A Bi-Objective Scheduling Algorithm for Desktop Grids with Uncertain Resource Availabilities

Louis-Claude Canon, Adel Essafi, Grégory Mounié, Denis Trystram

A Bi-Objective Scheduling Algorithm for Desktop Grids with Uncertain Resource Availabilities

Louis-Claude[1] Canon, Adel Essafi[1,2], Grégory Mounié[1], and Denis Trystram[1,3]

[1] Grenoble Institute of Technology, 51 avenue Jean Kuntzmann, 38330 Montbonnot Saint Martin, France
[2] UTIC, Ecole Supérieure des Sciences et Techniques de Tunis 5, Avenue Taha Hussein, B. P. : 56, Bab Menara, 1008 Tunis, Tunisia
[3] Institut Universitaire de France

Abstract. In this work, we consider the execution of applications on desktop grids. Such parallel systems use idle computing resources of desktops distributed over the Internet for running massively parallel computations. The applications are composed of workflows of independent non-preemptive sequential jobs that are submitted by successive batches. Then, the corresponding jobs are executed on the distributed available resources according to some scheduling policy.

However, most resources are not continuously available over time since the users give their idle CPU time only for some time when they are not using their desktops. Moreover, even if the dates of unavailability periods are estimated in advance, they are subject to uncertainties. This may drastically impact the global performances by delaying the completion time of the applications.

The aim of this paper is to study how to schedule efficiently a set of jobs in the presence of unavailability periods on identical machines. In the same time, we are interested in reducing the impact of disturbances on the unavailability periods. This is achieved by maximizing the *stability* that measures the distance between the makespan of the disturbed instance over the initial one. Our main contribution is the design of a new parametrized algorithm and the analysis of its performance through structural properties. This algorithm reduces the impact of disturbances on availability periods without worsening too much the makespan. Its interest is assessed by running simulations based on realistic workflows. Moreover, theoretical results are obtained under the assumption that the size of every availability interval is at least twice the size of the largest job.

Keywords: Scheduling; Availability Constraints; Uncertainty; Stability.

1 Introduction

1.1 Context and motivation

Today, many kind of parallel platforms are available for running applications. In this work, we focus on desktop grids, which gather idle computing resources of usual desktops distributed over the Internet for running massively parallel computations. Such systems provide a very large computing power for many applications issued from a wide range of scientific domains (including, protein folding [14], gravitational physics [19], etc.).

The applications are composed of workflows of sequential jobs that are submitted by successive batches to a particular user interface machine. Then, the corresponding jobs are transferred to be executed on the distributed available resources according to some scheduling policy. However, usually the resources are not continuously available over time since the users give their idle CPU time only for some time when they are not using their desktops.

Moreover, even if the dates of unavailability periods are estimated in advance, they are subject to uncertainties. This may drastically impact the global performances by delaying the completion time of the application.

In this paper, we study how to schedule efficiently a set of jobs (which corresponds to minimize the makespan) in the presence of unavailability periods. In the same time, we are interested in reducing the impact of disturbances on the unavailability periods. The corresponding objective is the *stability* that measures the ratio between the makespan of the disturbed instance over the initial one [3]. To the best of our knowledge, there is no work studying scheduling with unavailability periods under uncertainties.

1.2 Contributions

The first contribution of this work is to investigate the problem of scheduling with unavailabilities from the view point of studying the impact of uncertainties on the availability periods. Our main contribution is the design of an algorithm and the analysis of its performances through structural properties. It is based on the concept of *slacks* placed just before the unavailability periods that prevent jobs to be delayed. The lengths of the slacks are parametrized by the types of jobs allocated on the available intervals. Then, the good behavior of the algorithm is assessed by running simulations derived from actual workflows of BOINC [1].

The proposed methodology should be useful for solving other scheduling problems with various characteristics like failures or estimated energy consumption.

1.3 Organization of the paper

The paper is organized as follows. We first recall the most significant related works in Section 2. We distinguished between the works dealing with scheduling under availability constraints and the main existing approaches for studying

scheduling in a context with uncertainties. Section 3 is devoted to the description of the computation model and the main notations. Then, we present in Section 5 the algorithm and its worst-case analysis. Before concluding, we present experiments in Section 6 based on simulations on actual workflows and availability constraints.

