J. Parallel Distrib. Comput. 74 (2014) 2411–2422

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Fault-tolerant scheduling on parallel systems with non-memoryless failure distributions

Mohamed Slim Bouguerra [a], Derrick Kondo [a], Fernando Mendonca [a,b], Denis Trystram [b,c,*]

[a] INRIA, 655 Avenue de l'Europe, 38334 Saint Ismier cedex, France
[b] Grenoble Institute of Technology, France
[c] Institut Universitaire de France, France

HIGHLIGHTS

- If failure rates are decreasing, we prove that makespan and reliability are antagonistic.
- As there is no single optimal solution, we design an algorithm for computing the approximation set of the Pareto front.
- If failure rates are increasing, both makespan and reliability can be optimized at the same time.
- We prove that the scheduling problem can be solved optimally for several common failure distributions, such as Weibull.

ARTICLE INFO

Article history:
Received 29 July 2012
Received in revised form
20 November 2013
Accepted 17 January 2014
Available online 10 February 2014

Keywords:
Fault tolerance
Reliability
Scheduling
Multi-objective optimization

ABSTRACT

As large parallel systems increase in size and complexity, failures are inevitable and exhibit complex space and time dynamics. Most often, in real systems, failure rates are increasing or decreasing over time. Considering non-memoryless failure distributions, we study a bi-objective scheduling problem of optimizing application makespan and reliability. In particular, we determine whether one can optimize both makespan and reliability simultaneously, or whether one metric must be degraded in order to improve the other. We also devise scheduling algorithms for achieving (approximately) optimal makespan or reliability. When failure rates decrease, we prove that makespan and reliability are opposing metrics. In contrast, when failure rates increase, we prove that one can optimize both makespan and reliability simultaneously. Moreover, we show that the largest processing time (LPT) list scheduling algorithm achieves good performance when processors are of uniform speed. The implications of our findings are the accelerated completion and improved reliability of parallel jobs executed across large distributed systems. Finally, we conduct simulations to investigate the impact of failures on the performance, which is done using an actual application of biological sequence comparison.

1. Introduction

Failures are inevitable in emerging large-scale parallel systems, due to their immense size and complexity. As next-generation parallel systems use an increasing number of components, and the reliability of the individual components will not improve [5], higher failure rates are expected. For instance, new ExaFLOP systems with hundreds of thousands of nodes could have a single failure every 30 min [5]. In real systems, the arrival times of failures have complex space and time dynamics. Recent studies of real large-scale parallel systems show that failure rates are *not* constant, but in fact can increase or decrease over time [33,30,17,21]. Equivalently, the distribution of failure inter-arrival times is *not* memoryless.

Failure rates can decrease (Decreasing Failure Rate *DFR*), as defective hardware or software components are repaired or debugged, for instance. In [33], the authors study failure rates in 22 high-performance computing systems at Los Alamos National Laboratory over a 9-year time period. The best-fitting failure distribution is a Weibull with decreasing failure rate. In [21], the authors study the failure distribution of over 100,000 hosts over the Internet. They also find that failure rates are decreasing over time. Failure rates can also increase (Increasing Failure Rate *IFR*), as system components wear out, for instance. In [32], the authors study the failure rate of a system at IBM with about 400 nodes executing primarily scientific (MPI) applications. They find that the node failure rates increase over time.

* Corresponding author at: Grenoble Institute of Technology, France.
E-mail addresses: slim.bouguerra@imag.fr (M.S. Bouguerra), derrick.kondo@inria.fr (D. Kondo), fernando.machado-mendonca@inria.fr (F. Mendonca), denis.trystram@imag.fr, trystram@imag.fr (D. Trystram).

http://dx.doi.org/10.1016/j.jpdc.2014.01.005
0743-7315/

In contrast, the majority of scheduling studies focusing on reliability assume a memoryless exponential distribution for failure inter-arrival times. The complex properties of general non-memoryless distributions can make analytical solutions intractable.

Considering non-constant failure rates, the goal of this work is to design scheduling strategies that optimize both the completion time and the reliability of parallel applications. These strategies must determine first on which hosts to place the tasks, and then, when to execute the tasks. In particular, we investigate the following questions:

- Are application reliability and makespan opposing metrics? Can one improve reliability without degrading makespan, and vice versa?
- What scheduling algorithms or strategies can achieve optimal reliability or makespan, or approximations thereof?
- How do the answers depend on whether failure rates are increasing or decreasing?

Strategies for tolerating failures incur significant overheads. Checkpointing the state of an application incurs significant IO overheads that can strain the network or memory-to-disk bus. Replication of an application incurs significant overhead as duplicated tasks consume resources across multiple nodes. Thus, minimizing the probability of application failure when it is initially scheduled can help alleviate these overheads.

In general, our contribution is the following. We extend bi-objective scheduling considering reliability and makespan for arbitrary failure distributions, including the uniform and Weibull distributions. This is important in practice because actual systems behave according to such laws, and most existing works are limited to exponential failure distributions.

In particular, whether the reliability and makespan are opposing depends on the form of the law. For the case of *DFR* distributions, we prove that it is impossible to improve one criterion without degrading the other. This relationship between both objectives implies that the problem cannot be approximated by a single schedule. In this configuration, we provide an algorithm to compute an approximation set for the Pareto-optimal solutions. For *IFR* distributions, we show that when processors are identical, the relationship between the objectives changes. They are congruent, leading to optimal policies with respect to both objectives for identical processors. We show that the problem can be efficiently solved by classical list scheduling algorithms for many commonly used failure distributions, such as the Weibull distribution. Simulations and experimentations under different configurations show that LPT is a good list scheduling that optimizes the completion time and reliability as well.

The organization of this paper is as follows. We review in Section 2 the main existing results related to our work. Sections 3 and 4 describe respectively the problem statement and the mathematical tools and theorems. We develop in Sections 5 and 6 the theoretical contributions for the *DFR* case and *IFR* case as well. We present in Section 7 the simulations and experiments conducted in this study. Finally, we conclude this paper in Section 8 with a brief summary and look at future work.

2. Related work

We recall first some classical results in scheduling. Then, we present and discuss fault-tolerance issues related to our problem.

2.1. Classical results in scheduling

In what follows, we will use the classical three-field notation [15] $\alpha|\beta|\gamma$ for denoting a scheduling problem. $P\|C_{\max}$ is the basic scheduling problem where the parallel platform is composed of m identical processors and the application is a set of n independent and arbitrary tasks. The objective is to determine the optimal schedule that minimizes the maximum completion time of the application, i.e., the *makespan* denoted by $C_{\max}$. As shown in [11], $P\|C_{\max}$ is an $\mathcal{NP}$-complete ($\mathcal{NP}$-hard) problem. Hochbaum et al. [18] provided a polynomial time approximation scheme ($\mathcal{PTAS}$) to solve the problem. However, the computational complexity of such a solution is very expensive; hence this result has only a theoretical interest. For the general case of scheduling a precedence task graph, Graham [14] introduced a greedy list algorithm with low linear computational complexity that provides a schedule within a factor of 2 from the optimal schedule. Moreover, he proved that the schedule obtained using the largest processing time (LPT) priority rule is within a factor $\frac{4}{3} - \frac{1}{3m}$ from the optimal.

The second related scheduling problem is $Q|prec|C_{\max}$, which generalizes from the previous case (with m uniform processors) to processors with different speeds. s_i denotes the compute rate of processor i. Liu et al. showed in [24] that the approximation ratio of any greedy list scheduling is in $\mathcal{O}\left(1 + \max_i s_i / \min_i s_i - \max_i s_i / \sum_{i=1}^{m} s_i\right)$. Thus, the performance ratio of list scheduling is not bounded. Jaffe showed that if the ratio between the fastest and the lowest processor is bounded by $\mathcal{O}\left(1/\sqrt{m}\right)$, the approximation ratio can be bounded by $\mathcal{O}\left(\sqrt{m}\right)$ [20]. Finally, Chekuri et al. provided an algorithm to transform the input instance that produces a schedule within a factor $\mathcal{O}(\log m)$ [8].

The third scheduling problem related to our study is less popular: $P\| \sum_{j=i}^{m} g(C_i)$ where C_i is the maximum completion time on processor i and $g : \mathbb{R} \to \mathbb{R}$ is a positive increasing function. Considering the particular function $g(x) = x^2$, Chandra et al. showed that LPT is within a factor 25/24 from the optimal schedule [7]. Then, they extended their analysis to prove that for any function $g(x)$ of the form x^c (such that c is a fixed positive constant) LPT provides a schedule within a bounded approximation ratio that depends on c.

