Job Scheduling Using successive Linear Programming Approximations of a Sparse Model

Stephane Chretien[1], Jean-Marc Nicod[2], Laurent Philippe[2],
Veronika Rehn-Sonigo[2], and Lamiel Toch[2]

[1] Department of Mathematics, Université de Franche-Comté, Besançon, FRANCE
[2] FEMTO-ST Institute, UMR CNRS / UFC / ENSMM / UTBM, Besançon, France

Abstract. In this paper we tackle the well-known problem of scheduling a collection of parallel jobs on a set of processors either in a cluster or in a multiprocessor computer. For the makespan objective, i.e., the completion time of the last job, this problem has been shown to be NP-Hard and several heuristics have already been proposed to minimize the execution time. We introduce a novel approach based on successive linear programming (LP) approximations of a sparse model. The idea is to relax an integer linear program and use ℓ_p norm-based operators to force the solver to find almost-integer solutions that can be assimilated to an integer solution. We consider the case where jobs are either rigid or moldable. A rigid parallel job is performed with a predefined number of processors while a moldable job can define the number of processors that it is using just before it starts its execution. We compare the scheduling approach with the classic Largest Task First list based algorithm and we show that our approach provides good results for small instances of the problem. The contributions of this paper are both the integration of mathematical methods in the scheduling world and the design of a promising approach which gives good results for scheduling problems with less than a hundred processors.

1 Introduction

Nowadays clusters of computers or large shared memory computers are widely used by many communities such as researchers, universities or industries to speed up their applications. Due to their cost these computing facilities are usually shared between several users and several parallel jobs must be run at the same time on the same platform. The problem of scheduling parallel jobs on clusters without knowing in advance the submission times of user jobs has been widely studied [20]. In this case the scheduling problem is said to be "on-line" [12]. When all characteristics of the jobs are known in advance, the scheduling problem becomes "off-line" and it has been widely studied for sequential jobs [13] and for parallel jobs [8, 11].

The "off-line" problem considered here depends on the job characteristics. In the literature one distinguishes three kinds of parallel jobs. Rigid jobs [16] are performed with the number of processors originally required. Moldable jobs introduced by Turek et al. in [18] may run with different numbers of processors but cannot change their allocation after their start. Malleable jobs [9] can modify

the number of allocated processors during their execution. The rigid job model can easily be used in most of the cases of parallel jobs. The two other models however need an interaction between the application and the scheduler to define the number of allocated processors. This is for instance the case of applications developed with the *Bulk Synchronous Parallel* (BSP) model introduced in [19] that can be run as moldable jobs. Processor virtualization however could be a solution to transparently make standard parallel applications moldable as presented in [17]. Applying virtualization to malleable jobs is probably more difficult as it would need to use virtual machine migration. For these reasons we focus on rigid and moldable jobs.

The problem of scheduling several parallel rigid and moldable jobs on homogeneous computing resources has been shown to be NP-Hard respectively in [11] and [8]. Several previous works have already tackled the issue of providing heuristics that give efficient sub-optimal solutions. In [2] static scheduling of rigid parallel jobs for minimizing the makespan is studied and in [1] for minimizing the sum of the completion time of each job. In [10], Dutot et al. consider the problem of scheduling moldable jobs with the objective of minimizing the makespan. The authors present experimental results where the well-known Largest Task First (LTF) algorithm is the best for the makespan objective.

The contribution of this paper is a novel approach for scheduling a collection of rigid or moldable jobs using successive LP approximations based on the gradient operator. To the best of our knowledge there is no existing work using this promising approach based on the sparse recovery problem in statistics domain.

The remainder of the paper is organized as follows. In Section 2 we describe the problem and the model of moldable jobs. In Section 3 we present the sparsity promoting penalization as well as linear approximation principles. Then, in Section 4 we present how to adapt this method to our scheduling problem. In Section 5 we compare our technique with the algorithm developed by Dutot et al. in [10] and show experimental results to assess the performance of our approach, and finally we conclude and give future work directions in Section 6.

2 Framework

In this section we formally define the targeted framework and the problem. We consider the problem of scheduling a collection of n independent parallel jobs. We tackle both cases of rigid and moldable jobs.

