A Fast 5/2-Approximation Algorithm for Hierarchical Scheduling

Marin Bougeret[1,⋆], Pierre-François Dutot[1], Klaus Jansen[2,⋆⋆], Christina Otte[2], and Denis Trystram[1]

[1] Grenoble University
38330 Montbonnot Saint Martin, France
{bougeret,pfdutot,trystram}@imag.fr
[2] Department of Computer Science
Christian-Albrechts-University to Kiel
Christian-Albrechts-Platz 4, 24098 Kiel, Germany
{kj,cot}@informatik.uni-kiel.de

Abstract. We present in this article a new approximation algorithm for scheduling a set of n independent rigid (meaning requiring a fixed number of processors) jobs on hierarchical parallel computing platform. A hierarchical parallel platform is a collection of k parallel machines of different sizes (number of processors). The jobs are submitted to a central queue and each job must be allocated to one of the k parallel machines (and then scheduled on some processors of this machine), targeting the minimization of the maximum completion time (makespan). We assume that no job require more resources than available on the smallest machine.

This problem is hard and it has been previously shown that there is no polynomial approximation algorithm with a ratio lower than 2 unless $P = NP$. The proposed scheduling algorithm achieves a $\frac{5}{2}$ ratio and runs in $O(log(np_{max})knlog(n))$, where p_{max} is the maximum processing time of the jobs. Our results also apply for the Multi Strip Packing problem where the jobs (rectangles) must be allocated on contiguous processors.

1 Introduction

Context and motivation. The evolution of the technology over the last past years leads to the emergence of new types of parallel and distributed platforms. Many of these new computing systems are hierarchical in nature. Such computing systems may be composed of several parallel machines, each having a certain number of processors, or composed of several clusters, each having a certain number of computers [1]. Thus, we make no distinctions between machines/clusters on one side, and between processors/computers on the other side.

In order to fully exploit the large number of available resources for reaching the best performances, we need to revisit the classical resource management

⋆ M. Bougeret has a PhD grant from DGA-CNRS.
⋆⋆ Research supported by German Research Foundation (DFG) project JA 612/12-1, "Design and analysis of approximation algorithms for 2D and 3D packing problems".

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part I, LNCS 6271, pp. 157–167, 2010.

algorithms taking into account new features induced by the hierarchical hardware. In such platforms, the users submit their jobs in a centralized queue. The resource manager selects the jobs and allocates them to a machine, and then the jobs are scheduled locally. We assume in this work that the machines have enough processors so as each job fits entirely in any machine. Notice that jobs are not executed on processors that belong to different machines.

In the common context of an execution by successive batches, we target on the minimization of the makespan (defined as the maximum completion time over all the processors) which is the most popular objective function [2]. We assume that the processing times of the jobs are available at the beginning (the algorithm is *clairvoyant* [2]).

Related works. Let us qualify by *regular*, the case where the machines have the same number of processors. The *irregular case* corresponds to the situation where the machines have different sizes.

There exist a lot of related works for the regular case, as it is closely related to the multiple strip packing problem. Indeed, the only difference is that in multiple strip packing (MSP), the jobs are considered as rectangles and thus, they must be allocated on "consecutive" processors. This problem (and the non-contiguous version) are strongly $\mathcal{NP}$-hard, and Zhuk [3] (and later Schwiegelshohn et al. [7]) showed that there is no approximation algorithm with absolute ratio better than 2 unless $P = NP$, even for 2 machines. The proof is a gap reduction from 2-partition problem. The main related positive results for MSP are a $2 + \varepsilon$-approximation in [4] whose cost is doubly exponential in $\frac{1}{\epsilon}$, a (costly) 2-approximation and an AFPTAS in [5]. Some variants have been investigated (like in [6]) where additional constraints come from pre-allocation.