Finally, Alon et al. provided a $\mathcal{PTAS}$ for $P\| \sum_{j=i}^{m} g(C_i)$ [1] for any function g satisfying the two following conditions:

1. $g(x)$ is an increasing and convex function over $\mathbb{R}_+^*$.
2. $\forall \epsilon > 0, \exists \delta > 0$ whose value depends only on ϵ such that $g(x)$ satisfies:

$$\forall x, y \geq 0, \quad (1-\delta)x \leq y \leq (1+\delta)x \Rightarrow (1-\epsilon)g(x) \leq g(y) \leq (1+\epsilon)g(x). \tag{1}$$

The second condition claimed by this theorem stands that for any constant ratio between the image (here the ratio is $1 + \epsilon$) the ratio between the points x and y should be bounded by a constant (the constant is $1 + \delta$) and this constant should be independent from x and y values. For instance the functions $g(x) = x$, $g(x) = x^c$, and $g(x) = \max\{1, x\}$ fulfill both conditions. They also proved that LPT has a bounded approximation ratio for any arbitrary function g that satisfies those conditions.

2.2. Fault-tolerant scheduling

The main approach for fault tolerance is to duplicate tasks in order to increase the reliability of the schedule. In this way, Malewicz considered an application composed of a chain of n sequential tasks executed on m processors [27]. In the failure model of that work, task j has a probability of failure q_{ij} if it is executed on processor i. The author showed that minimizing the expected makespan is $\mathcal{NP}$-complete even if the tasks are independent. Moreover the general problem is inapproximable within a factor lower than 5/4 unless $P = NP$. Considering the same model, Rajaraman et al. designed several approximation algorithms [23]. First, when the tasks are independent, they provided an $\mathcal{O}(\log n)$ approximation. Then, when the application is composed of disjoint chain tasks, they proposed an $\mathcal{O}(\log n \log m \log(n+m)/\log\log(n+m))$ approximation and finally an $\mathcal{O}\left(\log^2 n \log m \log(n+m)/\log\log(n+m)\right)$ approximation when the precedence task graph is an oriented forest.

Jeannot et al. considered another way for tackling the fault-tolerance problem in [22]. They studied the problem $Q\|(C_{\max}, \mathcal{R})$ where the objectives are the minimization of the makespan and at the same time the maximization of the reliability without duplication. Failures occur according to an exponential distribution where each processor has its own failure rate. In that work, the authors considered several application models. First, for a set of independent tasks, they showed that both objectives (makespan and reliability) are antagonistic, i.e., one cannot improve one metric without worsening the other. Thus, it is impossible to determine a schedule that optimizes both the makespan and the reliability simultaneously. To overcome this problem, they proposed two approximation algorithms for computing an approximate set of the solutions on the Pareto-front[1] since the cardinality of this set is exponential (the approximation here is on the polynomial number of solutions). Finally, they proposed a list scheduling algorithm to schedule tasks with precedence constraints. This greedy algorithm prioritizes the processors that maximize the ratio between the failure rate and the execution speeds. As no theoretical guarantee could be established in this case, they assessed this algorithm with extensive experiments.

In the context of embedded system Girault et al. [13] considered a novel multi-objective problem. In the latter work instead of optimizing the classical reliability criteria they target the global failure rate of the application. Here the application is modeled by a DAG of tasks with precedence constraints. This application is executed on a set of m heterogeneous processors. Each processor has it own failure rate and computing rate. Again the failure rate is supposed to be constant. The authors proposed a scheduling algorithm that runs in an $\mathcal{O}\left(nm^2\right)$ where n is the number of tasks in the DAG. This algorithm computes a schedule with an optimal completion time and that verifies a threshold relative to the global failure rate. This work provides a good solution but for small platforms with limited processors number.

Recently Assayad et al. [3] proposed a tricriteria heuristic to tackle a multi-objective problem where the target is to minimize the maximum completion time, the power consumption and maximize the reliability. The proposed heuristic uses the active replication of the operations and the data dependencies between tasks to increase the reliability of the schedule. Also it uses dynamic voltage and frequency scaling to lower the power consumption. This heuristic takes as inputs servers constraint on the three criteria and it produces a set of non-dominated Pareto solutions allowing the user to choose the solution that best meets his/her application needs.

3. Problem statement and notations

In this paper, we analyze the problem of scheduling of parallel applications on a set $\mathbb{Q}$ of m uniform processors. The twofold objective is to minimize the application makespan and maximize the application reliability. Formally, we consider that the parallel application is represented by a set $\mathbb{T}$ of n independent tasks. Each task j is composed of p_j units of work. As introduced in [15], the uniform processors are described by their execution rates per time unit denoted by s_i. The completion time of task j on processor i is given by $p_{ij} = p_j/s_i$.

Typically, a schedule $\mathcal{S}$ is composed of two applications, namely, the space allocation function π and the temporal allocation function σ. π is a mapping from the set of tasks $\mathbb{T}$ to the processors $\mathbb{Q}$ defining for each task where it will be processed. σ is an application from the set of tasks $\mathbb{T}$ to $\mathbb{R}_+$ that determines for each task when it will start its execution. In this paper, we denote by $\mathbb{V}$ the set of valid schedules where no processor executes more than one task at the same time, and each task is executed by at least one processor. Moreover, we assume that a task cannot be preempted by other tasks (this means that as soon as a task starts its execution, it cannot be preempted by another task until the end of its completion).

Regarding the failure model, we consider that all processors have the same failure distribution denoted by F, such that $F(t)$ is the probability a failure occurs in the interval $[0, t]$ on a given processor.

We note that those assumptions concerning the failure distribution and the uniformity of the processors fit perfectly with the context of the volunteer computing. In recent work [21] authors designed a methodology to discover individual hosts' distribution. The authors go on fit probability distributions to the availability durations of these hosts. They find out that they can classify 20% (about 35 686 workers) of BOINC platform [2] workers into 6 clusters of workers with the same failure distribution and uniform speed.

Concerning the application model where tasks are independent and CPU-bound jobs, reflects the tasks found in the context of volunteer computing systems, such as those in the Grid Workload Archive [19] or BOINC [10]. The objective is to determine a schedule $\mathcal{S}$ that minimizes the application makespan $C^{\mathcal{S}}_{\max}$ with a maximal reliability denoted by $\mathcal{R}(\mathcal{S})$. In the remainder of this paper, we denote by $C^{\mathcal{S}}_i = \max_{j\in\mathbb{T}|i=\pi(j)}\{\sigma(j) + p_{ij}\}$ the completion time of the last task on the processor i with respect to schedule $\mathcal{S}$. The first objective is given formally by $C^{\mathcal{S}}_{\max} = \max_{i\in\mathbb{Q}}\{C^{\mathcal{S}}_i\}$.

Any failure occurring during the execution of the application leads to an infeasible schedule. Thus, the reliability of a schedule is given by the probability that no failure occurs on any processor during the entire execution. More formally, consider a schedule $\mathcal{S}$ with the probability that no failure occurs on processor i during the interval $[0, C^{\mathcal{S}}_i]$ $(\overline{F}(C^{\mathcal{S}}_i))$, the reliability $\mathcal{R}$ of $\mathcal{S}$ is given by

$$\mathcal{R}(\mathcal{S}) = \prod_{i=1}^{m} \overline{F}(C^{\mathcal{S}}_i). \tag{2}$$

Alternatively, we define the unreliability $\overline{\mathcal{R}}$ of $\mathcal{S}$ by $\overline{\mathcal{R}}(\mathcal{S}) = 1 - \prod_{i=1}^{m} \overline{F}(C^{\mathcal{S}}_i)$.

Given that the logarithm function is a strictly increasing function, the maximization (or the minimization) of any arbitrary function is equivalent to the maximization (or minimization) of the logarithm of this function. Thus, in the remainder of this work rather than maximizing the classical objective function of the reliability given in Expression (2), we consider the maximization (respectively the minimization) of the logarithm of the reliability denoted by $\mathcal{R}_l$ (respectively $\overline{\mathcal{R}}_l$):

$$\begin{aligned}\max_{\mathcal{S}\in\mathbb{V}} \mathcal{R}(\mathcal{S}) &= \max_{\mathcal{S}\in\mathbb{V}} \prod_{i=1}^{m} \overline{F}(C^{\mathcal{S}}_i) \equiv \max_{\mathcal{S}\in\mathbb{V}} \mathcal{R}_l(\mathcal{S}) \\ &= \max_{\mathcal{S}\in\mathbb{V}} \sum_{i=1}^{m} \log \overline{F}(C^{\mathcal{S}}_i).\end{aligned} \tag{3}$$

As will be shown in the next section, this new objective function has many useful mathematical properties that simplify significantly the problem.