The jobs are run on a homogeneous cluster of distributed computing nodes or on a shared memory multiprocessor or multicore computer. In a cluster each node is made up of identical processors which are in turn made up of identical cores. The scheduling policy used on most clusters does not pay any attention to the exact distribution of the cores allocated on the nodes provided that the job is parallel. For this reason, in this paper, we will only consider the number of allocated cores, assimilated to processors and called Processing Elements (PEs). The results can then be applied either on clusters or on multiprocessor-multicore computers. In the remainder of the paper m denotes the number of available PEs in the execution platform.

Rigid jobs are defined by an execution time and a static number of requested PEs, i.e., the job cannot be run on neither more nor less PEs than originally requested. Each rigid job i is defined by its number of requested PEs $reqproc_i$ and its duration $reqtime_i$.

Moldable jobs can be run on a different number of PEs or cores but this number is fixed at the job execution start and cannot change during the execution. The considered moldable jobs respect the model defined in [10]. Let $reqtime_i$ be the duration of job i which requires at most $reqproc_i$ PEs. Let $t_i(n)$ be the duration of the job i if n PEs are allocated for job i. The relation between the duration of a job i and its number of allocated PEs is stated as:

$$\forall i, \ \forall n \leq reqproc_i, \ t_i(n) = \left\lceil \frac{reqproc_i}{n} \right\rceil reqtime_i$$

Given this framework our objective is to minimize the makespan of the schedule. According to the $\alpha|\beta|\gamma$ (platform | application | optimized criterion) classification of scheduling problems given by Graham in [15], the above problem is denoted by $P|\text{parallel jobs}|C_{max}$.

3 Sparsity promoting penalization with successive linearizations

The optimization method presented in the paper relays on two steps. First we formulate the problem as an integer linear program, then we relax it and apply the sparsity promoting penalization which tries to find almost integer solutions. As the sparsity promoting penalization implies to minimize a non linear objective function we use successive LP approximations to linearize it. In this section we detail the main steps of the method.

3.1 Sparsity promoting penalization

Recent works on the sparse recovery problem in statistics and signal processing have brought to general attention the fact that using non-differentiable penalties such as the ℓ_p norm can be an efficient ersatz to combinatorial constraints in order to promote sparsity. This approach for constructing continuous relaxations to hard combinatorial problems is a key ingredient in e.g., the new field called Compressed Sensing which originated in the work of Candès, Romberg and Tao [3]. Donoho [7] showed that finding the sparsest solution to an under-determined system of linear equations may sometimes be equivalent to finding the solution with smallest ℓ_1-norm. This discovery lead to a intense research activity in the recent years focusing on finding weaker sufficient conditions on the linear system under which it is possible to prove this equivalence. It was found in particular that for matrices satisfying certain incoherence conditions (implying that the columns of the associated matrix are almost orthogonal), the equivalence between finding the sparsest and the least ℓ_1 norm solution holds for systems with a number of unknowns to the order of exponential of the number of equations. Other non-differentiable penalties have also been proposed in order to increase the performance of sparse recovery procedures. Candès, Wakin and Boyd proposed an iterative reweighted ℓ_1 procedure in [4]. In our setting, the standard

ℓ_1 relaxation is not suitable. Indeed, as will be detailed in the sequel (e.g. equation 1 below), our constraints will always imply that the ℓ_1 is constant. A more appropriate sparsity promoting penalization in this case is the ℓ_p-quasi-norm relaxation, for $p \in (0,1)$. This corresponds to minimizing $\|x\|_p := (\sum_k x_k^p)^{1/p}$ instead of $\|x\|_1$, under the same design constraints. Such a non–convex relation was successfully implemented in, e.g. [6].

3.2 Linear and conic approximation

In physics and mathematics a function f is often approximated with a linear formulation at point x_0, if f is differentiable at point x_0. The gradient of a function with several parameters ($f : \mathbb{R}^n \to \mathbb{R}$), noted ∇f, is the vector whose components are equal to derivatives of f with respect to the parameters. Taylor's expansion gives

$$f(x + h) = f(x) + \langle \nabla f, h \rangle + o(h)$$

where x and h belong to $\mathbb{R}^n$, and $\langle \ \rangle$ represents the dot product.