Concerning the hierarchical scheduling problem in the irregular case, one of the most relevant work is the result of Schwiegelshohn, Tchernykh and Yahyapour [7]. The authors address a version of this problem where a job may not fit everywhere, meaning that a job may requires more processors than the total number of processors of some machines. They provide a 3-approximation for the off-line case, and a 5-approximation when the jobs are submitted over time (on-line). Their algorithms handle non-clairvoyant jobs (meaning that the processing time of a job is only known at the end of its execution), but do not easily apply for MSP. They also emphasize that the performance of the classical Garey and Graham's list scheduling algorithm is significantly worsened in hierarchical environments. Finally, (as we will see in remark 4), it is possible to derive a better approximation ratio using the remark in [4] that leads to a $2+\epsilon$ ratio (in time polynomial in $\frac{1}{\epsilon}$) for the regular case. However, the cost of the induced algorithm is considerably too large, for instance taking $\epsilon = \frac{1}{2}$ leads to a cost in $\Omega(n^{60})$!

Contribution. Our main contribution is a new $\frac{5}{2}$-approximation (almost) greedy algorithm (called $FAHS$ for fast approximation hierarchical scheduler) for scheduling a set of (clairvoyant) rigid jobs on parallel machines that have

different number of processors, and where every job fits everywhere. The algorithm runs in $O(log(np_{max})knlog(n))$. This result also applies for MSP.

2 Preliminaries

2.1 Problem Statement

The problem studied in this paper is to schedule n jobs on a parallel system composed of k parallel machines. Machine M_i (for $1 \leq i \leq k$) has m_i processors. We assume that $m_i \leq m_{i+1}$ for any i. A job J_j is characterized by the number of processing resources needed for its completion, denoted by $q_j \in \mathbb{N}^*$ (sometimes called the *degree of parallelism*) and its corresponding processing time $p_j \in \mathbb{N}^*$. The q_j processors must be allocated on the same machine. We assume that every jobs fits everywhere, meaning that $max_j q_j \leq m_1$. We denote by $w_j = p_j q_j$ the *work* of job J_j. The objective is to minimize the maximum completion time over all the machines (called the *makespan* and denoted by C_{max}). Given a fixed instance, we denote by C^* the optimal makespan for this instance.

2.2 Other Notations

Given a schedule, we denote by $u_i(t)$ the utilization of M_i at time t, which is the sum of the required number of processors of all the jobs that are running on M_i at time t. We say that job J_{j_1} is higher than job J_{j_2} if $q_{j_1} \geq q_{j_2}$. We extend the p_j, q_j, w_j previous notations to $P(X)$, $Q(X)$ and $W(X)$ where X is a set of jobs and the capital letter denotes the sum of the corresponding quantity of all the jobs in the set (for instance $P(X) = \Sigma_{J_j \in X} p_j$).

We call *shelf* a set of jobs that are scheduled on the same machine and finish at the same time. Scheduling a shelf at time t means that all the jobs of the shelf finish at time t. This reverse definition of shelf (where usually jobs start simultaneously) is used because some machines are filled from right to left. Using v as a guess for the optimal makespan C^*, let $L = \{J_j | p_j > \frac{v}{2}\}$ denote the set of long jobs, and let $H_i = \{J_j | q_j > \frac{m_i}{2}\}$ denote the set of high jobs for M_i, for $1 \leq i \leq k$.

2.3 Useful Results

We present briefly in this section some classical results that we will use later. Let us recall first the well known Highest First (HF) list algorithm that, given a set of jobs and one machine, starts from time 0 and schedules at any time the highest job which fits. Secondly, let us state the following result that was established by Steinberg [8]. Notice that even if this result is stated in term of rectangle packing, it remains valid for parallel jobs (without the contiguous constraint).

Theorem 1 ([8]). *Let $L = \{r_1, \ldots, r_n\}$ be a set of rectangles. Let w_j, h_j and s_j denote the width, the height and the surface of r_j. Let $S(L) = \Sigma_{j=1}^{n} s_j$, $w_{max} = max_j w_j$ and $h_{max} = max_j h_j$. If*

$$w_{max} \leq u, h_{max} \leq v \text{ and } 2S(L) \leq uv - max(2w_{max} - u, 0)max(2h_{max} - v, 0)$$

then it is possible to pack L (in time $O(nlog^2(n)/log(log(n)))$ in a rectangular box of width u and height v.