2 Related works

In this section, we recall briefly the most significant works related to our problem. We investigate successively each of both sides of the problem, namely scheduling with unavailability constraints and scheduling under uncertainties.

2.1 Scheduling with unavailabilities

First, notice that most of the approaches used to solve the problem of scheduling with unavailabilities are based on the well-known LPT rule (Largest Processing Times). Lee introduced the problem of scheduling independent jobs with non-simultaneous available times in [15]. This corresponds to scheduling jobs when all the unavailabilities are at the beginning. The main result was to establish that the performance of LPT is bounded by 3/2. He also proposed a modified version of LPT with an improved performance of 4/3. A more general problem was studied in [16] for any pattern of availability. Lee showed that the problem cannot be polynomially approximable if no restrictions are done on the availabilities. However, the performance of LPT is bounded by $\frac{m+1}{2}$ when at least one machine is always available and at most one unavailability period per machine is allowed.

In [9], Hwang and Chang analyzed the problem when no more than the half of the machines are unavailable simultaneously. Under this condition, the performance of LPT is bounded by 2. This result was generalized in [10] as follows: if at most λ (ranging from 1 to $m-1$) processors are allowed to be unavailable simultaneously, then LPT generates a schedule whose performance is bounded by $1+\frac{1}{2}\lceil\frac{m}{m-\lambda}\rceil$.

Liao et al. [18] studied the restriction of the problem on two machines where each machine has one fixed unavailability period. They proposed an optimal exponential-time algorithm. A variant of this particular problem was studied in [20] where the first machine is always available whereas periodic unavailabilities are scheduled on the second. All the unavailabilities have the same duration and all the availabilities have also the same duration. In this case, the performance of LPT is $\frac{3}{2}$ and 2, respectively, for the offline and the online context.

The problem where one machine is always available and with an arbitrary number of unavailabilities on the other processors was analyzed in [4]. It admits no FPTAS, however, a Polynomial Time Approximation Scheme (PTAS) based on the multiple knapsack was designed. A simple list strategy was also proposed.

Notice that all the above approaches are related to sequential jobs. Eyraud et al. studied the problem of scheduling with unavailabilities for parallel rigid jobs [6]. They proved that there is no approximation algorithm in the general

case, and they proposed an approximation algorithms for non-increasing unavailability patterns. Moreover, for the problem with restricted unavailabilities, lower and upper bounds were provided for a general list algorithm.

2.2 Scheduling under uncertainties

Solving scheduling problems with uncertain data has received recently a great attention. There exists a lot of possible approaches depending on the target problem and the desired objectives. A nice survey of scheduling problems was compiled by Billaut et al. [3]. They discuss several complementary approaches from pure pro-active methods (sensitivity analysis), pure on-line strategies and semi on-line methods (flexibility). We focus on this last approach which builds an efficient solution on estimated data and allow simple correction mechanisms at run-time.

Numerous publications are similar to this article, proposing pro-active heuristics based on slacks. In [7], the authors investigate preemption. In [12, 13], the authors explore stochastic resource breakdown.

We concentrate on a problem in which interrupted jobs are restarted from the beginning, without migration, but the unavailability constraints can only advance forward in time. Hence, in our case, the reaction and uncertainties are restricted.

3 Models

We present in this section the model of execution that defines the workload and the platform characteristics. Then, without loss of generality, the disturbances are restricted to early shifts of the unavailability periods. Finally, we express formally the problem and define the objectives.

3.1 Model of execution

In the context of desktop grids, the workload consists of a set of independent jobs. The processing time of the j-th job is p_j. These jobs do not have release or due dates and cannot be preempted.

The platform is composed of m identical machines that are indexed by i. Each of these machines possesses a set of unavailability constraints. We define an interval as an availability period followed by an unavailability period. As intervals are indexed by k, the starting time of unavailability period k on processor i is denoted s_i^k and has a duration u_i^k (see Figure 1). Hence, this period ends at $e_i^k = s_i^k + u_i^k$. Additionally, the duration of the availability period that precedes unavailability k is a_i^k. Finally, the first unavailability period on a processor starts at s_1^i and $e_0^i = 0$.

We denote by λ the number of processors that have no unavailability constraints that are called *free processors*.

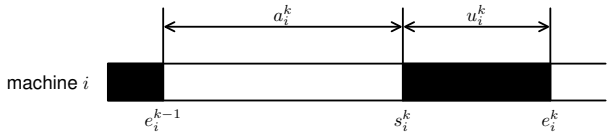

Fig. 1. Representation of interval k on machine i, which contains an availability period followed by an unavailability period with starting dates e_i^{k-1} and s_i^k.