In cases such as $x \mapsto \|x\|_p$, where f is non-differentiable, it is still possible linearize by using the appropriate generalization of the gradient, called the Clarke-subdifferential. In simple words, a non-differentiable function may have several tangents in a generalized sense and the Clarke-subdifferential, denoted by $\partial f(x)$, is the set of all such generalized tangents. The nonsmooth counterpart to Taylor's expansion is given by

$$f(x + h) = f(x) + \sup_{g \in \partial f(x)} \langle g, h \rangle + o(h)$$

.

In order to implement our ℓ_p-based relaxation, we will implement successive linearizations on a standard linear programming solver.

4 Applying the method on the job scheduling problem

In this section we apply the method on the job scheduling problem. First it implies to define a sparse representation of the problem then we apply the two steps of sparsity promoting penalization and linear approximation.

4.1 Formulation as an integer linear program

In the defined framework a solution to the scheduling problem must provide at least the start time of the jobs for the rigid jobs as their duration and the number of used PEs are constants of the problem. For the moldable jobs, the duration depends upon the number of PEs that are allocated to the jobs. So the scheduled jobs are characterized by their start time and the number of allocated PEs and the duration of the job is determined as soon as this number of allocated PEs is determined. We call "configuration" of a job the number of allocated PEs. We call "position" of a job, its position determined by its start time in a discrete time scale. Finally, we call "slot" the couple (configuration, position).

Let us create a list of slots (configurations, positions) for each job. The idea is to create a vector x_i for each job i. Each component $x_{i,j}$ of the vector x_i is a

binary variable which indicates whether slot j of job i is chosen or not. Then we fix a time horizon T and we let a linear program find a solution. We iteratively reduce the time horizon T until the linear program cannot find a solution any more.

The following constant values are defined to formulate the problem:

– $proc_{i,j}$: the number of PEs for the configuration j of job i
– $nconf_i$: the number of all possible configurations for job i (for rigid jobs i, $nconf_i = 1$)
– $nslots_i$: the number of all possible slots for job i
– $C_{i,s}$: the configuration index of job i used in the slot s of job i
– $run_{i,s,t}$ indicates whether in the slot s the job i is running at time t.

Then we define the binary variable $x_{i,s}$ which indicates whether slot s of job i is chosen or not. For each job i, we note x_i the vector whose components are the values $x_{i,s}$ and we define a vector x which is equal to the concatenation of the n vectors x_i of every job i, $1 \leq i \leq n$.

The problem can be formulated as an integer linear program. Since we only have to determine whether a feasible solution exists or not for a given time horizon T, we only need the constraints to be respected. That is why we set all the coefficients of the variables in the objective function to 0. The problem is stated as: *"find a feasible solution which respects the following linear constraints:"*

$$\forall 1 \leq i \leq n, \quad \sum_{s=1}^{s=nslot_i} x_{i,s} = 1 \tag{1}$$

$$\forall 1 \leq t \leq T, \sum_{i=1}^{i=n} \sum_{s=1}^{s=nslot_i} x_{i,s} \times run_{i,s,t} \times proc_{i,C_{i,s}} \leq m \tag{2}$$

Constraint 1 imposes the unicity of the chosen slot s on each job i. Constraint 2 means that at each time t, the set of all running jobs does not consume more than the m available PEs in the considered cluster.

4.2 Relaxation via sparsity promoting penalization

The solution of the Integer formulation of the problem cannot be found in polynomial execution time. So we make a relaxation of it. We transform all binary variables $x_{i,s}$ into rational variables and with $0 \leq x_{i,s} \leq 1$. Since vector x – the concatenation of x_i vectors – indicates which slots are chosen, we are tempted to strongly enforce its sparsity. In fact, vector x must have exactly n "1" and many "0". Thus, we legitimately expect the binary constraints to be naturally recovered by imposing sufficient sparsity. Notice that the proposed constraints impose that the sum of the components of x is equal to one jobwise. Since the components are positive, this implies that the ℓ_1 norm is equal to one jobwise, which explains why minimizing the ℓ_1 norm for promoting sparsity is unfortunately useless in the present context. In order to overcome this difficulty, we

