it must be assigned to one of the machines. Our goal is to minimize the *makespan*, i.e., the maximum completion time over all jobs in a schedule. For any algorithm H, its quality is measured by the worst-case ratio

$$R^H(m) = \sup_L \{C^H(L)/C^{OPT}(L)\},$$

where L is a list of jobs, $C^H(L)$ denotes the makespan produced by the heuristic H on the machines and the list L of jobs, and $C^{OPT}(L)$ denotes the corresponding makespan in some optimal schedule.

In this paper, we consider the particular instance of the above problem when $s_i = 1$ for $i = 1, ..., m-1$ and $s_m = s > 1$. We call the machine of speed s *fast*, and all others are *regular* machines. This case is interesting since the systems with one fast processor managing several regular machines are quite common.

The paper is organized as follows. Section 2 analyzes previous works on the problem. Section 3 establishes a lower bound for an optimal solution. In section 4, we present a new on-line algorithm that beats the worse-case ratio of existing heuristics for this problem. Finally, in section 5 we draw some conclusions.

2 Previous work

The basic problem with all identical machines ($s = 1$) was studied in a sequence of papers, each improving either the upper bound or the lower bound on the competitive ratio [10, 7, 3, 12, 1, 9, 6]. The problem with one fast machine was initially stated in [8]. List Scheduling (LS) is a simple example of a non-preemptive on-line algorithm, that fits in particular for the problem with one fast and several regular machines. It always assigns the current job to the machine that will complete it first. In 1980, Cho and Sahni [4] showed that for our special case

$$R^{LS}(m,s) \leq 1 + \frac{m-1}{m+s-1} min\{2,s\} \leq 3 - \frac{4}{m+1},$$

and the bound $R^{LS}(m) = 3 - \frac{4}{m+1}$ is achieved when $s = 2$.

Li and Shi [11] proved that the worst-case performance guarantee of LS cannot be improved for $m = 2$ and $m = 3$ by any heuristic, and presented the algorithm with worst-case performance at most

$$\frac{3m-1}{m+1} - \epsilon_m.$$

Asymptotically their algorithm did not improve the heuristic LS since ϵ_m may tend to zero as m tends to infinity. The worst-case ratio of that algorithm equals $R^{LS}(m,s)$ for most $s > 1$. Finally, for $m \geq 4$ machines, they gave a lower bound of 2. Two questions were left open.

1. Are there on-line scheduling algorithms with worst case better than $R^{LS}(m,s)$ for any fixed $s > 1$?
2. For $m \geq 4$ machines, a lower bound of 2 was given. How can we get a greater lower bound?

We propose a heuristic with worst-case ratio of 2, that is better than $R^{LS}(m)$ for $m \geq 4$ and an arbitrary $s > 1$. This result automatically proves that a greater lower bound does not exist.

The main idea of our algorithm is related to load balancing [2]. A comprehensive discussion of the systems where several processors are faster than others can be found in [5].

3 Lower bounds for the optimal solution

Let $L^n = (l_1, ..., l_n)$ denote a sequence of jobs to be scheduled. To simplify notation, we will identify each job with its length, that is, l_i, $l_i > 0$, denotes a processing time on a regular machine for the i-th job. Let $(l_{max_1}, ..., l_{max_k})$ be a sequence of k largest jobs ordered by length.

Theorem 1. *The following lower bound for an optimal solution holds*

$$c(L^n, s) = max\left\{\sum_{i=1}^{n} \frac{l_i}{m+s-1}, min\left\{\sum_{i}^{k} \frac{l_{max_i}}{s}, l_{max_k}\right\}\right\}.$$

An optimal makespan cannot exceed the average machine load

$$C^{OPT}(L^n, s) \geq \sum_{i=1}^{n} \frac{l_i}{m + s - 1},$$

and either the time to process k largest jobs on the fast machine or the time to process the smallest of these jobs on the regular machine, that is

$$C^{OPT}(L^n, s) \geq min \left\{ \sum_{i}^{k} \frac{l_{max_i}}{s}, l_{max_k} \right\}.$$

Combining the above inequations we get the theorem statement.

Corollary 1. *For any heuristic H,* $C^H(L^n, s) \geq c(L^n, s)$.