4. Mathematical preliminaries

As a starting point before analyzing the multi-objective problem, we examine each single-objective problem separately. As indicated in Section 2, optimizing the application makespan is a well-known problem that has been extensively studied. On the contrary, there exist only a few results concerning the optimization of reliability. Several questions are still open, such as how to determine the computational complexity of the problem or how

[1] The notion of the Pareto-front will be explained later. Intuitively, it corresponds to the set of the best solutions that compromise among different opposing metrics.

to design an optimal (or at least efficient) scheduling strategy for arbitrary failure distributions.

In what follows, we present several mathematical properties of the reliability objective function that allow us to determine the complexity of the problem and also to design optimal or approximation scheduling algorithms. As shown in Expression (3), the reliability objective function is based on a function that describes failure arrivals. Typically, several functions may be suitable for describing the failure arrivals. Barlow et al. [4] provided a survey about the properties and the usability of the different mathematical functions used for modeling failure arrivals. In this paper, we use the failure distribution function denoted by $F(t)$ and the failure rate function denoted by $\lambda(t)$. These two functions are linked by the following expression:

$$\overline{F}(t) = e^{-\int_0^t \lambda(u)du}. \tag{4}$$

The distributions for which $\lambda(t)$ is increasing in t are denoted by *IFR* (Increasing Failure Rate), or by *DFR* (Decreasing Failure Rate) when $\lambda(t)$ is decreasing [4]. The elementary property used throughout this work is the log-convexity (respectively log-concavity) of *DFR* distributions (respectively *IFR* distributions).

Additionally, we will use several Schur-convexity (Schur-concavity) theorems from the theory of majorization [28].

Majorization is a partial order relation between vectors of real numbers, which applies only to vectors having the same sum. A Schur-convex function preserves the order of majorization between the vectors.

Considering two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. In this work, we use the notation $x_{(1)}$ to indicate the largest element in $\mathbf{x}$ and $x_{(2)}$ to indicate the second-largest element, and so on. Also we use $x_{[k]}$ to denote the kth smallest element in $\mathbf{x}$. By definition, the vector $\mathbf{x}$ is majorized by $\mathbf{y}$ and denoted by $\mathbf{x} \prec \mathbf{y}$ if and only if

$$\sum_{i=1}^{k} x_{(i)} \leq \sum_{i=1}^{k} y_{(i)}, \quad k = 1, 2, \ldots, n \quad \text{and} \quad \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i.$$

Similarly to majorization, weak majorization is a partial order between vectors without the second condition about the equality. We have $\mathbf{x}$ is weakly super-majorized by $\mathbf{y}$ and denoted by $\mathbf{x} \prec^w \mathbf{y}$ if and only if

$$\sum_{i=1}^{k} x_{(i)} \geq \sum_{i=1}^{k} y_{(i)}, \quad k = 1, 2, \ldots, n.$$

Intuitively, if x majorizes y, then y is more mixed than x. This can be used on the scheduling theory as follows. Suppose that $\mathbf{x}$ and $\mathbf{y}$ are the completion time vectors of the different processors. A given scheduling where all the processors have the same completion time like Fig. 1(a) is majorized by all the other schedules' configurations. Equivalently to that a scheduling with an unbalanced or mixed completion time like Fig. 1(b) weakly super-majorizes any other schedule.

The main theorem used in this work is Theorem (3.A.8) in [28]. It states that if f is a Schur-convex and decreasing function then $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $\mathbf{x} \prec^w \mathbf{y}$ ($\mathbf{x}$ is weakly super-majorized by $\mathbf{y}$) we have $f(\mathbf{x}) \leq f(\mathbf{y})$. Under either assumption, it is possible to obtain several dominating strategies that optimize the reliability objective.

Let $\mathbf{c}^{\mathcal{S}} = (C_1^{\mathcal{S}}, \ldots, C_m^{\mathcal{S}})$ denote the vector of processor completion time of schedule $\mathcal{S}$. Recall that $C_{(k)}$ denotes the kth largest completion time and $C_{[k]}$ the kth smallest one. Therefore, $C_{[m]} \leq C_{[m-1]} \leq \cdots \leq C_{[1]}$ similarly $C_{(1)} \leq C_{(2)} \leq \cdots \leq C_{(m)}$.

Theorem 1. *The objective function $\mathcal{R}_l(\mathcal{S}) = \sum_{i=1}^{m} \log \overline{F}(C_i^{\mathcal{S}})$ is Schur-convex (resp. Schur-concave) if F is DFR (resp. IFR).*

Proof. Based on Theorem (4.1) in [4] the function $\log \overline{F}(x)$ is convex (resp. concave) and decreasing (resp. increasing) if F is a *DFR* (resp. *IFR*) distribution.

As a corollary of Theorem (3.A.8) [28], the function $\mathcal{R}_l(\mathcal{S}) = \sum_{i=1}^{m} \log \overline{F}(C_i^{\mathcal{S}})$, is Schur-convex (resp. Schur-concave). □

Theorem 1 reveals a dominating scheduling strategy depending on the nature of the failure rate. When F is a *DFR* distribution, the objective function is Schur-convex. Thus, the optimal scheduling strategy is to allocate all the tasks to the fastest processor. This results in the most skewed distribution of tasks across processors in terms of completion time (as all tasks are assigned to a single processor) as depicted in Fig. 1(b). If F is *IFR*, the optimal scheduling strategy leads to a well-balanced distribution of tasks across processors in terms of completion time since the objective function is Schur-concave as shown in Fig. 1(a). The formal proofs of these two properties are detailed in the next sections.

5. DFR distributions

We consider in this section the case where the failure distribution of processors has a decreasing failure rate. First we focus on the single-objective problem and we show that optimizing the reliability is a polynomial problem when the failure rate is decreasing. Then we study the bi-objective problem and we prove that the makespan and the reliability are antagonistic objectives. Finally, we present an approximation methodology that provides a set of approximated Pareto-optimal solutions.

5.1. Optimizing the reliability

Theorem 2. *The schedule $\mathcal{S}_1$ that allocates all the tasks to the fastest processor is optimal for the problem $Q \| \mathcal{R}_l$ for arbitrary DFR distributions.*

Proof. Consider an application that contains $\mathcal{W}$ units of work $\sum_{j=1}^{n} p_j = \mathcal{W}$.

The completion time vector of $\mathcal{S}_1$ is given by $\mathbf{c}^{\mathcal{S}_1} = (0, \ldots, 0, \frac{\mathcal{W}}{s_1})$.

Let $\mathbf{c}$ be an arbitrary valid completion time vector.

By construction, the partial sum of $C_{(i)}$ and $C_{(i)}^{\mathcal{S}_1}$ verifies

$$\forall k < m, \quad \sum_{i=1}^{k} C_{(i)} \geq \sum_{i=1}^{k} C_{(i)}^{\mathcal{S}_1} = 0,$$

where $C_{(i)}$ is the ith element in $\mathbf{c}$ considering an increasing order.

For $k = m$, let $W_i = C_i s_i$ denote the amount of work allocated to processor i.

Thus, we have: $\sum_{i=1}^{m} C_{(i)} \geq \sum_{i=1}^{m} \frac{W_i}{s_1} = \frac{\mathcal{W}}{s_1}$. This implies that: $\mathbf{c} \prec^w \mathbf{c}^{\mathcal{S}_1}$.

We recall that when F is *DFR*, $\mathcal{R}_l(\mathbf{c})$ is a Schur-convex decreasing function. Thus applying Theorem (3.A.8) [28], we obtain

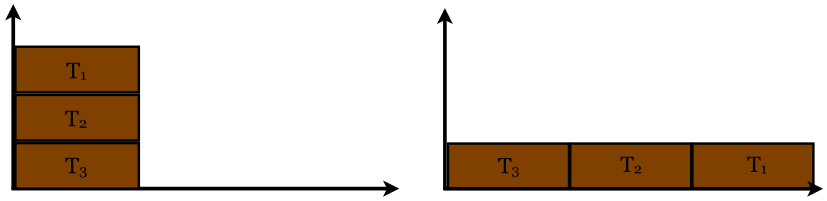
$$\mathbf{c} \prec^w \mathbf{c}^{\mathcal{S}_1} \Rightarrow \mathcal{R}_l(\mathbf{c}) \leq \mathcal{R}_l(\mathbf{c}^{\mathcal{S}_1}). \quad \square$$

According to our knowledge, this is the first result showing that optimizing the reliability without taking into account the makespan is polynomial for any arbitrary *DFR* failure distribution.

5.2. Bi-objective optimization

Consider now the bi-objective problem. Theorem 2 states that optimizing the makespan and the reliability are antagonistic objectives, since the makespan of a schedule with an optimal reliability could be arbitrarily far from the optimal makespan.