---
Algorithm 1: A successive LP scheme
---
1 $lb \leftarrow$ lower bound of makespan
2 $sched \leftarrow$ compute a schedule with LTF ; $listMakepsan \leftarrow makepsan(sched)$;
　　$T \leftarrow listMakespan$; $end \leftarrow false$; $incT \leftarrow false$;
3 **while** $T > lb$ *and not* end **do**
4　　$proc \leftarrow$ compute the configurations $(\mathcal{J})$; $run \leftarrow$ compute all possible slots $(\mathcal{J}, m, T)$;
　　　　$iter \leftarrow 1$; $found \leftarrow false$
5　　$\forall i, k, x_{i,k}^{(iter)} \leftarrow 0$
6　　**while** $iter < maxIter$ *and not* $found$ **do**
7　　　　set the objective function of $\mathcal{LP}(\mathcal{J}, m, T, proc, run)$ to
　　　　　　$\sum_i^{|\mathcal{J}|} f(x_i^{(iter)}) + \langle \nabla f(x_i^{(iter)}), x_i - x_i^{(iter)} \rangle$
8　　　　$x \leftarrow$ execute $\mathcal{LP}(\mathcal{J}, m, T, proc, run)$
9　　　　**if** $\forall i$, x_i *contains exactly one "1"* **then**
10　　　　　　$sched \leftarrow$ convert into schedule $(x, proc, run)$
11　　　　　　$T \leftarrow makespan(sched)$
12　　　　　　$T \leftarrow T - 1$
13　　　　　　$found \leftarrow true$
14　　　　　　**if** $incT = true$ **then**
15　　　　　　　　$end \leftarrow true$

16　　　　$\forall i, k, x_{i,k}^{(iter)} \leftarrow x_{i,k}$; $iter \leftarrow iter + 1$
17　　**if** *not* $found$ **then**
18　　　　**if** $T = listMakespan$ **then**
19　　　　　　$incT \leftarrow true$
20　　　　**if** $incT = true$ **then**
21　　　　　　$T \leftarrow T + 1$
22　　　　**else**
23　　　　　　$end \leftarrow true$

24 **return** $sched$
---

chose to minimize the ℓ_p norm non-convex function

$$f(x) = \sum_i \|x_i\|_p \tag{3}$$

under constraints (1) and (2) for $p \in]0, 1[$.

4.3 Successive LP approximation scheme

We now apply successive LP approximation schemes to linearize the problem. Let $f_i(x_i) = \|x_i\|_p$, for all jobs i. Thus, $f = \sum_i f_i$. We use the value of each variable computed during the previous iteration. We will use the following arbitrary choice $g \in \partial f$ among all possible subgradients of f:

$$g_{i,j} = \begin{cases} x_{i,j}^{p-1} \times f_i\left(x_i\right)^{1-p} & \text{if } x_{i,j} \neq 0 \\ \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The method is implemented in Algorithm 1. It starts with any initial value e.g. the zero vector. First we compute a lower bound of the makespan at line 1, which is equal to the maximum between the duration of the longest job and $\frac{\sum_i reqproc_i \times reqtime_i}{m}$. The time horizon T is set to the makespan of the LTF

list algorithm. If the linear program $\mathcal{LP}$ finds a satisfactory solution (line 9), it reduces the time horizon (line 12) until it cannot (line 23) before $maxIter$ iterations. If it does not find a satisfactory solution with $T = Listmakespan$ before $maxIter$ iterations (line), it increases the time horizon T (line 21). For a given time horizon T, it iteratively updates the objective function of the linear program (line 7) according to the subgradient-based Taylor approximation rule of the sparsity promoting penalization.

4.4 Improving the algorithm efficiency

During the experiment step of our work *a problem* appeared in the linear resolution. Satisfactory solutions for Algorithm 1 are only detected (at line 9) if all jobs i have their vector x_i with exactly one "1" as the algorithm is designed to find exclusively exact solutions. In fact, for a given time horizon T, the successive linear approximations manage to find a schedule for most of the jobs of the collection but it let few jobs j of the collection with fuzzy schedules. That is to say, vectors x_i contain exactly one "1" while vectors x_j do not. In this case the algorithm often continues to iterate, even if x_j is close to 1, until $maxIter$ is reached without being able to find a solution. This leads to longer computing times for the algorithm while giving inefficient solutions.