We are now ready to describe the $FAHS$ algorithm.

3 Algorithm

As we target a $5/2$ ratio, it is convenient to consider that we have in each machine an empty available rectangular area of height m_i and length $5C^*/2$. Moreover, we want to particularly take care of the jobs that have a processing time greater than $C^*/2$, since these jobs are harder to schedule. As C^* is unknown, we use the classical dual approximation technique [9], and we denote by v the current guess of C^*. According to this technique, our goal is to provide the $FAHS$ (Fast Approximation Hierarchical Scheduler) algorithm that either schedules all the jobs with a makespan lower than $5v/2$ or rejects v, implying that $v < C^*$. For the sake of simplicity, we do not mention in the algorithm the "reject" instruction. Thus, we consider throughout the paper that $v \geq C^*$, and it is implicit that if one of the claimed properties is wrong during the execution, the considered v should be rejected.

$FAHS$ is described in detail in Algorithm 1. The principle is to fill the machines from the smallest (M_1) to the largest one. As long as it is possible, we aim at scheduling on each M_i a set of jobs whose total work is greater than $m_i v$. The algorithm is designed such that the only reason for failing (meaning that it does not reach $m_i v$) is the lack of jobs. Thus, if a failure occurs, there will be only a few amount of jobs that will be straightforward to schedule.

In order to schedule a work of (at least) $m_i v$, we first intent to schedule a set X of jobs of H_i such that $P(X)$ is greater than $2v$ (phase 1). If it fails, we schedule as many jobs of H_i as possible (again in phase 1), with a strict limit $P(X) < 3v/2$ to have enough space to pack a shelf of long jobs between $3v/2$ and $5v/2$. Then, we select (in phase 2) some jobs of $I \backslash H_i$, targeting a total work of at least $m_i v$.

These selected jobs are then scheduled in phase 3. If the work goal $m_i v$ is reached we schedule the considered jobs in M_i (using the *add* algorithm in Algorithm 2 in case 1 or HF in case 2), otherwise (case 3) there is by construction only a few amount of unscheduled jobs, and they are scheduled using the *pack* procedure described in Algorithm 3.

Remark 1 (Notation). For any set X considered in the algorithm or in the proof, we use the $\tilde{X}$ notation to denote the set X during the execution of the algorithm ($\tilde{X} \subset X$). We consider that a job is removed from the instance when it is scheduled.

Algorithm 1. Fast Approximation Hierarchical Scheduler ($FAHS$)

for all i, $\tilde{H}_i \leftarrow H_i$ and $\tilde{L} \leftarrow L$
for $i = 1$ to k **do**
——————————- phase 1 ——————————-
if $\tilde{H}_i \cap \tilde{L} \neq \emptyset$ **then**
• $J_{j_i} \leftarrow$ highest job of $\tilde{H}_i \cap \tilde{L}$
• schedule J_{j_i} at time 0 and remove it from $\tilde{H}_i \cap \tilde{L}$
else
• $J_{j_i} \leftarrow$ "dummy" job (with $p_{j_i} = q_{j_i} = 0$)
end if
if $p_{j_i} + P(\tilde{H}_i \setminus \tilde{L}) \geq 2v$ **then**
• $target \leftarrow 2v$
else
• $target \leftarrow v$
end if
• schedule sequentially (in any order) jobs of $\tilde{H}_i \setminus \tilde{L}$ (and remove them from $\tilde{H}_i$) until one of them ends later than $target$ or until $(\tilde{H}_i \setminus \tilde{L} = \emptyset)$
• $High \leftarrow$ jobs scheduled on M_i
• reschedule jobs of $High$ in non-increasing order of height
if $target = v$ **then**
——————————- phase 2 ——————————-
• $Select \leftarrow \emptyset$
• add to $Select$ some jobs of $\tilde{I} \setminus \tilde{H}_i$ (and remove them from $\tilde{I}$) in non increasing order of their work, until $(W(High) + W(Select) \geq m_i v)$ or until $(\tilde{I} \setminus \tilde{H}_i = \emptyset)$
——————————- phase 3 ——————————-
if $W(High) + W(Select) > \frac{5}{4} m_i v$ **then** //**case 1**
• schedule $Select$ using $add(i, Select)$
else if $\frac{5}{4} m_i v \geq W(High) + W(Select) \geq m_i v$ **then** //**case 2**
• reschedule jobs $High \cup Select$ on M_i using HF algorithm
else //**case 3**
if i=k **then**
• reschedule $High \cup Select \cup \tilde{H}_i$ on M_i using HF algorithm (and remove jobs from $\tilde{H}_i$)
else
• reschedule $High \cup Select$ on M_i using HF algorithm
• schedule $\tilde{H}_i$ on $\{M_{i+1}, \ldots, M_k\}$ using $pack(i, \tilde{H}_i)$ (and remove jobs from $\tilde{H}_i$)
end if
• break //all the jobs are scheduled
end if
end if
end for