To illustrate this point, let us consider the following instance: m identical processors, n identical tasks with $m = n$ and a Weibull distribution of the form $\overline{F}(x) = e^{-x^{\beta}}$ with $\beta < 1$ (which is DFR). For this instance, the best makespan that can be achieved is 1. The corresponding reliability of this schedule is e^{-m}. On the other hand, the optimal schedule for the reliability has a makespan of m and a



(a) Balanced completion time configuration. (b) Unbalanced completion time configuration.

Fig. 1. Gantt chart of scheduling configurations.

reliability e^{-m^β}. Therefore it is clear that the performance ratio is m for the makespan and $e^{m^\beta - m}$ for the reliability.

This implies that optimizing one objective will lead to the deterioration of the other by an unbounded factor. There is no concept of an absolute optimal solution here. Moreover, since we aim at optimizing simultaneously more than one objective, we cannot establish an absolute order between the solutions. This brings us to another question: is it possible to design schedules with a guaranty on both objectives? One possible answer is to use multi-objective optimization techniques that are recalled below.

Since there is no absolute order between the different solutions, a partial order called the Pareto-dominance is used to distinguish the best solutions. More precisely, if a solution is better than another one in both objectives, it is said that this solution Pareto-dominates the latter. In the remainder of this paper, we call the Pareto-optimal set (also called the Pareto-front) the set of all Pareto-dominating solutions. It is denoted by $\mathbb{P}^*$. However, most of the time it is inconceivable to compute all the Pareto-dominating solutions mainly for two reasons. Typically the single-objective optimization problem is often $\mathcal{NP}$-complete. (It is for instance the case for minimizing the makespan.) Furthermore, the cardinality of such a set is often exponential. As it will be shown later in our case study, both reasons stand. Given those difficulties, the alternative is to design approximation algorithms that compute in polynomial time an approximation set denoted by $\mathbb{P}^*_a$. This set approximates all the valid solutions in $\mathbb{P}^*$ and verifies that for any Pareto-optimal schedule $\mathcal{S}^* \in \mathbb{P}^*$ it corresponds necessarily to an approximation solution $\mathcal{S} \in \mathbb{P}^*_a$ within a constant factor from $\mathcal{S}^*$.

Definition 1. Let ρ_1 and ρ_2 be two real numbers such that $1 \leq \rho_1,\ 1 \leq \rho_2$. The approximation Pareto-optimal set $\mathbb{P}^*_a$ is called a $\langle \rho_1, \rho_2 \rangle$-approximation if and only if for any valid Pareto-optimal schedule $\mathcal{S}^* \in \mathbb{P}^*$ it corresponds to an approximation solution $\mathcal{S} \in \mathbb{P}^*_a$ that verifies the following approximation bounds $C^{\mathcal{S}}_{\max} \leq \rho_1 C^{\mathcal{S}^*}_{\max}$ and $\mathcal{R}_l(\mathcal{S}^*) \leq \rho_2 \mathcal{R}_l(\mathcal{S})$.

In other words, this notation represents the approximation ratios related to each objective separately. An efficient method to approximate the Pareto-optimal set was proposed by Papadimitriou and Yannakakis [31]. This method is composed of two following steps.

In the first step we have to construct a ρ_2-approximation of the second objective (in our case the reliability) constrained by a fixed threshold τ on the first objective (the makespan in our case). Before presenting the second step, we point out the following definition.

Definition 2. Given a threshold τ, we call a $\langle \overline{\rho}_1, \rho_2 \rangle$-APPROX an algorithm that computes a solution $\mathcal{S}$ satisfying the following conditions:

1. The maximum completion time verifies $\rho_1 C^{\mathcal{S}}_{\max} \leq \tau$.
2. For any Pareto-optimal solution $\mathcal{S}^* \in \mathbb{P}^*$ with a maximum completion time verifying $C^{\mathcal{S}^*}_{\max} \leq \tau$ the algorithm returns a solution $\mathcal{S}$ verifying $\mathcal{R}_l(\mathcal{S}^*) \leq \rho_2 \mathcal{R}_l(\mathcal{S})$.

This means in our case that the $\langle \overline{\rho}_1, \rho_2 \rangle$-APPROX provides a solution within a factor ρ_2 from the reliability and a factor ρ_1 from the makespan of any Pareto-optimal solution with a makespan less than the threshold τ.

Then, in the second step, we use this $\langle \overline{\rho}_1, \rho_2 \rangle$-APPROX with successive values of thresholds $\tau_1 \cdots \tau_k$ between lower and upper bounds (L_b and U_b, respectively) as indicated in Algorithm 1. Then we obtain a $\langle \rho_1 + \epsilon, \rho_2 \rangle$-approximation of the Pareto-optimal set. As a result, the cardinality of the approximated Pareto-optimal set depends essentially on the number of iterations k which is bounded by $\mathcal{O}\left(\log_{1+\epsilon/\rho_1}(U_b/L_b)\right)$. Let us notice that the quantity $\log_{1+\epsilon/\rho_1}(U_b/L_b)$ which represents the cardinality of $\mathbb{P}^*_a$ is polynomial, since in our context U_b/L_b is the ratio between the optimal and the worst makespan. Thus, the computational complexity of Algorithm 1 is polynomial if the complexity of $\langle \overline{\rho}_1, \rho_2 \rangle$-APPROX is polynomial as well.

Algorithm 1 Approximation Algorithm of the Pareto-optimal set [31]

```
function <ρ1 + ε, ρ2>-APPROX(ε, ρ1, ρ2, Ub, Lb)
    k ← 0
    P*a ← ∅
    while k ≤ ⌈log_{1+ε/ρ1}(Ub/Lb)⌉ do
        τk ← (1 + ε/ρ1)^k Lb
        Sk ←< ρ̄1, ρ2 >-APPROX(τk)
        P*a ← P*a ∪ Sk
        k ← k + 1
    end while
    return P*a
end function
```

5.3. Case of unitary tasks

We consider here the specific case where the application is composed of independent and unitary tasks. The input instance is given by the number of tasks n, the processor speed vector $(s_1, \ldots, s_m)$, and the failure distribution F. The first question addressed here concerns the cardinality of the Pareto-optimal solutions.

Theorem 3. *The cardinality of the Pareto-optimal set is exponential in the size the instance for the problem* $2|p_i = 1|(C_{\max}, \mathcal{R}_l)$ *when* F *is DFR.*

Proof. Let us consider the following instance: $2n$ unitary tasks, 2 identical processors and a failure distribution with a strictly decreasing failure rate (for instance, a Weibull distribution with a shape parameter strictly less than 1). Notice that in this case the function $g(x) = \log \overline{F}(x)$ is a strictly convex function.

Since the tasks are unitary, scheduling strategies are simply characterized by the number of tasks assigned to the first or to the second processor.

Let $\mathcal{S}_j$ denote the scheduling strategy that schedules $2n - j + 1$ tasks on processor 1 and $j - 1$ on processor 2.

For each j and δ verifying $1 \leq j \leq n+1$ and $1 \leq \delta \leq n-j+1$, it is clear that the makespan of $\mathcal{S}_{j+\delta}$ is strictly better that the makespan of $\mathcal{S}_j$:

$$C^{\mathcal{S}_{j+\delta}}_{\max} < C^{\mathcal{S}_j}_{\max}. \tag{5}$$

Based on the strict convexity inequality, the reliability of $\mathcal{S}_{j+\delta}$ is strictly less than the reliability of $\mathcal{S}_j$:

$$\mathcal{R}_l(\mathcal{S}_j) = g(2n-j+1)+g(i-1) > g(2n-j-\delta+1) + g(j-1+\delta) = \mathcal{R}_l(\mathcal{S}_{j+\delta}). \quad (6)$$

Inequalities (5) and (6) imply that each $\mathcal{S}_j$ is a Pareto-optimal solution.

Thus we have an $\mathcal{O}(n)$ Pareto-optimal solution while the encoding of the input instance is in $\mathcal{O}(\log n)$.

Therefore, the cardinality of the Pareto-optimal front is exponential in the function of this encoding. □

Despite this somehow negative result, it is possible to enumerate all the Pareto-optimal solutions for reasonable values of n. But such an operation could become costly for a large value of n, for instance when the number of tasks is exponential with respect to the number of processors $n = \mathcal{O}(2^m)$. In what follows, we propose an algorithm to enumerate all the Pareto-optimal solutions or to compute an approximation set of the Pareto-optimal set for very large values of n.