So we modify Algorithm 1 and its detection criterion at line 9 as follows: when a valid rational schedule is found we keep the exact schedule for jobs i whose x_i have exactly one "1" and we schedule the rest of the jobs for which the linear program gives fuzzy schedules with the LTF list algorithm. If a solution shorter than the time horizon T is found, then the $found$ variable is set to $true$ otherwise we continue to iterate.

5 Simulation and results

In this section we present the results obtained on the two versions of the algorithm and we compare them to the well-known Largest Task First algorithm. We assess both cases of rigid and moldable jobs. Notice that the problem we propose to solve is nonconvex and very high dimensional. Moreover, no theoretical guarantee for convergence of the proposed iterative procedure is available and it is well known that minimizing an ℓ_p quasi-norm, $0 < p < 1$, is NP-hard already. On the other hand, various non-convex ℓ_p-based strategies have been successfully used for promoting sparsity in the literature. Despite the current lack of appropriate theoretical foundation, in most reported experiments the ℓ_p-based approach managed to reach a local solution significantly superior to the ℓ_1 minimizer for, e.g., the Compressed Sensing reconstruction problem [6]. The goal of this section is to show that such a good performance can also be observed for the studied scheduling problem. Notice that we did not optimize the computational aspects of the problem, in particular, we made no use of the very special properties of the constraint matrix. This explains why the computing time is currently much higher than what could be obtained after a careful design of the algebraic aspects of our algorithms.

5.1 Experimental settings

Carrying out real experiments on clusters is difficult: experiments are not reproducible and may be long. Furthermore a cluster is expensive and meant to be used for calculations while experiments may monopolize it. For these reasons, we have developed a simulator of a homogeneous cluster based on a master/slave architecture. This simulator is also meant to check schedules obtained by the different algorithms. The simulator is implemented using SimGrid [5] and its MSG API. It takes a workload as input and it gives a schedule as output.

To simulate the job collection, we use synthetic workloads generated with uniform distributions. The parameters associated with a workload is the job granularity, the ratio of the duration of the longest job over the duration of the shortest one.

5.2 Assessing performance of Algorithm 1

In a first set of experiments the simulations have been run with a ℓ_p norm where $p = 0.1$, $maxIter$ is set to 15000 in the algorithm and the machine is made up of 64 PEs. We have scheduled a collection of 60 jobs and, for each number of jobs in the collection, we performed 40 experiments to compute an average value of the ratio of the makespan over the lower bound. The results where disappointing: they were far from the optimal and very time consuming.

So we ran another set of experiments with less jobs, ℓ_p norm where $p = 0.1$, $maxIter$ set to 15000. The machine is made up of 32 PEs and the number of PEs requested by each job is uniformly chosen between 1 and 8. The granularity is set to 25. For each number of jobs in the collection we perform 20 experiments, then we remove the best and the worse results in order to reduce the deviation, and we compute an average of the ratio of the makespan over the lower bound.
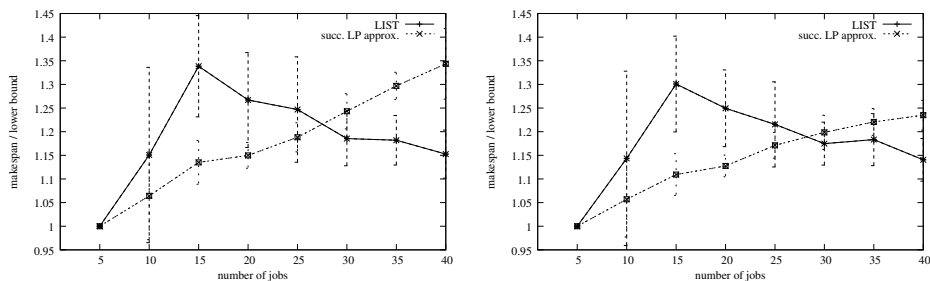
(a) rigid jobs   (b) moldable jobs

Fig. 1: Performance comparison 32 PEs

Figure 1a shows the ratio of the makespan over the lower bound against the number of rigid jobs, while Figure 1b shows this ration for moldable jobs. In the figures the algorithms are noted *succ. LP approx* for our algorithm and *LIST* for

the LTF implementation. The figures also show the standard deviation σ: the height of a vertical line is equal to 2σ.