Let us now quickly analyze the running time of $FAHS$.

Remark 2. $FAHS$ runs in $O(log(np_{max})knlog(n))$.

Proof. Phase 1 and phase 2 run in $O(nlog(n))$ as the jobs are sorted according to their height at the end of phase 1 and according to their work before starting

Algorithm 2. $add(i, Select)$

- create a shelf Sh with the highest jobs of $Select$ and schedule Sh at time $\frac{5v}{2}$
- reschedule Sh such that the shortest jobs are on the top (as depicted figure 1)

if one job $J_1 \in Select$ remains **then**
- $t_1 \leftarrow$ smallest t such that $u_{M_i}(t) + q_1 \leq m_i$
- schedule J_1 at time t_1

end if
if two jobs $\{J_1, J_2\} \subset Select$ remain **then**
- create a shelf Sh' using J_1 and J_2 and schedule Sh' at time $\frac{3v}{2}$

end if

Algorithm 3. $pack(i, \tilde{H}_i)$

if $\tilde{H}_i \cap L = \emptyset$ **then**
- schedule jobs of $\tilde{H}_i$ sequentially on M_{i+1} from 0 (we know that $i < k$)

else
- $x \leftarrow i$

while $\tilde{H}_i \cap L \neq \emptyset$ **do**
- $x \leftarrow x + 1$
- $Sh \leftarrow \emptyset$

while $((Q(Sh) \leq m_x)$ and $(\tilde{H}_i \cap L \neq \emptyset))$ **do**
- add to Sh a job J_j from $\tilde{H}_i \cap L$

end while
- schedule all the jobs of $Sh \setminus \{J_j\}$ at time 0

if $Q(Sh) > m_x$ **then**
- schedule J_j at time v

else
- schedule J_j at time 0

end if
end while
- schedule all the remaining jobs (which all belong to $\tilde{H}_i \setminus L$) sequentially on M_k from v

end if

the phase 2. Phase 3 runs in $O(nlog(n))$ because of the HF algorithm. Thus, for a fixed value of v, the cost of $FAHS$ is in $O(knlog(n))$, and the dichotomic search on v can be done in $log(np_{max})$ since np_{max} is an upper bound of the optimal. □

4 Analysis of the Algorithm

Notice first that the case where $target = 2v$ is clear as in this case the schedule has a makespan lower than $\frac{5v}{2}$, and the scheduled work is greater than $2v\frac{m_i}{2} = m_i v$. Thus, we will focus on the second part of $FAHS$, where $target = v$. We will first prove in lemma 1, 2 and 3 that (if $v \geq C^*$, as usual) it is possible to schedule the considered set of jobs with a makespan lower than $\frac{5v}{2}$ in the three cases of phase 3. Then, the final proof of $FAHS$ is derived in theorem 2.