The proposed algorithm is based on the approximation methodology proposed by Papadimitriou and Yannakakis [31]. First we design the $\langle\overline{\rho_1}, \rho_2\rangle$-APPROX algorithm. Then using this algorithm with different values of threshold a $\langle\rho_1+\epsilon, \rho_2\rangle$-approximation set is provided.

For the unitary job case we provide a $\langle\overline{1}, 1\rangle$-APPROX. Recall that when the distribution is *DFR* the objective function is Schur-convex. Then the main idea is to schedule tasks in such a way that the completion time vector is mixed enough to weakly super-majorize any other scheduling. To obtain such a schedule the idea is to allocate as many jobs as possible to the fastest processors. Of course we recall that the maximum completion time should be less than the threshold τ. We propose Algorithm 2 to achieve this using a greedy scheduling approach. Let w_i denote the amount of work allocated to processor i. The first step is to sort the processors in decreasing order according to the speed s_i. Then, each processor will receive Min $(n-A, \lfloor\tau s_i\rfloor)$. We note that A denotes the number of the already scheduled tasks. Algorithm 2 is a $\langle\overline{1}, 1\rangle$-APPROX that returns a schedule within a factor 1 from the threshold τ and with a reliability better than any other schedule with a maximum completion time less than τ.

Algorithm 2 Scheduling unitary tasks for *DFR* distributions

```
function <1̄, 1>-APPROX(τ, n, ℚ)
    A ← 0                                   ▷ Number of scheduled tasks
    for i ← 1 to m do                       ▷ Processors are sorted according to
decreasing order of their s_i
        w_i ← 0
        if A < n then
            w_i ← Min (n − A, ⌊τ s_i⌋)      ▷ Amount of work allocated
to i
        end if
        A ← A + w_i
    end for
    if A==n then
        return w                            ▷ Vector of workload allocated to each
processor
    end if
    return Error                            ▷ No feasible solution for the constraint τ
end function
```

The correctness proof of this algorithm is given in two separated theorems. First in Theorem 4 we propose a workload transfer rule and we show that this rule improves the reliability of the input schedule. Then in Theorem 5 we show that the output schedule of Algorithm 2 could be obtained by successive iterations of the workload transfer rule.

Workload transfer rule. Consider two processors i and i' such that processor i is faster than processor i'. Let us consider any arbitrary schedule $\mathcal{S}$ in which we transfer δ units of workload from the slower to the fastest processor. This transfer should verify that the new makespan of processor i is greater or equal to the new makespan of processor i'. Transferring workload according to this rule leads to a new completion time scheduling vector that weakly super-majorizes the initial schedule vector. Thus this leads to a new schedule with a better reliability when F is *DFR* since the objective function is Schur-convex.

Theorem 4. *Consider a completion time vector* $\mathbf{c}$ *of an arbitrary schedule and two arbitrary processors* i *and* i' *satisfying* $s_{i'} < s_i$. *Let* δ *represent the amount of workload to transfer from processor* i' *to processor* i. *This will produce the completion time vector* $\hat{\mathbf{c}} = (\hat{C}_1, \ldots, C_i+\delta/s_i, \ldots, C_{i'}-\delta/s_{i'}, \ldots, C_m)$ *such that* $C_{i'}-\delta/s_{i'} \le C_i+\delta/s_i$. *Then* $\mathbf{c}$ *and* $\hat{\mathbf{c}}$ *satisfy* $\mathbf{c} \prec^w \hat{\mathbf{c}}$.

The proof of this theorem is based on a case by case analysis. Details appear in the Appendix at the end of the paper. This theorem states that the proposed workload transfer operation leads to a new completion time vector that weakly super-majorizes the initial vector. Since the reliability objective function is Schur-convex when F is *DFR*, the reliability of the new schedule is better than the initial one. Based on this result, we are able to extend the scheduling algorithm proposed by Jeannot et al. [22] to more general cases, taking into account any failure distributions with a decreasing failure rate.

In the following theorem we show that Algorithm 2 verifies Definition 2 where $\rho_1 = \rho_2 = 1$.

Theorem 5. *If there exists a valid schedule with a makespan* $C_{\max} \le \tau$, *then Algorithm* 2 *produces a valid schedule* $\hat{\mathcal{S}}$ *such that* $\hat{C}_{\max} \le \tau$ *and* $\forall \mathcal{S} \in \mathbb{V} \mid C^{\mathcal{S}}_{\max} \le \tau, \mathcal{R}_l(\mathcal{S}) \le \mathcal{R}_l(\hat{\mathcal{S}})$.

Proof. First, note that if the algorithm does not return a solution there is no valid schedule satisfying $C_{\max} \le \tau$.

Second, if the algorithm returns a solution $\hat{\mathcal{S}}$, this solution verifies the following:

1. By construction the completion time of this algorithm verifies $\hat{C}_{\max} \le \tau$. Recall that each processor receives at most $\lfloor\tau s_i\rfloor$.
2. The reliability of $\hat{\mathcal{S}}$ is greater or equal to the reliability of any schedule $\mathcal{S}$ that satisfies $C_{\max} \le \tau$.

To prove item 2 we show that it is possible to transform any arbitrary schedule to $\hat{\mathcal{S}}$ using the workload transfer rule according to Theorem 4.

Consider an arbitrary schedule $\mathcal{S} \in \mathbb{V}$ that verifies $C_{\max} \le \tau$.

By construction the makespan of processor i in $\mathcal{S}$ and $\hat{\mathcal{S}}$ verifies the following relation $C_i \le \hat{C}_i$, since each processor in schedule $\hat{\mathcal{S}}$ is loaded to the maximum if there is enough workload.

In this case processor i received less work in schedule $\mathcal{S}$ than in schedule $\hat{\mathcal{S}}$.

This amount of work is assigned to a processor i' which is slower than i since by construction all the processors faster than i are fully loaded.

Based on Theorem 4, transferring some load from processor i' to processor i such that $\hat{C}_i = C_i + \delta/s_i = \lfloor\tau s_i\rfloor$ will necessarily increase the reliability.

Finally, let us notice that this algorithm is deterministic. Therefore after a set of transfers of workload starting from the fastest processor, it produces necessarily the schedule $\hat{\mathcal{S}}$. □

Using this $\langle\overline{1}, 1\rangle$-APPROX with Algorithm 1, we are able to provide a $\langle 1+\epsilon, 1\rangle$-approximation of the optimal Pareto-optimal set. This is obtained by increasing the constraint τ from $L_b = C^*_{\max}$ to $U_b = n/s_1$. $C^*_{\max}$ is computed in polynomial time since the tasks are unitary [6]. The computational complexity of this algorithm is $\mathcal{O}\left(m\log m + m\log_{1+\epsilon}(n/v_1)\right)$ which is linear with respect to the encoding of the input instance. It is worth noting that when $\epsilon = 0$ this algorithm enumerates all of the Pareto-optimal solutions.

5.4. Arbitrary tasks

We analyze in this section the general case where the tasks are of arbitrary size. The input instance of the problem are the tasks' set $\mathbb{T}$ and their sizes in terms of amount of work $(p_1, p_2, \ldots, p_n)$, the processors' set $\mathbb{Q}$ and their speeds $(s_1, s_2, \ldots, s_m)$ and the failure distribution function F.

Theorem 6. *The cardinality of the Pareto-optimal set of the problem $Q\|(C_{\max}, \mathcal{R}_l)$ is exponential in the size of the input.*

Proof. Let us consider the following instance:

- n tasks defined by $p_j = 2^{j-1}$ for $1 \leq j \leq n$.
- 2 identical processors.
- Failure distribution with a strictly decreasing failure rate (Weibull distribution with a shape parameter equal to $\frac{1}{2}$).

We note that, for this instance there are 2^{n-1} schedules with a different makespan ranging in $2^{n-1} \leq C_{\max} \leq 2^n$.

Let $\mathcal{S}_j$ be the family of schedules that allocate $2^n - j$ units of work on processor 1 and j units of work on processor 2 for $0 \leq j \leq 2^{n-1}$.

Using the same exchange argument as in Theorem 3, we verify easily that each $\mathcal{S}_j$ is a Pareto-optimal solution.

Thus, the cardinality of the Pareto-optimal set is in $\mathcal{O}(2^n)$. □

Unlike the proof presented by Jeannot et al. [22] for the particular case of exponential distribution, this theorem indicates that the cardinality of the Pareto-optimal set is exponential even for 2 identical processors. Therefore, enumerating all the Pareto-optimal solutions is too costly for reasonable instances. As for the case of unitary tasks, we design an approximation algorithm to approximate the Pareto-optimal set using the methodology as before. The proposed algorithm is based on the classical greedy list scheduling strategy. The principle is to schedule first as many of the largest tasks on the fastest processors as possible such that the maximum completion time meets a given threshold. Note that there is no polynomial algorithm for scheduling optimally arbitrary tasks unless $P = NP$ [12]; thus unlike the first case the target threshold considered is 2τ. Contrary to the case where tasks are unitary we provide a $\langle 2 + \epsilon, 1\rangle$-approximation of the Pareto-optimal set. The proposed approximation algorithm is based on the classical list scheduling that achieves a 2-approximation.