3.2 Model of disturbances

Let δ_i^k be the disturbance that impacts unavailability k on processor i. As we consider that unavailability periods may come earlier, we denote the disturbed unavailability starting time $\tilde{s}_i^k = s_i^k + \delta_i^k$ (by convention, we denote $\tilde{x}$ the disturbed value of the variable x). Unavailability periods cannot overlap, therefore the earliness is limited by the duration of the previous availability period (*i.e.*, $-a_i^k \leq \delta_i^k \leq 0$). Moreover, as it can be seen on Figure 2, only the starting dates are disturbed.

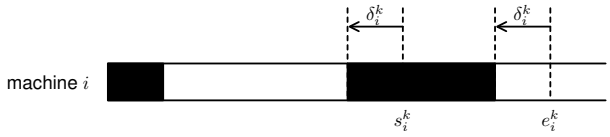

Fig. 2. Unavailability k may start and end earlier due to the disturbance δ_i^k.

3.3 Problem definition

The objective is to generate a schedule given a set of jobs and a set of machines with their unavailability constraints. A schedule is specified by an allocation function $\pi(i, k)$ that gives the set of jobs to be executed during each k-th interval on each i-th processor (jobs are then executed by non-increasing processing times on each interval).

In a disturbed scenario, each unavailability starting date comes early according to our model of disturbance. Moreover, the execution of a schedule is dynamically adapted by using two rules:

- each interrupted job is re-executed as soon as possible without delaying the starting dates of the jobs that follow on the same processor;
- when a processor becomes idle, it starts the execution of its next allocated job.

Assessing the quality of a schedule is done through two objectives: the efficiency and the ability to cope with uncertainties.

We evaluate the first objective by measuring the reference makespan [17] of a schedule, *i.e.*, the makespan when there is no disturbance. It is classically denoted by $C_{\max} = \max_j C_j$ (where C_j is the end date of job j in a given schedule) and the optimal makespan for a given instance is denoted by $C^*_{\max}$.

The second objective is called the *stability*. It is defined as the ratio between the highest disturbed makespan (*i.e.*, the worst makespan among all the possible disturbed scenarios) and the reference makespan, *i.e.*, $S = \frac{\widetilde{C}_{\max}}{C_{\max}}$. This objective represents the insensitivity of a schedule to the disturbances. A schedule is said to be *stable* if $S = 1$.

The problem consists in finding a schedule with minimum values of makespan and stability.

4 Analysis of the stability

In this section, we present the main flexibility mechanism used for coping with uncertain availabilities. The idea is to reserve idle times before the unavailability period to absorb the effect of the disturbances. Idle time, or *slack*, is used for re-executing interrupted jobs in such a way that the reference makespan is delayed the least possible.

Definition 1 (Slack). *The amount of idle time d_i^k preceding unavailability k on processor i is called the* slack.

Definition 2 (Slack rule). *Each slack must be greater or equals to the maximum size of the jobs assigned to its interval,* i.e.*, for each interval k on each machine i:*

$$d_i^k \geq \max_{j \in \pi(i,k)} p_j$$

Proposition 1. *Every schedule based on the slack rule is stable if there is no job scheduled on the last availability period starting before the makespan on each machine (*i.e.*, on the last availability period k for which e_i^k is lower or equals to the makespan).*

Proof. The proof is straightforward since any schedule based on the slack rule is stable within each interval (*i.e.*, in the worst case, the unavailability interrupts the longest job, which can then be absorbed by the slack on this interval).

This is no longer true for the intervals on which a job may be delayed after the makespan. It is the case when an interval starts before the makespan and ends after it. Notice that this condition is strong and may be relaxed in practice. □

Note that a schedule cannot be stable when there is no free processor: if $\lambda = 0$, the job that terminates at the same time as the makespan may be interrupted and re-executed after the reference makespan. In this case, Proposition 1 is violated because this job is necessarily scheduled on a machine during the last availability that starts before the makespan.

Theorem 1 (Complexity). *Finding a stable schedule with the previous mechanism (flexibility and slack) with minimal $C_{\max}$ is an NP-Hard problem in the strong sense.*

Proof. First, remark that it is easy to verify that the problem of scheduling with the slack rule is in NP.