We can note that for less than 25 jobs the successive LP approximation algorithm gives better results than the LTF algorithm and with more than 25 jobs the latter outperforms successive LP approximation. Note that after 40 jobs the performance ratio of LTF quickly tends toward 1.1 which means, on the one hand, that it probably finds most of the time the optimal solution and, on the other hand, that it is difficult to find better solutions. Moreover, with more than 25 jobs, the problem becomes so complex that the successive LP approximation algorithm must increase its T to find a solution. Under this threshold the maximum gain is about 15% for 16 moldable or rigid jobs. Furthermore the standard deviation of the experiments with our new approach is less than the standard deviation of LTF. We can easily understand that for 5 jobs the optimal is found due to the experimental settings: the number of PEs that each job requires is uniformly chosen between 1 and 8. As a consequence, all jobs may start at time 0. That also explains the peak with 15 jobs which do not necessarily start at time 0. We can also notice that for rigid jobs and moldable jobs the successive LP approximation algorithm has the same behavior, that is to say, when the number of jobs increases, the ratio of the makespan over the lower bound increases.
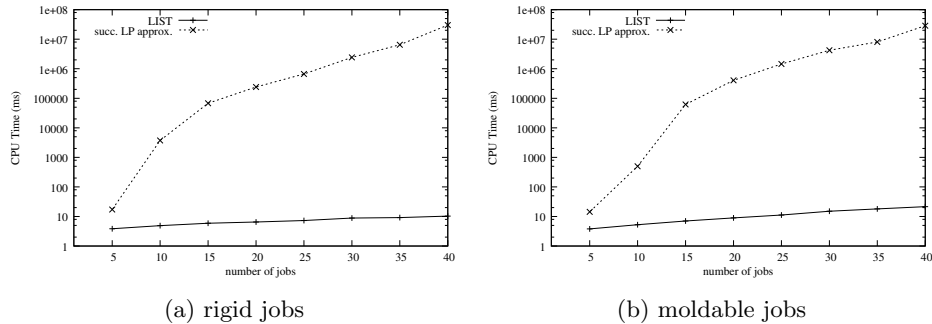

(a) rigid jobs   (b) moldable jobs

Fig. 2: Compute time for 32 PEs

As we can see in Figure 2a and Figure 2b, Algorithm 1 is very time consuming with both rigid jobs and moldable jobs compared to the LTF algorithm. Note however that for 15 jobs, in the case where gives the best results, the time taken by the LP approximation is not more than 1.5 minutes which is still reasonable. We assess the performance of the improved version in the following section.

5.3 Performance of the improved algorithm

To assess the improved algorithm, we performed experiments with two simulated machines made up of 64 and 128 PEs. The number of PEs requested by each job is uniformly chosen between 1 and 16 for the machine with 64 PEs, and between 1 and 32 for the machine with 128 PEs. Granularity is set to 25 for the

machine with 64 PEs and to 10 for the machine with 128 PEs. We set $p = 0.1$ and $maxIter = 200$. For each number of jobs in the collection, we perform 40 experiments.
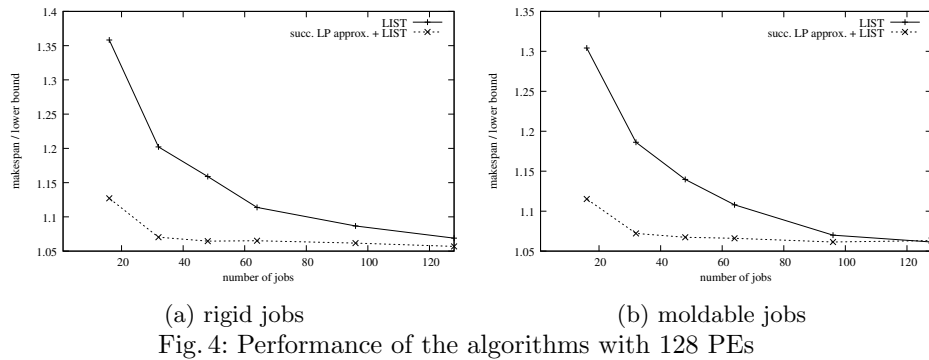