Lemma 1 (Feasibility of case 1). *Any call to add$(i, Select)$ produces a valid schedule of makespan lower than $\frac{5v}{2}$.*

Proof. Notice first that calling the *add* procedures implies that $target = v$. We assume that the sets $High$ and $Select$ have been constructed in phase 1 and 2, implying $W(High) + W(Select) > \frac{5}{4}m_i v$. Let $p = |Select|$ and $Select = \{J'_1, \ldots, J'_p\}$, with $w'_{j+1} \leq w'_j$ for all j, as stated in Algorithm 1. Let us start with the following properties:

1) $P(High) \leq \frac{3}{2}v$
2) for all $J'_j \in Select$, $w'_j > \frac{m_i v}{4}$, implying $J'_j \in L$ and $0 \leq p \leq 4$
3) $\forall X \subset Select, W(High) + W(Select \setminus X) < vm_i$

The first property is true because $target = v$, and the jobs of $High$ scheduled in phase 1 have a processing time lower than $\frac{v}{2}$. The second point comes from the fact that $W(Select \setminus \{J'_p\}) < m_i v$ and $W(Select) > \frac{5}{4}m_i v$. This implies that $w'_p > \frac{m_i v}{4}$ and property 2. The third point is true by definition of phase 2.

We start by creating a shelf Sh with the higher jobs of $Select$ (as depicted figure 1). We schedule Sh at time $\frac{5v}{2}$ (remind that it means that all the jobs of Sh finish at time $\frac{5v}{2}$), which is possible according to property 1. Let α denote the number of jobs in this shelf. As $Select \cap H_i = \emptyset$, we know that $\alpha \geq 2$, implying that cases where $p \leq 2$ are straightforward.

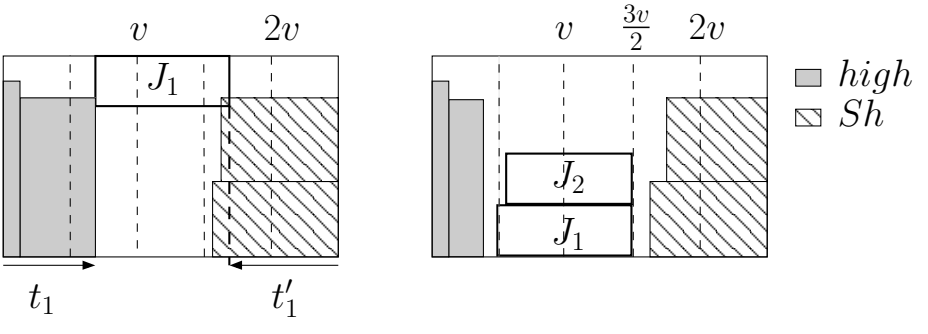

Fig. 1. Example for the *add* procedure. Case where $p = 3$ is depicted on the left side, and case where $p = 4$ is depicted on the right side.

We now study the cases where $p = 3$. Let J_1 be the remaining job (we use the same notations as in Algorithm 2). Let us now analyze how the *add* procedure schedules J_1. Let us suppose by contradiction that J_1 intersects Sh. Let $t'_1 = \frac{5v}{2} - t_1 - p_1$. We get

$$\begin{aligned} W(High \cup Sh) &> t_1(m_i - q_1) + t'_1(m_i - q_1) + (Q(Sh) - (m_i - q_1))\frac{v}{2} \\ &> t_1(m_i - q_1) + (\frac{3v}{2} - t_1)(m_i - q_1) + (3q_1 - m_i)\frac{v}{2} \\ &= m_i v \end{aligned}$$

The inequality $Q(Sh) \geq 2q_1$ used above is true since *add* schedules the highest jobs in the shelf. The inequality $W(High \cup Sh) > m_i v$ is a contradiction according to property 3, implying that J_1 must fit in the "staircase".