The proof is based on a reduction from the 3-Partition problem that we recall below (3-PART in [8, SP15]).

3-PART
INPUT: A set A of $3m$ elements, a bound $B \in Z^+$, and a size $s(a) \in Z^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = mB$.
QUESTION: Can A be partitioned into m disjoint sets $A_1, ...A_m$ such that for $1 \leq i \leq m, \sum_{a \in A_i} s(a) = B$?

Let us build an instance of the problem from an instance of 3-PART. $4m$ jobs are scheduled on m processors. $3m$ of these jobs have a processing time equal to the size of the elements of 3-PART. m of these jobs are of size $B + 1$. The targeted makespan of the problem is $3B + 2$.

Each of the m processors must contain exactly one of the m jobs of size $B+1$. Indeed, two such jobs can not be scheduled on the same processor without breaking the slack rule as $2(B+1) + \max(B+1, B+1) = 3B+3 > 3B+2$. The shape of the schedule is depicted on Figure 3.

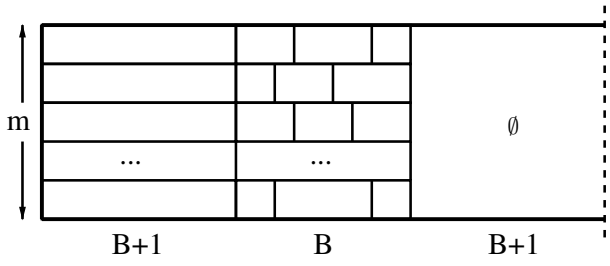

Fig. 3. Scheduling instance respecting the slack rule

Solving the scheduling problem is exactly solving 3-PART. Hence the scheduling problem is NP-Hard in the strong sense. □

5 Bi-objective algorithm

In this section, we describe a bi-objective algorithm and analyze theoretically its stability. We assumed that there exists at least one free processor (with no unavailability constraint), otherwise it is not possible to generate stable schedules.

5.1 Description

Our bi-objective algorithm uses a compromise parameter β for providing schedules resistant to disturbances (see Algorithm 1). Informally, this parameter indicates at which degree the slack rule is respected (this is called the *relaxed slack rule*). The minimal slack of each interval is proportional to β, *i.e.*, $d_i^k \geq \beta \times \max_{j \in \pi(i,k)} p_j$. Moreover, jobs are never scheduled on the last availability periods that start before the makespan. When $\beta = 1$, the produced schedule is stable because of the slack rule (see Proposition 1). When $\beta = 0$, the slack rule is ignored.

Algorithm 1 Greedy Allocation with Parametrized Slack (GAPS)

Input: a set of jobs J
Output: the allocation function π
1: Sort intervals by non-decreasing e_i^k (end dates of unavailabilities)
2: Sort the set of jobs by non-increasing processing times
3: $S = J$ {Set of unscheduled jobs}
4: **for all** interval (i,k) **do** {Consider each interval k on machine i in given order}
5: **for all** $j \in S$ **do** {Consider each job in given order}
6: $M = \sum_{j' \in \pi(i,k)} p_{j'}$ {Processing times of the jobs in current interval}
7: **if** $e_i^k \leq \frac{\sum_{j' \in S} p'_j - p_j}{\lambda}$ **then**
8: **if** $a_i^k - M - p_j \geq \beta \max_{j \in \pi(i,k)} p_j$ **then** {Relaxed slack rule}
9: $\pi(i,k) = \pi(i,k) \cup \{j\}$ {Schedule job j in the current availability}
10: $S = S \setminus \{j\}$ {Update the set S}
11: **end if**
12: **end if**
13: **end for**
14: **end for**
15: Schedule the remaining jobs using LPT on the λ free processors

The first step is to fill greedily the availability periods without violating the relaxed slack rule (Line 8). Note that this step can be seen as a modified version of First Fit Decreasing algorithm for the bin packing problem. In the second step, the λ free processors are treated at once after all the available periods have been filled (Line 15). The transition to the second step occurs when the condition on Line 7 fails. It consists of a lower bound on the time that would be necessary to execute all the unscheduled jobs on the λ free processors. As no job must be scheduled on an availability period that starts just before the makespan, this condition guarantees that the last executed job (such that $C_j = C_{max}$) will be executed on one of the free processors.