(a) rigid jobs   (b) moldable jobs

Fig. 3: Performance of the algorithms with 64 PEs

(a) rigid jobs   (b) moldable jobs

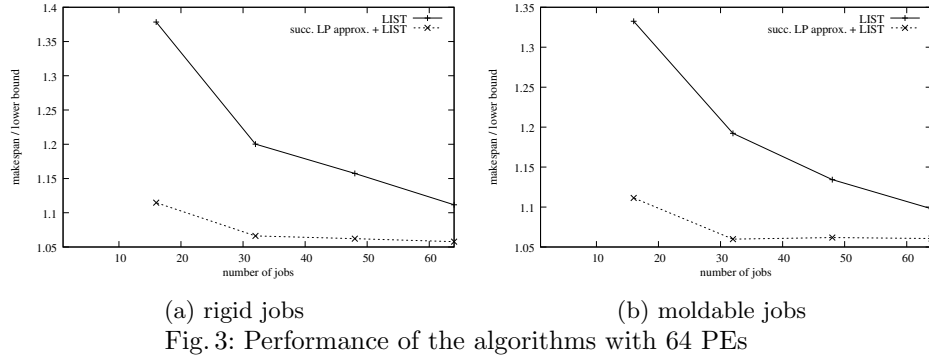Fig. 4: Performance of the algorithms with 128 PEs

Figure 3a shows the ratio of the makespan over the lower bound against the number of rigid jobs in a cluster of 64 PEs, while in Figure 3b we consider scheduling moldable jobs. The performances of the new approach are better than LTF for moldable and rigid jobs, and better than the unmodified algorithm. Figure 4a and Figure 4b give the results for a machine with 128 PEs.

We can note that in the four cases the performance ratio between LTF and our approach is up to 20%. The results obtained with a 128 PEs machine show an improvement for the LTF algorithm compared to 64 case while the behavior of the new algorithm is quite similar. This is probably because our solution is very close to (if not at) the optimal solution and nothing more can be gained.

We have also recorded some statistics data after each execution of the linear program. On average with 64 PEs and 16 jobs almost 75% of jobs have exact schedules, while with 64 jobs 50% of them have exact schedules. We notice that when the number of jobs to schedule increases the number of exact schedules

found by the linear program decreases. We get the same behaviour with 128 PEs: on average with 128 PEs and 16 jobs almost 80% of jobs have exact schedules, while with 128 jobs 60% of them have exact schedules.

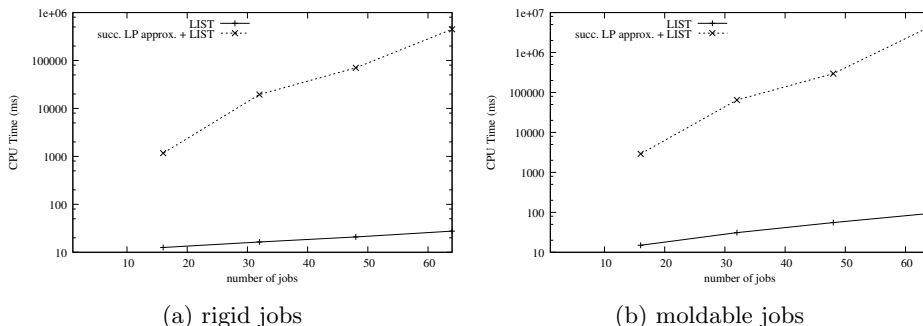|	
(a) rigid jobs	(b) moldable jobs

Fig. 5: CPU Time consumed to compute a schedule with 64 PEs

Figures 5a and 5b show the time spend by the two algorithms. We notice that the hybrid algorithm is less time consuming than the original algorithm but still consumes more time than the LTF algorithm.