Corollary 2. $C^{OPT}(L^n, s) \geq \frac{l_{max_1}}{s}$.

4 The algorithm

In this section we present our heuristic. In the algorithm the parameter α, defining the upper load bound for the regular machines, is used. The exact value of α will be specified later.

Let L_j^k denote the load of machine j after an assignment of $k, k \leq n$, jobs. Machine load is calculated as the sum of processing times over all jobs scheduled to the machine.

Algorithm 1 (Load Balancing LB**)** *1. Assign the first job to the fast machine.*

2. *Assuming that k jobs arrived,* $2 \leq k \leq n$, *calculate* $c(L^k, s)$.
3. *Find set* $I = \{i : i \in \{1, ..., m-1\}\}$, *such that*

$$L_i^k \leq (1 + \alpha)c(L^k, s).$$

4. *Assign job* l_k *to any of the machines* M_i, $i \in I$, *if* $I \neq \emptyset$, *or to the machine* M_m, *otherwise.*

Since for $m \geq 3$ the lower bound for any on-line heuristic is 2 [11], we conclude that $\alpha \geq 1$. Throughout the following analysis we establish some important properties of the above algorithm that will help us to define its worst-case ratio. Assuming that s is constant for a given problem instance further we refer to $c(L^n, s)$ as to $c(L^n)$.

Lemma 1. *Let* $(l_{k_1}, l_{k_2}, ..., l_{k_r})$, $k_1 < k_2 < ... < k_r$, *be a sequence of jobs, assigned by the algorithm LB to the fast machine, for which the following conditions take place:*

$$\begin{cases} c(L^{k_1}) = c(L^{k_r}), \\ L_m^{k_1} > c(L^{k_r}). \end{cases}$$

The number r of such jobs cannot exceed $\lceil s - 1 \rceil$, *where* $\lceil \cdot \rceil$ *denotes the ceiling function.*

Algorithm LB assigns jobs to the fast machine if $\forall j$, $j = 1, ..., m-1$, the following holds

$$\begin{cases} L_j^{k_1-1} + l_{k_1} > (1+\alpha)c(L^{k_1}), \\ \qquad\qquad ... \\ L_j^{k_r-1} + l_{k_r} > (1+\alpha)c(L^{k_r}), \end{cases} \tag{1}$$

where $L_j^{k_i-1}$ defines the load of machine j after scheduling $k_i - 1$, $i = 1, ..., r$, jobs.

For any assignment k_i, $i = 1, ..., r$, suppose that

$$L_j^{k_i-1} > c(L^{k_i}), \quad \forall j, 1 \le j \le m-1.$$

Since the algorithm does not assign the job l_{k_i} to the regular machine, its load after the assignment of $k_i - 1$ jobs does not change after the assignment of k_i-th job

$$L_j^{k_i-1} = L_j^{k_i}.$$

Obviously, $L_m^{k_i} > L_m^{k_1}$, $\forall i$, $1 \le i \le r$. Taking into account the lemma assumptions, we get

$$L_j^{k_i} > c(L^{k_i}), \quad \forall j, 1 \le j \le m,$$

i.e., the load of any machine is higher than average, which is impossible. This proves that our supposition was wrong and

$$\forall k_i, 1 \le i \le r, \exists j, 1 \le j \le m-1 : \quad L_j^{k_i-1} \le c(L^{k_i}). \tag{2}$$

From (1) and (2) we get

$$l_{k_i} > \alpha c(L^{k_i}), \quad i = 1, ..., r.$$

Suppose that $r > \lceil s-1 \rceil$. Since r is integer, there exists a job l_t, $t = \lceil s-1 \rceil + 1 = \lceil s \rceil$, scheduled to the fast machine. Assuming $\alpha \ge 1$, and retaining lemma assumptions $c(L^{k_1}) = c(L^{k_t}) = c(L^{k_r})$, we infer

$$l_{k_i} > c(L^{k_t}), \quad \forall i, 1 \le i \le t. \tag{3}$$

Summing up all equations (3) we get

$$sc(L^{k_t}) \le \lceil s \rceil c(L^{k_t}) < \sum_{i=1}^{t} l_{k_i}. \tag{4}$$

On the other side, from the theorem 1, assigning $k = t$, we obtain either

$$c(L^{k_t}) \ge l_{min},$$

where l_{min} is the smallest job among $(l_{k_1}, ..., l_{k_t})$, or

$$sc(L^{k_t}) \ge \sum_{i=1}^{t} l_{k_i}.$$

The first case contradicts (3), whereas the second one does not comply with (4). Consequently, $r \leq \lceil s-1 \rceil$, and the lemma assertion is proved.