Algorithm 3 Scheduling arbitrary tasks for *DFR* distribution

```
function <2̄, 1>-APPROX(τ, 𝕋, ℚ)
    j ← 1
    i ← 1
    Sort p_j and s_i in decreasing order
    ∀i ∈ ℚ, C_i ← 0
    while j ≤ n do
        if C_i ≤ τ then
            π[j] ← i
            σ[j] ← C_i
            C_i ← C_i + p_j / s_i
            j ← j + 1
        else i ← i + 1
        end if
        if i > m then
            return Error ▷ No feasible solution for the threshold τ
        end if
    end while
    return π, σ
end function
```

As indicated in Algorithm 3 the principle is to schedule first the largest jobs on the fastest processors such that the makespan of the schedule is less than 2τ. Algorithm 3 is a $\langle\bar{2}, 1\rangle$-APPROX that produces a schedule with a makespan less than 2τ and a reliability better than any other schedule with a makespan less than the threshold τ.

Theorem 7. *If there exists a valid schedule verifying $C_{\max} \leq \tau$, then Algorithm 3 produces a valid schedule $\hat{\mathcal{S}}$ verifying $\hat{C}_{\max} \leq 2\tau$ and $\forall \mathcal{S} \in \mathbb{V} \mid C^{\mathcal{S}}_{\max} \leq \tau,\ \mathcal{R}_l(\mathcal{S}) \leq \mathcal{R}_l(\hat{\mathcal{S}})$.*

Proof. Firstly we show by contradiction that Algorithm 3 returns at least one valid solution or there is no valid schedule with a makespan less than τ.

Suppose by contradiction that there exists a valid schedule with a $C_{\max} \leq \tau$ and Algorithm 3 does not return a valid solution.

This implies that there exists a task j' scheduled on processor i' at a time less or equal to τ such that $\hat{C}_{i'} > 2\tau$.

Therefore, the amount of work of this task verifies $p_{j'}/s_{i'} > \tau$.

We recall that the proposed algorithm schedules first the biggest tasks on the fastest processor.

Therefore, in a valid schedule whose makespan verifies $C_{\max} \leq \tau$, the job j' could not be scheduled on any processor slower than processor i' since the amount of work of job j' satisfies $p_{j'}/s_{i'} > \tau$.

We recall also that all the processors fastest than processor i' have a makespan greater than τ.

Hence in any valid schedule verifying $C_{\max} \leq \tau$, j' should be scheduled on i'.

Thus, there is no valid schedule that verifies $C_{\max} \leq \tau$.

Secondly, we show that the reliability of the provided schedule $\hat{\mathcal{S}}$ is better than the reliability of any valid schedule $\mathcal{S}$ with a $C_{\max} \leq \tau$.

In what follows we show that it is possible to transform $\mathcal{S}$ to $\hat{\mathcal{S}}$ according to the workload transfer rule as in Theorem 4.

Recall that by construction, Algorithm 3 returns a schedule $\hat{\mathcal{S}}$ such that there exists an index $k \leq m$ where

- The makespan of the k fastest processors satisfies $1 \leq i \leq k$, $\hat{C}_{[i]} > \tau$.
- The makespan of processor $k+1$ satisfies $\hat{C}_{[k+1]} \leq \tau$.
- The makespan of all the remaining processors satisfies $k+1 < i \leq m,\ \hat{C}_{[i]} = 0$.

The transformation of $\mathcal{S}$ to $\hat{\mathcal{S}}$ is done as follows:

- The k fastest processors of $\hat{\mathcal{S}}$ are filled by transferring works from the $m-k$ slowest processors in $\mathcal{S}$.
- For the $(k+1)$th fastest processor there are two cases to consider.
  1. The makespan satisfies $C_{k+1} < \hat{C}_{k+1}$; thus we transfer all the remaining work of the $m-k$ slowest processors in $\mathcal{S}$ to processor $k+1$.
  2. The makespan satisfies $C_{k+1} > \hat{C}_{k+1}$; the extra amount of work of processor $k+1$ has been allocated to a faster processor since by construction $\hat{C}_i = 0$ for all $i > k$.

Based on Theorem 4 all those operations increase necessarily the reliability. □

Using this algorithm with the methodology proposed by Papadimitriou and Yannakakis [31], we provide a $\langle 2+\epsilon, 1\rangle$-APPROX algorithm for solutions in the Pareto-optimal set. In this case, the lower bound is given by $\max\{\sum_{j=1}^{n} p_j / \sum_{i=1}^{m} s_i, p_{\max}/s_1\}$ and the upper bound remains the same $\sum_{j=1}^{n} p_j/s_1$.

Hence, the cardinality of the approximation set is bounded by $\mathcal{O}\left(\log_{1+\epsilon}(\sum_{j=1}^{n} p_j/s_1)\right)$, and the computational complexity of the algorithm that computes the $\langle 2+\epsilon, 1\rangle$-approximation is bounded by $\mathcal{O}\left(m \log m + n(\log n + \log_{1+\epsilon}(\sum_{j=1}^{n} p_j/s_1))\right)$.

6. IFR distributions

We consider in this section the second configuration where the distribution F is *IFR*. This class contains many important probability distributions including the Weibull law with a shape parameter

greater than 1. The first question addressed in this section concerns the complexity of each single-optimization problem.

6.1. Complexity analysis

Theorem 8. *The problem $P\|\overline{\mathcal{R}}_l$ is $\mathcal{NP}$-complete for some IFR distributions.*

Proof. We recall that when the failure rate is increasing $g(x) = -\log \overline{F}(x)$ is a convex increasing function.

In this case the same problem is similar to the problem introduced in [1] if we consider for instance a Weibull distribution with a shape parameter equal to 2 ($g(x) = x^2$). □

Theorem 8 points out that unlike the case for *DFR* distributions, for *IFR* distributions, optimizing the reliability is an $\mathcal{NP}$-complete problem even when the processors are homogeneous in terms of computing speeds. However, this is not discouraging, because the objective function $\sum_{j=1}^{m} -\log \overline{F}(C_i)$ is Schur-convex. In fact, based on Theorem (3.A.8) [28] the objective function is optimized when all C_i are well-balanced if processors are identical. This implies that the makespan and the reliability are congruent objectives, i.e., one can improve the makespan while at the same improve reliability.

6.2. Unitary tasks

We consider in this subsection the case where the tasks are unitary. We recall that according to the finding in Theorem 8, the optimal schedule is the most well balanced one. Thus intuitively one can have such a schedule using the classical list scheduling. Consider that the application is composed of n tasks such that $n = \alpha m + r$. Using the list scheduling we will have r processors loaded with $\alpha + 1$ tasks and $m - r$ processors loaded with α tasks.

In the next theorem we show that this algorithm produces a schedule $\hat{\mathcal{S}}$ with the best balanced completion time vector denoted by $\hat{\mathbf{c}}$ that maximizes the reliability. Recall that $\overline{\mathcal{R}}_l = 1 - \mathcal{R}$ represents the unreliability of the schedule.

Theorem 9. *The problem $P|p_j = 1|(\overline{\mathcal{R}}_l, C_{\max})$ is polynomial.*

Proof. We show that the vector $\hat{\mathbf{c}}$ satisfies: $\hat{\mathbf{c}} \prec \mathbf{c}$, where $\mathbf{c}$ is any valid arbitrary completion time vector of a valid schedule $\mathcal{S} \in \mathbb{V}$.

Suppose by contradiction that the relation $\hat{\mathbf{c}} \prec \mathbf{c}$ does not hold.

- First case:

$$\exists k \leq r \mid \sum_{i=1}^{k} C_{[i]} < \sum_{i=1}^{k} \hat{C}_{[i]}. \tag{7}$$

Then $\sum_{i=1}^{k} C_{[i]} < k(\alpha + 1) \Rightarrow C_{[k]} \leq \alpha$ (since there exists a processor loaded more than $\alpha + 1$).
Therefore, $\forall i \mid k < i \leq m,\ C_{[k]} \leq \alpha \Rightarrow C_{[i]} \leq \alpha$.
In this case the sum of workload on the remaining processors is given by

$$\sum_{i=k+1}^{m} C_{[i]} \leq (m-k)\alpha. \tag{8}$$

Thus inequalities (7) and (8) imply

$$\sum_{i=1}^{m} C_{(i)} < r - k + \alpha m.$$

This leads to the desired contradiction since $r - k - 1$ tasks are none scheduled in $\mathcal{S}$.