6 Conclusion and Future Work

In this paper, we assess the use of successive linear programming approximations of a sparse model for job scheduling. This method is applied on clusters to schedule rigid and moldable jobs. Experimental results show that the pure successive LP approximation only gives good performances regarding the makespan for scheduling up to dozens jobs on a machine with dozens PEs. In contrast, a variant associated with LTF gives good results for bigger instances with up to a hundred jobs on machines with up to a hundred PEs. This variant is a good alternative to the LTF algorithm and provides a significant improvement of the schedules for the range of machine size where the LTF algorithm is less efficient.

For future work we plan to implement the Split Bregman Method [14] to speed up the solving time and try other relaxations of the linear program. We also plan to use a multi-level scheduling approach for which we distinguish small jobs and large jobs. We then apply our method on different collections of jobs.

An important part of the simulations has been run thanks to the computing facilities of the *Mésocentre de Calcul de Franche-Comté* in Besançon, France.

References

1. Foto N. Afrati, Evripidis Bampis, Aleksei V. Fishkin, Klaus Jansen, and Claire Kenyon. Scheduling to minimize the average completion time of dedicated tasks. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, FST TCS 2000, pages 454–464, London, UK, 2000.

2. Abdel Krim Amoura, Evripidis Bampis, Claire Kenyon, and Yannis Manoussakis. Scheduling independent multiprocessor tasks. *Algorithmica*, 32(2):247–261, 2002.

3. E. J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *Information Theory, IEEE Transactions on*, 52(2):489–509, February 2006.

4. Emmanuel J. Candes, Michael B. Wakin, and Stephen P. Boyd. Enhancing Sparsity by Reweighted L1 Minimization. *Journal of Fourier Analysis and Applications*, 14(5):877–905, December 2008.

5. H. Casanova, A. Legrand, and M. Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *UKSIM '08*, pages 126–131, 2008.

6. Rick Chartrand and Wotao Yin. Iteratively reweighted algorithms for compressive sensing. In *33rd International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2008.

7. David L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52:1289–1306, 2006.

8. Pierre-François Dutot, Lionel Eyraud, Grégory Mounié, and Denis Trystram. Bi-criteria algorithm for scheduling jobs on cluster platforms. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '04, pages 125–132, New York, NY, USA, 2004.

9. Pierre-François Dutot and Denis Trystram. Scheduling on hierarchical clusters using malleable tasks. In *SPAA'01*, pages 199–208, 2001.

10. Pierre-François Dutot, Alfredo Goldman, Fabio Kon, and Marco Netto. Scheduling moldable BSP tasks. In *11th JSSPP*, volume 3834 of *LNCS*, pages 157–172, Cambridge, MA, USA, 2005.

11. Dror G. Feitelson. Job scheduling in multiprogrammed parallel systems. Research Report RC 19790 (87657), IBM T. J. Watson Research Center, 1997.

12. Dror G. Feitelson and Ahuva W. Mualem. On the definition of "on-line" in job scheduling problems. Technical report, SIGACT News, 2000.

13. Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

14. Tom Goldstein and Stanley Osher. The split bregman method for l1-regularized problems. *SIAM J. Img. Sci.*, 2:323–343, April 2009.

15. Ronald Lewis Graham and al. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, pages 287–326, 1979.

16. Uri Lublin and Dror G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63:2003, 2001.

17. Jean-Marc Nicod, Laurent Philippe, Veronika Rehn-Sonigo, and Lamiel Toch. Using virtualization and job folding for batch scheduling. In *ISPDC'2011, 10th Int. Symposium on Parallel and Distributed Computing*, pages 39–41, Cluj-Napoca, Romania, July 2011. IEEE Computer Society Press.

18. John Turek, Joel L. Wolf, and Philip S. Yu. Approximate algorithms scheduling parallelizable tasks. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, SPAA '92, pages 323–332, New York, NY, USA, 1992. ACM.

19. Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33:103–111, August 1990.

20. Deshi Ye and Guochuan Zhang. On-line scheduling of parallel jobs in a list. *J. of Scheduling*, 10:407–413, December 2007.