Let us now study the case where $p = 4$. Let $Select' = Select \setminus \{J'_4\}$. We have $W(Select') > 3\frac{m_i v}{4}$ (according to property 2), implying $W(High) + 3\frac{m_i v}{4} < W(High \cup Select') \leq v m_i$. Thus, we get $P(High)\frac{m_i}{2} < W(High) < \frac{m_i v}{4}$, leading to $P(High) < \frac{v}{2}$. Thus, the second shelf sh' scheduled at time $\frac{3v}{2}$ does not intersect jobs of $High$. It ends the proof of feasibility of case 1. □

Concerning the feasibility of case 2, we prove a slightly more general statement which can be interpreted as a particular case of Steinberg's theorem [8].

Lemma 2 (Feasibility of case 2). *Let us consider an arbitrary machine M with m processors. Let v such that for all $j \in X$, $p_j \leq v$. Let X be a set of jobs such that $W(X) \leq \alpha m v$, with $\alpha \geq 1$. Then, the HF algorithm schedule X on M with a makespan lower than $2\alpha v$.*

Proof. Let C denote the makespan of the schedule, and let J_{j_0} be a job that finishes at time C. If $q_{j_0} > \frac{m}{2}$, then all the jobs J_j of X have $q_j > \frac{m}{2}$, implying $\alpha m v \geq W(X) > \frac{C}{2} m$. Let us suppose now that $q_{j_0} \leq \frac{m}{2}$. Let s denote the starting time of J_{j_0}. As there are strictly less than q_{j_0} available processors between time 0 and s, we have $W(X) > p_{j_0} q_{j_0} + s(m - q_{j_0})$, implying $s < \frac{W(X)}{m - q_{j_0}} - \frac{p_{j_0} q_{j_0}}{m - q_{j_0}}$. Thus,

$$\begin{aligned} C &\leq s + p_{j_0} \\ &\leq \frac{W(X)}{m - q_{j_0}} + p_{j_0} \frac{m - 2q_{j_0}}{m - q_{j_0}} \\ &\leq v \frac{(\alpha + 1)m - 2q_{j_0}}{m - q_{j_0}} \end{aligned}$$

As the last expression is an increasing function of q_{j_0} and $q_{j_0} \leq \frac{m}{2}$ we get $C \leq 2\alpha v$. □

Remark 3. Notice that Lemma 2 does not apply for strip packing. However this lemma can be simply adapted using Steinberg algorithm [8], as explained in Section 5

We now study the feasibility of case 3.

Lemma 3 (Feasibility of case 3). *The pack procedure schedules all the remaining jobs.*

Proof. Let i be the index of the current machine when case 3 occurs for the first time. Let $\tilde{H}_i^0$ be $\tilde{H}_i$ at line 7 of phase 1 (after setting the value of J_{j_i}) and let $\tilde{H}_i^1$ be $\tilde{H}_i$ at the beginning of phase 3. Notice that $W(High \cup Select) < m_i v$ implies that we collected all the jobs of $\tilde{I} \setminus \tilde{H}_i^1$, and thus the set of jobs that

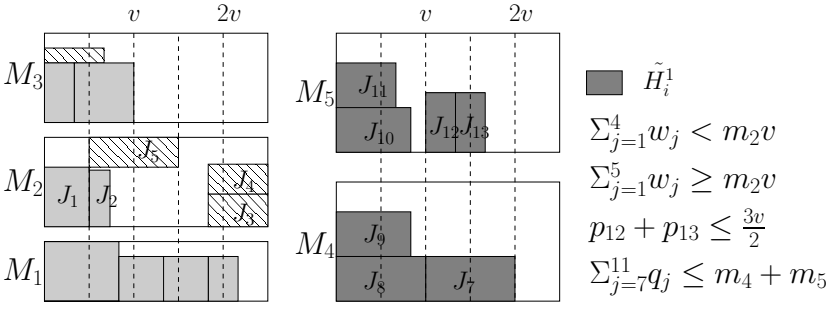

Fig. 2. Example for the *pack* procedure. Case 3 occurs for $i = 3$. $\tilde{H_i^1}$ is defined as in Lemma 3.