The cost of GAPS is low as it only requires jobs and intervals to be sorted. Therefore, its complexity is loglinear in the number of jobs and intervals.

5.2 Theoretical analysis

In order to schedule at least one job in each interval such that the execution of the job is completed, an assumption is done on the size of the jobs relative to the lengths of the availability periods. These lengths should be greater than twice the maximum size of the jobs, *i.e.*, $2 \times p_{\max} \leq a_{\min}$ (with $p_{\max} = \max_j p_j$ and $a_{min} = \min_{i,k} a_i^k$).

We introduce below the *unavailability ratio* γ that prevents an arbitrarily large approximation ratio for the stability. It characterizes the worst percentage of time during which any machine will stay inactive relatively to its previous availability period.

Definition 3. *Let* $u_{\max} = \max_{i,k} u_i^k$. *The unavailability ratio* γ *is*

$$\gamma = \frac{u_{\max}}{a_{\min}}$$

Intuitively, the larger γ, the longer any rescheduled job will wait before its next execution.

Theorem 2 (Stability). *Under the assumption* $2 \times p_{\max} \leq a_{\min}$, *the stability of the GAPS algorithm is*

$$r_S = \begin{cases} \frac{5}{2} - \beta + \gamma & \text{if } \beta \neq 1 \\ 1 & \text{otherwise} \end{cases}$$

Proof. For any schedule built with GAPS, we determine the amount of jobs that are interrupted and that need to be rescheduled after the makespan in the worst case scenario. We focus on one processor but the argument is general and can be extended easily to any number of processors.

Let K be the number of intervals that finish before the makespan on processor i. Hence, the sum of the slacks is $\sum_{k=1}^{K} \beta \times \max_{j \in \pi(i,k)} p_j$. In the worst case, the K-th unavailability finishes at the same time than the makespan (*i.e.*, $e_K^i = C_{\max}$) and the unavailability periods are arbitrarily small (*i.e.*, $\forall k \in [1..K], u_i^k = \epsilon$). Indeed, it maximizes both the number of jobs that are scheduled and their sizes (which maximizes thus the amount of interrupted work). Then, the sum of the processing times of the scheduled jobs is no more than

$$C_{\max} - \sum_{k=1}^{K} \beta \times \max_{j \in \pi(i,k)} p_j$$

(by discarding the ϵ durations).

In the worst case, each unavailability period undergoes disturbances and come earlier (while being still constant). Moreover, it interrupts a job of maximum duration scheduled in its corresponding availability period just before it can finish its execution. Thus, the sum of the jobs that need to be re-executed is $\sum_{k=1}^{K} \max_{j \in \pi(i,k)} p_j$.

As stated in Section 4, the execution is compact, namely, each job is executed as early as possible. Additionally, each unavailability may only interrupt one job. Therefore, we consider that a fraction of the interrupted jobs are re-executed before the makespan using the time reserved by each slack. Hence, the amount of work that need to be re-executed after the makespan is $(1-\beta)\sum_{k=1}^{K}\max_{j\in\pi(i,k)}p_j$. This amount is maximal when the minimum availability period is maximum, which occurs when each period has the same size (*i.e.*, $a_{\min}=\frac{C_{\max}}{K}$). Moreover, the largest job has half this size (*i.e.*, $p_{\max}=\frac{C_{\max}}{2K}$) and each interval has one such job. Therefore, the amount of work to be re-executed becomes $\frac{(1-\beta)}{2}\times C_{\max}$.

We separate this amount into two parts. The first corresponds to jobs of maximum sizes while the second corresponds to the remainder (which can also be a job of maximum size that begins before the makespan and finishes after it). The delays due to each of these part are denoted D_1 and D_2, respectively.

There are $\left\lfloor \frac{(1-\beta)}{2}\times\frac{C_{\max}}{p_{\max}}\right\rfloor = \lfloor(1-\beta)K\rfloor$ jobs of maximum sizes that takes each an entire interval of minimum availability period and maximum unavailability to be re-executed (as it can be interrupted again on this interval). Therefore,

$$D_1 = \lfloor(1-\beta)K\rfloor\,(a_{\min}+u_{\max})$$

The second part of this amount (*i.e.*, $\frac{(1-\beta)}{2}C_{\max} \mod p_{\max}$) either belongs to a job starting its execution before the makespan or is a smaller job. In both cases, it takes a part of an availability period and one complete unavailability period to re-execute it. Therefore,

$$D_2 = (((1-\beta)\times K \mod 1)+\frac{1}{2})a_{\min} + \lceil(1-\beta)K \mod 1\rceil\, u_{\max}$$