Intuitively, lemma 1 asserts that the scheduling of $([s-1]+1)$-th job on the fast machine inevitably increases the lower bound for the optimal solution.

Theorem 2. *Given $\alpha \geq 1$, algorithm LB can assign all jobs to the machines in such a way that*

$$L_j^n \leq (1+\alpha)C^{OPT}(L^n), \quad j = 1, ..., m.$$

Given $\alpha \geq 1$, assume that there exists a job l_k, $1 \leq k \leq n$, such that the theorem statement does not hold. In particular, it means that the assignment of the job l_k to the ordinary machine j, $1 \leq j \leq m-1$, gives

$$L_j^n \geq L_j^{k-1} + l_k > (1+\alpha)C^{OPT}(L^n). \tag{5}$$

From (5) and $L_j^{k-1} \leq c(L^n) \leq C^{OPT}(L^n)$ follows

$$l_k > \alpha C^{OPT}(L^n).$$

On the other hand,

$$l_k \leq C^{OPT}(L^n)$$

if in the optimal plan l_k is executed on the regular machine, or

$$l_k < sC^{OPT}(L^n)$$

if in the optimal plan l_k is executed on the fast machine. So we infer that algorithm LB may not be able to assign the job l_k to the regular machine for

$$\begin{cases} \alpha < 1, & l_k \in L_j^n(OPT),\ 1 \leq j \leq m-1, \\ \alpha < s, & l_k \in L_m^n(OPT). \end{cases} \tag{6}$$

In such a situation it assigns the job to the fast machine. Below we will ascertain the parameters α for which

$$L_m^k > (1+\alpha)C^{OPT}(L^n). \tag{7}$$

For a list L^n, let $(l_{k_1}, ..., l_{k_r})$ be the jobs, assigned to the fast machine in the way the following conditions take place

$$\begin{cases} L_m^{k_1-1} \leq c(L^n), \\ L_m^{k_1} > c(L^n). \end{cases}$$

According to lemma 1, $r \leq \lceil s-1 \rceil$, that is

$$L_m^n = L_m^{k_1-1} + b > (1+\alpha)C^{OPT}(L^n),$$

where

$$b = \sum_{i=1}^{\lceil s-1 \rceil} \frac{l_{k_i}}{s}$$

is the time to process jobs $(l_{k_1}, ..., l_{k_r})$ on the fast machine.

If (7) holds, then $l_k \in (l_{k_1}, ..., l_{k_r})$. Taking into account the assumption

$$L_m^{k_1-1} \leq c(L^n) \leq C^{OPT}(L^n),$$

we get

$$b > \alpha C^{OPT}(L^n). \tag{8}$$

Let us now estimate in what time the jobs $(l_{k_1}, ..., l_{k_{\lceil s-1 \rceil}})$ can be completed in the optimal plan. Since for each l_{k_i} executed on the regular machine $l_{k_i} \leq C^{OPT}$, we have

$$b \leq \frac{\lceil s-1 \rceil}{s} C^{OPT}$$

if at least one of the jobs is executed on the regular machine, and

$$b \leq C^{OPT}$$

if all jobs are executed on the fast machine. Since

$$\sup_s \{\lceil s-1 \rceil / s\} = 1,$$

algorithm LB may not be able to assign the job l_k to the fast machine if $\alpha < 1$.

Finally, the job l_k may not be assigned neither to the regular nor to the fast machine in such a way that

$$L_j^n \leq (1+\alpha) C^{OPT}(L^n)$$

only if $\alpha < 1$. This contradicts $\alpha \geq 1$ clause, and we conclude that the job l_k affecting the theorem statement does not exist.