have not been scheduled on $\{M_1, \ldots, M_{i-1}\}$ is $High \cup Select \cup \tilde{H_i^1}$. If $i = k$, then $W(High \cup Select \cup \tilde{H_i^1}) \leq vm_i$ (otherwise, v is rejected), and HF is sufficient according to lemma 2. We consider now that $i < k$. Jobs of $High \cup Select$ are scheduled (on M_i) with a makespan lower than $\frac{5v}{2}$ using again lemma 2. We now prove that $pack(i, \tilde{H_i^1})$ successfully schedules $\tilde{H_i^1}$. If $\tilde{H_i^1} \cap L = \emptyset$, it implies that $\tilde{H_i^0} \setminus \tilde{L} = \tilde{H_i^0}$, and thus $P(\tilde{H_i^1}) \leq P(\tilde{H_i^0}) \leq 2v - p_{j_i} \leq 2v$ because phase 3 only occurs if $target = v$. Thus, we can schedule $\tilde{H_i^1}$ sequentially on M_{i+1}.

If $\tilde{H_i^1} \cap L \neq \emptyset$, then there is one job of $H_i \cap L$ scheduled on $M_1, \ldots, M_i$. Moreover, $P(\tilde{H_i^1} \setminus L) \leq P(\tilde{H_i^0} \setminus L) \leq \frac{3v}{2}$ as there was not enough jobs to fill M_i up to $2v$. We will prove that $pack(i, \tilde{H_i^1})$ can schedule $\tilde{H_i^1} \cap L$ on machines $\{M_{i+1}, \ldots, M_k\}$ such that no job (of $\tilde{H_i^1} \cap L$) is scheduled after time v on M_k. Then, it will be obvious that $\tilde{H_i^1} \setminus L$ can be added on M_k between v and $\frac{5v}{2}$.

Let us prove it by contradiction by assuming that there is a job of $\tilde{H_i^1} \cap L$ scheduled on machine M_k after time v (meaning that we stared a second shelf of jobs of $\tilde{H_i^1} \cap L$ on M_k).

For any $t \in [|1, \ldots, k|]$, let S_t denote the schedule constructed by $FAHS$ on machine M_t, and S_t^* be the schedule (for a fixed optimal solution) on machine M_t. Let $X = \cup_{t=1}^{i} S_t$ and $X^* = \cup_{t=1}^{i} S_i^*$. We have $Q(X \cap H_i \cap L) \geq Q(X^* \cap H_i \cap L)$ as in phase 1 we scheduled on machines $M_1 \ldots M_i$ the i highest jobs of $H_i \cap L$ (called the J_{j_t} in the algorithm), and the optimal cannot schedule more than i jobs of $H_i \cap L$ on these machines. Moreover, we have $Q(S_t \cap H_i \cap L) > m_t$ for every t in $[|i+1, k|]$ as we started a second shelf on each of these machines. Thus, it means that $Q(\tilde{H_i^1} \cap L) > \Sigma_{t=i+1}^{k} m_t$ which is impossible as the optimal solution had a total height of jobs of $H_i \bigcap L$ greater than $Q(\tilde{H_i^1} \cap L)$ to schedule on $M_{i+1}, \ldots, M_k$. □

We can now complete the main proof.

Theorem 2. *If $v \geq C^*$, $FAHS$ schedules all the jobs with a makespan lower than $\frac{5v}{2}$.*

Proof. If case 3 of phase 3 occurs, then we know according to lemma 3 that all the remaining jobs are scheduled in $\frac{5v}{2}$. Otherwise, we know that for every $i \in [|1, \ldots, k|]$, the $FAHS$ algorithm had a *target* equal to $2v$, or executed case 1 or case 2. This implies by construction that the area scheduled on M_i is greater than (and in this case exactly equal to) $m_i v$. Thus, there can not be unscheduled jobs at the end. □

5 Concluding Remarks

In this paper, we presented a new algorithm for scheduling a set of clairvoyant rigid jobs in a hierarchical parallel platform, assuming that every job fits in any machine. A sophisticated analysis proved that it is a $\frac{5}{2}$-approximation, improving the best existing algorithm. There exists for this problem a lower bound equal to 2.