- Second case:

$$\exists k, |r < k \leq m, \quad \sum_{i=1}^{k} C_{[i]} < \sum_{i=1}^{k} \hat{C}_{[i]}. \tag{9}$$

Therefore using the same argument $C_{[k]} < \alpha$ (since there exists at least one processor loaded more than α otherwise we have the equality).

Thus the sum of the workload on the remaining processors is given by

$$\sum_{i=k+1}^{m} C_{[i]} \leq (m-k)(\alpha - 1). \tag{10}$$

This leads to a contradiction since inequalities (9) and (10) imply that $m - k - 1$ are not scheduled in $\mathcal{S}$. □

A direct corollary of this theorem is that the produced solution is a global optimum for both objectives.

6.3. Arbitrary tasks

We consider in this subsection the general problem where the length of the tasks are arbitrary. We recall that in this case both single-objective problems are $\mathcal{NP}$-complete. To solve the multi-objective problem, two methodologies are possible.

The first methodology is to design a polynomial time approximation scheme ($\mathcal{PTAS}$) to solve optimally the problem. Therefore one can use the $\mathcal{PTAS}$ designed by Alon et al. [1] to solve the problems $P\|\sum_{i=1}^{m} g(C_i)$ and $P\|C_{\max}$ at the same time. However, this approximation scheme is usable if and only if the function $g(x)$ satisfies the following conditions:

1. $g(x)$ is an increasing and convex function over $\mathbb{R}_+^*$.
2. $\forall \epsilon > 0, \exists \delta > 0$ whose value depends only on ϵ such that $g(x)$ verify

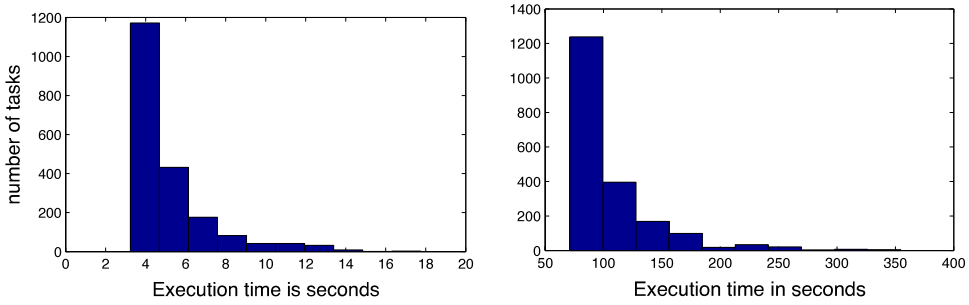
$$\forall x, y \geq 0, \quad (1-\delta)x \leq y \leq (1+\delta)x \Rightarrow (1-\epsilon)g(x) \leq g(y) \leq (1+\epsilon)g(x).$$

We note that in our case, any *IFR* distribution satisfies the first condition. But we do not have any formal proof that any *IFR* distribution satisfies the second condition. However, many popular failure distributions such as the Weibull distribution satisfy this condition since $g(x)$ is of the form αx^{β} with $\alpha \in \mathbb{R}_+^*$ and $\beta \geq 1$. We notice that the computational complexity is too costly. The alternative methodology is to use heuristics with a low computational complexity cost and a constant approximation ratio to solve the problem (for example, the LPT (largest processing time) list scheduling algorithm). The principle is to allocate iteratively the longest remaining task of the list to an idle processor. It provides a 2-approximation of the makespan [15]. However, for the second objective, this heuristic can be achieve a constant factor only for particular a failure distribution such as Weibull. For instance, based on the analysis given in [7], for a Weibull distribution with a shape parameter equal to 2, the LPT scheduling algorithm produces a schedule where the reliability is within a factor 25/24 from the optimal.

7. Simulations and evaluations

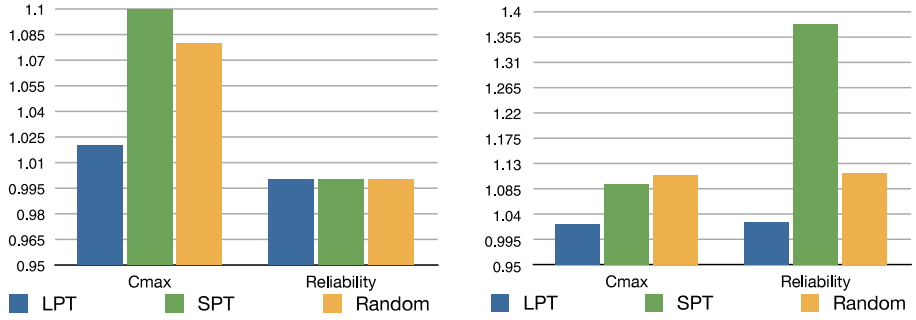
We investigate in this section the impact of failures on the performance. The first set of simulations is to study the performance ratio of various list scheduling strategies when the failure rate is increasing (IFR). In the second set of simulations we consider the class of *DFR* distributions. We consider the case where the application is executed over a set of identical processors in terms of execution and failure rate. We also consider an actual application of comparison of protein sequence databases as the source of tasks for the simulations. Two commonly used protein databases [34,25,16,26] were used in the simulations: RefSeq Mouse and UniProtKB. They are briefly described below.

RefSeq The Reference Sequence (RefSeq) collection [29] provides a comprehensive and non-redundant set of sequences, including genomic DNA and proteins. RefSeq sequences are used for medical, functional, and diversity studies. They



(a) RefSeq Mouse database. (b) UniProtKB database.

Fig. 2. Histogram of tasks' execution time in seconds.



(a) RefSeq Mouse protein with 1 failure/day per processor. (b) UniProtKB with 6 failures/day per processor.

Fig. 3. Execution with 80 processors. Each bar represents the $C_{\max}$ and reliability ratio to the optimal.

provide a stable reference for genome annotation, gene identification and characterization, mutation and polymorphism analysis, expression studies, and comparative analyses. The May 2011 RefSeq collection (Release 47) includes sequences from more than 12,000 distinct taxonomic identifiers, ranging from viruses to bacteria to eukaryotes.

We used the RefSeq Mouse database. The current version contains 29,437 sequences ranging from a few residues to a few billions in size.

UniProt The Universal Protein Resource (UniProt) [9] is a comprehensive resource for protein sequence and annotation data. The UniProt databases are the UniProt Knowledgebase (UniProtKB), the UniProt Reference Clusters (UniRef) and the UniProt Archive (UniParc). It is also the central hub for the collection of functional information on proteins, with accurate, consistent and rich annotation.

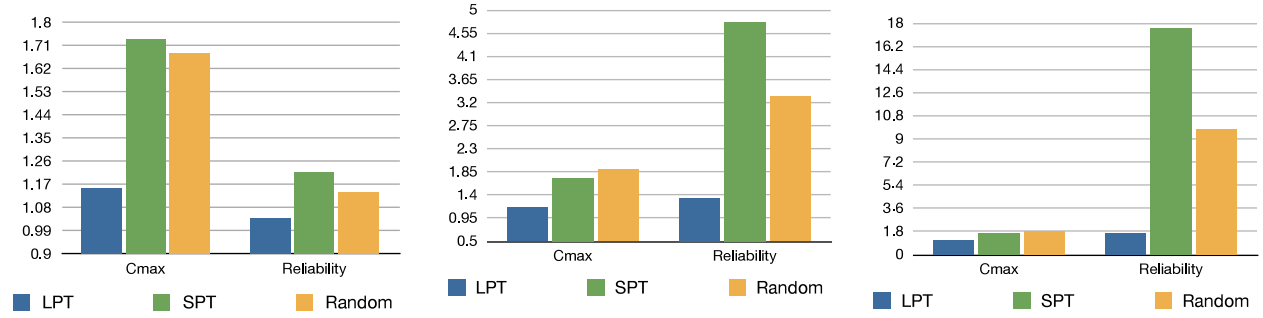
We used the UniProtKB database. The most recent version contains 537,505 sequences ranging in size from a few hundred thousands to 6 trillion residues.

In order to obtain execution times for the tasks chosen for the simulations, we considered the computing capabilities of the Idgraf machine located in Inria in Grenoble. It contains two Intel Xeon X5650 processors (Westmere, 6 cores each, total 12 cores) and 64 GB of RAM. As shown in Fig. 2 both databases have similar length distributions but with different overall task sizes. Tasks taken from the RefSeq Mouse database have execution times as high as 18 s while tasks taken from the UniProtKB are executed in up to 350 s.