Theorem 3 (Worst-case ratio). *The worst-case ratio*

$$R^{LB} = 2, \quad \forall s > 1, \forall m \geq 3.$$

The statement follows straightforwardly from the theorem 2 when $\alpha = 1$. The condition $m \geq 3$ arises from the lower bound estimation which was used in the proof of lemma 1.

The proposed LB algorithm is an efficient on-line heuristic for our scheduling problem. In contrast to LS, its strategy is to assign jobs to the regular machines if their loads are not greater than some flexible bound, and to the fast machine, otherwise (see Table 1). The bound depends on the length of the previously processed jobs and it is reestablished after receiving each new job for an assignment.

5 Conclusions

In this paper we proposed an on-line algorithm for scheduling jobs on a set of machines in which one machine $s, s > 1$ times faster than others. We proved

Table 1. Load Balancing

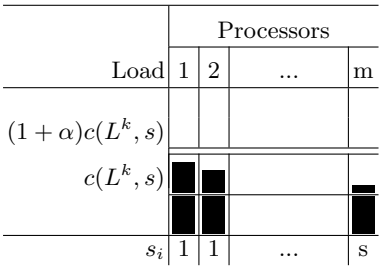

Table 2. Heuristic Comparison

m	$R^{LS}(m,2) - R^{LB}(m,2)$	$R^{A}(m,2) - R^{LB}(m,2)$
4	0.2	0.1835
5	0.3333	0.3025
9	0.6	0.5353
∞	1	0.8795

that our algorithm has the worst-case ratio of 2 for $s > 1$ and $m \geq 3$. Its upper bound coincides with the previously proven lower bound of 2 for this problem. This result solves the problems posed in [11]. In Table 2 the reduction of the worst-case ratio for the cases studied in that work is shown.

Note, however, that the lower bound was established in view of an arbitrary speed $s > 1$. For some fixed s the worst-case ratio can be still improved.

The idea of using flexible bounds in the scheduling heuristics can help to find the better algorithms for other assignment problems. As future work, we intend to proceed verifying this strategy on different heterogeneous systems.

Acknowledgments

I am grateful to Professor Vladimir Kotov for bringing this problem to my attention.

References

1. Suzanne Albers, *Better bounds for online scheduling*, SIAM Journal on Computing, 29, pp. 459-473, 1999.
2. Yossi Azar, Andrei Broder, Anna Karlin *On-line load balancing*, In Proc. 33rd IEEE Annuul Symposium on Fourds, 1992.
3. Yair Bartal, Howard Karloff, Yuval Rabani, *A better lower bound for on-line scheduling*, Information Processing Letters, 50, pp. 113-116, 1994.
4. Yookun Cho, Sartaj Sahni, *Bounds for List Schedules on Uniform Processors*, SIAM Journal on Computing, 9(1), pp. 91-103, 1980.

5. Leah Epstein, Rob van Stee, *Online Scheduling of Splittable Tasks in Peer-to-Peer Networks*, Proceedings of SWAT, LNCS, 3111, pp. 408–419, 2004.
6. Rudolf Fleischer, MichaelaWahl, *Online scheduling revisited*, Journal of Scheduling, 3, pp. 343-353, 2000.
7. Gabor Galambos, Gerhard J. Woeginger, *An on-line scheduling heuristic with better worst case ratio than Grahams list scheduling*, SIAM Journal on Computing, 22, pp. 349-355, 1993.
8. Teofilo F. Gonzalez, Oscar H. Ibarra, S. Sahni, *Bounds for LPT Schedules on Uniform Processors*, SIAM Journal on Computing, 6(1), pp. 155-166, 1977.
9. Todd Gormley, Nick Reingold, Eric Torng, Jeffery Westbrook, *Generating adversaries for request-answer games*, In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 564-565, 2000.
10. Ronald L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Technical Journal, 45, pp. 1563-1581, 1966.
11. Rongheng Li, Lijie Shi, *An on-line algorithm for some uniform processor scheduling*, SIAM Journal on Computing, 27(2), pp. 414–422, 1998.
12. David R. Karger, Steven J. Phillips, Eric Torng, *A better algorithm for an ancient scheduling problem*, Journal of Algorithms, 20, pp. 400-430, 1996.