We now explain how to adapt the algorithms and the proofs to the MSP problem where jobs, called rectangles, must be allocated on contiguous processors. The only adaptation needed is in Lemma 2, where HF can be directly replaced by Steinberg's algorithm (see Section 2.3). Phase 1, *pack* and *add* build contiguous schedule (and thus a valid schedule for rectangles), and proofs of Lemma 1 and 3 still hold for the contiguous case. Thus, we get a $\frac{5}{2}-$approximation algorithm for MSP that runs $O(log(np_{max})knlog^2(n)/log(log(n))$.

We describe now how to achieve a $2+\epsilon$ ratio for this problem at the price of a much higher algorithm complexity. This result is an extension of the result in [4] that was established for the regular case.

Remark 4. There is a $2 + 2\epsilon$-approximation that runs in $O(nlog^2(n)/log(log(n))f(k, n, \epsilon))$ where $f(k, n, \epsilon)$ is the complexity of any $1+\epsilon$ algorithm for the $Q||C_{max}$ problem with k machines and n jobs (for example $f(\epsilon) = O(kn^{\frac{10}{\epsilon^2}})$ using [9]).

Proof. We follow here the same idea as in [4]. Let I denote the instance of the irregular hierarchical problem (where every job fits everywhere). We define an instance I' of the $Q||C_{max}$ problem as follows. For each job J_J of I we associate a job J'_j of I' of processing time $p'_j = w_j$. For each machine M_i of I' we associate a machine M'_i of I' whose speed is m_i. Then, we apply a $PTAS$ (and get a schedule $S' \leq (1+\epsilon)Opt(I')$) on I'. The schedule gives a partition of the jobs of I among the different machines. Let X'_i denote the set of jobs scheduled on M'_i in S', and let X_i denote the corresponding set of jobs (meaning the jobs of same index) in I. Let c'_i denote the completion time on M'_i. We create the schedule S of I as follows. According to the Steinberg's theorem (see Section 2.3), we know that for each i the set X_i can be scheduled in a "box" (created on machine M_i) of size $m_i \times 2c'_i$ as $c'_i = \frac{W(X'_i)}{m_i} = \frac{W(X_i)}{m_i}$. Thus, the makespan of S is lower than $2max_i c'_i \leq 2(1+\epsilon)Opt(I') \leq 2(1+\epsilon)Opt(I)$.

A natural (but hard) question is to fill the gap between the new approximation ratio $\frac{5}{2}$ and the lower bound with a low cost algorithm.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications 15(3), 200 (2001)
2. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C., Wong, P.: Theory and practice in parallel job scheduling. In: JSSPP, pp. 1–34 (1997)
3. Zhuk, S.: Approximate algorithms to pack rectangles into several strips. Discrete Mathematics and Applications 16(1), 73–85 (2006)
4. Ye, D., Han, X., Zhang, G.: On-Line Multiple-Strip Packing. In: Proceedings of the 3rd International Conference on Combinatorial Optimization and Applications (COCOA), p. 165. Springer, Heidelberg (2009)
5. Bougeret, M., Dutot, P.F., Jansen, K., Otte, C., Trystram, D.: Approximation algorithm for multiple strip packing. In: Bampis, E., Jansen, K. (eds.) WAOA 2010. LNCS, vol. 5893, pp. 37–48. Springer, Heidelberg (2010)
6. Pascual, F., Rzadca, K., Trystram, D.: Cooperation in multi-organization scheduling. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 224–233. Springer, Heidelberg (2007)
7. Schwiegelshohn, U., Tchernykh, A., Yahyapour, R.: Online scheduling in grids. In: IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, pp. 1–10 (2008)
8. Steinberg, A.: A strip-packing algorithm with absolute performance bound 2. SIAM Journal on Computing 26, 401 (1997)
9. Hochbaum, D., Shmoys, D.: A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. SIAM J. Comput. 17(3), 539–551 (1988)