Therefore, the worst disturbed maskepan is

$$\begin{aligned}\tilde{C}_{\max} &= C_{\max} + D_1 + D_2\\ &\leq C_{\max} + (1-\beta)Ka_{\min} + Ku_{\max} + \frac{a_{\min}}{2}\\ &\leq C_{\max} + (1-\beta)C_{\max} + \gamma C_{\max} + \frac{C_{\max}}{2}\end{aligned}$$

The stability r_S can directly be derived from this last equation. □

6 Experiments

Simulations are run using data gathered from projects involving BOINC [1]. Traces about availabilities are collected from the project SETI@home [2]. For each processor, the traces provide the starting and ending dates of the availability periods of more than 110,000 processors. These traces were analyzed in [11] and clusters of processors with correlated availabilities were identified.

Workload traces were gathered from project Docking@Home (which was provided to us by Michela Taufer who also modeled the in-progress delay, *i.e.*, the

computation time required by jobs in several desktop grid projects [5]). These traces report the processing times of more than 150,000 jobs.

A preliminary analysis reports that the traces contain jobs with very short effective execution times (from some seconds to few minutes). In practice, they correspond to jobs that were interrupted during their executions. We remove 382 jobs that are shorter that 30 minutes, assuming that this is a reasonable lower duration that a job should have[4].

Each instance consists of a set of machines and a set of jobs. They are both generated randomly from the traces using a uniform distribution law. Moreover, 20% of processors are free (*i.e.*, with no unavailability period). Indeed, more than 20% of the machines were characterized to be available 95% or more of the time [11].

For each simulation, the inputs of the GAPS algorithm consists of an instance and a parameter β. We measure a lower bound of the makespan, the reference makespan and the disturbed makespan. The latter value is obtained by disturbing the actual schedule 30 times and by getting the median disturbed makespan. Disturbances are generated according to our model using a uniform distribution law. The makespan ratio is obtained by dividing the reference makespan by its lower bound, while the stability is the ratio between the disturbed makespan and the reference makespan.

Figure 4 depicts the effect of β on the performances of GAPS. In the boxplots, the bold line is the median, the box shows the quartiles, the bars show the whiskers (1.5 times the interquartile range from the box) and additional points are outliers. For easing the reading, a line links each median. As expected, the stability decreases with high values of β whereas the makespan ratio increases. For low values of β, the makespan can increase by an order of magnitude in presence of disturbances. However, increasing β leads to a far better stability for a reasonable degradation of the makespan ratio (around 20%). Note also that it is not necessary to select a high β in order to obtain a good stability (*e.g.*, for $\beta \geq 0.7$, the stability is close to 1).

7 Concluding remarks

In this work, we have proposed a complete study for scheduling jobs with unavailability periods in an uncertain context. We have introduced a new flexible mechanism based on the concept of adaptive slacks whose sizes are parametrized with the job durations. This leads to a bi-objective algorithm whose principle is to fill the successive intervals by jobs according to non-increasing processing times. The theoretical analysis was assessed by simulations based on actual data for both jobs and availability periods from projects involving BOINC.

Future work is directed towards the implementation of our algorithm for actual large scale applications. Moreover, we plan to extend our model and to

[4] See the statistics reported in http://www.boinc-wiki.info/Catalog_of_BOINC_Powered_Projects.

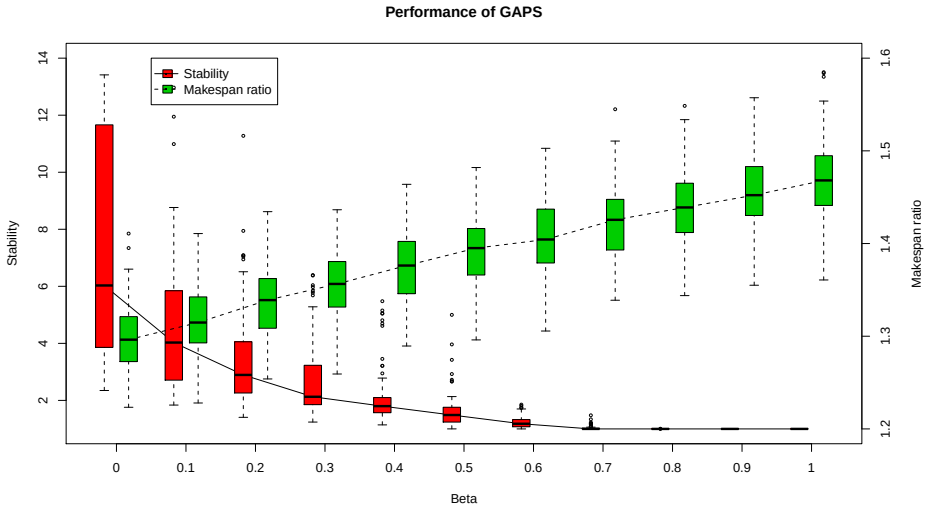

Fig. 4. Effect of the parameter β on the stability and the makespan ratio of GAPS. Each of the 1100 measures represents a simulation with 300 processors and 3000 jobs (GAPS is executed on 100 distinct task and machine instances with 11 values of β for each instance).

adapt our results to uniform machines. Finally, we will derive analogous results for the lateness case in order to develop a more general theoretical framework.

References

1. D. P. Anderson. Boinc: A system for public-resource computing and storage. In *5th International Workshop on Grid Computing (GRID)*, pages 4–10, Nov. 2004.
2. D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11):56–61, 2002.
3. J.-C. Billaut, A. Moukrim, and E. Sanlaville, editors. *Flexibility and Robustness in Scheduling*. Wiley-ISTE, 2008.
4. F. Diedrich, K. Jansen, F. Pascual, and D. Trystram. Approximation Algorithms for Scheduling with Reservations. *Algorithmica*, 58(2):391–404, 2010.
5. T. Estrada, M. Taufer, and K. Reed. Modeling Job Lifespan Delays in Volunteer Computing Projects. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 331–338, Washington, DC, USA, 2009.
6. L. Eyraud, G. Mounié, and D. Trystram. Analysis of Scheduling Algorithms with Reservations. In *21st IEEE International Parallel & Distributed Processing Symposium*, 2007.
7. M. Fallah, M. Aryanezhad, and B. Ashtiani. Preemptive resource constrained project scheduling problem with uncertain resource availabilities: Investigate worth

of proactive strategies. In *Industrial Engineering and Engineering Management (IEEM), 2010 IEEE International Conference on*, pages 646 –650, Dec. 2010.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
9. H.-C. Hwang and S. Y. Chang. Parallel Machines Scheduling with Machine Shutdowns. *Computers & Mathematics with Applications*, 36(11):21–31, Aug. 1998.
10. H.-C. Hwang, K. Lee, and S. Y. Chang. The effect of machine availability on the worst-case performance of LPT. *Discrete Applied Mathematics*, 148(1):49–61, Apr. 2005.
11. D. Kondo, A. Andrzejak, and D. P. Anderson. On correlated availability in internet-distributed systems. In *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 276–283, Tsukuba, Japan, Sept. 2008.
12. O. Lambrechts, E. Demeulemeester, and W. Herroelen. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of Scheduling*, 11(2):121–136, 2008.
13. O. Lambrechts, E. Demeulemeester, and W. Herroelen. Time slack-based techniques for robust project scheduling subject to resource uncertainty. *Annals of Operations Research*, pages 1–22, 2010.
14. S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande. Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. *ArXiv e-prints*, Jan. 2009.
15. C.-Y. Lee. Parallel machines scheduling with nonsimultaneous machine available time. *Discrete Applied Mathematics*, 30(1):53–61, Jan. 1991.
16. C.-Y. Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9(3):363–382, 1996.
17. J. Y.-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall/CCR, 2004.
18. C.-J. Liao, D.-L. Shyur, and C.-H. Lin. Makespan minimization for two parallel machines with an availability constraint. *European Journal of Operational Research*, 160(2):445–456, June 2005.
19. LIGO Scientific Collaboration. The Einstein@Home search for periodic gravitational waves in LIGO S4 data. *ArXiv e-prints*, Apr. 2008.
20. D. Xu, Z. Cheng, Y. Yin, and H. Li. Makespan minimization for two parallel machines scheduling with a periodic availability constraint. *Computers & Operation Research*, 36(6):1809–1812, June 2009.