7.1. IFR distributions

In this simulation we use the popular Weibull failure distribution [17,21,30]. We consider different scale and shape parameters of Weibull varying from 1 to 6 failures per day per processor and a shape parameter ranging between 1.1 and 2. We investigate the performance ratio of various list scheduling strategies. To this end, we consider a set of candidate scheduling algorithms, namely LPT (largest processing time), SPT (short processing time) and Random. We recall that in LPT (respectively SPT) the largest tasks (respectively smallest tasks) are assigned first. In Random, tasks are assigned randomly as soon as a processor is available. To study and compare these scheduling algorithms we compute the performance ratio of each strategy (ratio of the actual makespan over the optimal). The lower bound for the completion time L_b is given by $L_b = \max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j\}$. Recall based on Theorem (3.A.8) [28] when the distribution is *IFR* the objective function is optimized when all C_i are well-balanced and identical processors. To compute the reliability lower bound we apply the Formula (2) using the optimal vector $\hat{\mathbf{c}} = (\sum_{j=1}^{n} \frac{p_j}{m}, \ldots, \sum_{j=1}^{n} \frac{p_j}{m})$.

We consider first a configuration where all the three algorithms provide very good results and LPT achieves a nearly optimal schedule. It is well known that LPT is almost optimal when the number of jobs is much larger than the number of processors. Hence in this first study case the number of processors was set to 80. Fig. 3(a) and (b) depict the outcome of this first configuration for two failure rates. Fig. 3(a) represents an optimistic case where the failure rates and tasks' length are relatively small with one failure per day and tasks' length ranging between 4 and 18 s with the RefSeq Mouse database. Fig. 3(b) shows a pessimistic case where the failure rate of each processor is 6 failures per day using the UniProtKB database. Bars show the ratio of the $C_{\max}$ and reliability to the lower bound. As we can observe in both figures the reliability ratio is close to the optimal when $C_{\max}$ is close to the optimal. This suggests that a good list scheduling for $C_{\max}$ minimization is good for reliability too. We notice also that when the failure rate increases we observe a bigger difference between LPT, SPT and Random. In fact, we can clearly observe in Fig. 3(b) that LPT is better than Random and SPT. We also point out that this figure shows that SPT is the worst in this scenario.



(a) 1 failure/day per processor. (b) 4 failures/day per processor. (c) 6 failures/day per processor.

Fig. 4. Ratio to the optimal C_{max} and reliability for increasing number of failures per day.

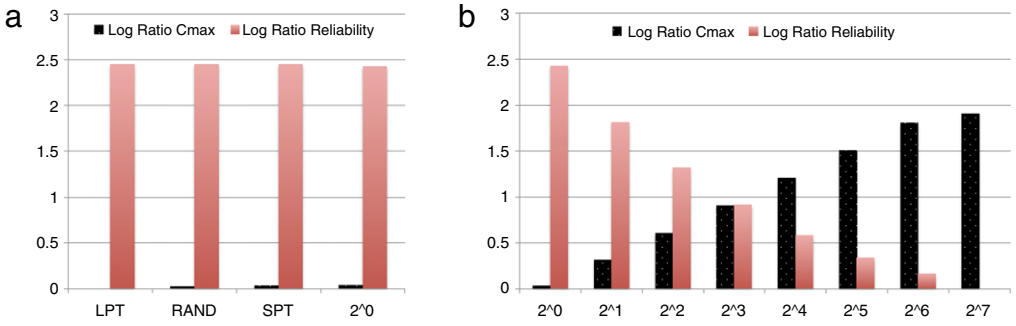


Fig. 5. Log of the ratio to the optimal C_{max} and reliability for 1 failure per day using 80 processors.

In order to better explore the impact of the scheduling policies on the reliability, we study the case where the LPT ratio is close to the worst case bound 4/3. To reach such a configuration, we consider a simulation where the number of tasks is not much larger than the number of processors. We also consider tasks with high execution times using only the UniProtKB database. Bars in Fig. 4(a)–(c) represent the outcome of this second simulation where the failure rate increases from 1 to 6 failures per day per processor.

This simulation is quite revealing in several ways that all strategies provide almost a very similar performance for the C_{max}, however the difference in reliability can be dramatic (in Fig. 4(c), reliability factor is equal to 1.7 for LPT versus 17 for SPT).

The results, as shown in Fig. 4(a)–(c), clearly indicate the dominance of LPT especially when the failure rate is high (6 failures per day). Bars in Fig. 4 confirm that the Random schedule outperforms SPT in terms of reliability, even with a C_{max} SPT slightly better (C_{max} ratio between SPT and Random is less than 1.03). This striking result is due to the fact that SPT leads to the most unbalanced completion time vector. This finding was unexpected and suggests that a solution with a good C_{max} does not lead necessarily to a better reliability. One of the more significant findings emerging from this study is that balanced scheduling strategies like LPT are more appropriate to optimize both the reliability and C_{max}.

7.2. DFR distributions

We conduct now a simulation to assess the tradeoff between the C_{max} and reliability when the failure distribution is *DFR*. We consider the values of failure shape and scale reported in [21] for the Weibull distribution. For this simulation, the shape parameter is fixed as 0.7 and the failure rate is one failure per day per processor. We investigate the performance ratio of the same list scheduling strategies as before: LPT, SPT and Random. In addition, we study the performance of the proposed strategy in Algorithm 3. Recall that the inputs of this algorithm are job set, processor set and the threshold τ.

In order to compare these scheduling algorithms, the performance ratio of each strategy is computed in regard to the optimal schedule for the *DFR* case where all jobs are affected to one processor.

In this simulation, we consider tasks from the RefSeq Mouse database using 80 processors. Bars in Fig. 5(a) represent the log of the ratio for both C_{max} and reliability for the list scheduling algorithms and Algorithm 3 using a threshold $\tau = 2^0 L_b$. Bars in Fig. 5(b) represent the performance of Algorithm 3 while using different values of threshold given by $2^x L_b$.

We observe from Fig. 5(a) that the scheduling algorithms achieve good makespan but unsatisfactory reliability, which is almost the same for all heuristics. In contrast, we observe from Fig. 5(b) that better reliability can be obtained but by means of a considerably higher C_{max} ratio (reliability is decreasing when C_{max} is increasing). We also observe that it is possible to obtain a configuration for which degradation for C_{max} and reliability are the same like when $\tau = 2^3 L_b$. Finally, we note that the reliability metric is much more sensitive than the C_{max}, which is due to the exponential nature of the Weibull distribution.

8. Conclusion

Real-world large parallel systems often exhibit non-memoryless failure distributions. In this work, we study scheduling strategies that consider both application completion time and the application's probability of failure. In the context of decreasing or increasing failure rates, we examine two main questions. First, can one optimize both reliability and makespan, or do we have to degrade one metric to improve the other? Second, what scheduling strategy can obtain an optimal or an approximate solution optimizing reliability and makespan?

Our main results are summarized as follows:

- If the failure rate is decreasing, then we prove that makespan and reliability are antagonistic. Hence, one must worsen one metric in order to improve the other. As there is no single optimal solution, we design an algorithm for computing the approximation set for the Pareto-optimal solutions.
- If the failure rate is increasing, then we show that one can optimize both makespan and reliability at the same time when processors are of identical speed. We prove also that the scheduling problem can be solved optimally for several common failure distributions, like the Weibull, by using a largest processing time (LPT) list scheduling algorithm.
- This is the first study showing that LPT is a very good strategy providing good performance for $C_{\max}$ and reliability as well.

The implications of these results are effective scheduling strategies that can accelerate parallel applications and/or improve their reliability over large distributed systems.

We conduct simulations that assess the impact that failures impose on the performance. First, we study the performance ratio of the list scheduling strategies when the failure rate is increasing (IFR). This simulation reveals that all strategies provide similar results in terms of $C_{\max}$, although they differ drastically in reliability. It is also shown that LPT dominates specially when the failure rate is high (6 failures per day).

Secondly, we consider the case when the failure rate is decreasing (DFR). In this simulation, we analyze Algorithm 3 and use different values of threshold. We observe that completion time and reliability are antagonistic objectives as expected in the theoretical analysis. We also notice that the reliability metric is much more sensitive than the $C_{\max}$.

As future direction, we would like to consider more general application models where there exist dependencies between tasks for both *DFR* and *IFR* distributions.

Appendix. Proof of Theorem 4

Theorem 4. Consider a completion time vector $\mathbf{c}$ of an arbitrary schedule and two arbitrary processors i and i' satisfying $s_{i'} < s_i$. Let δ represent the amount of workload to transfer from processor i' to processor i. This will produce the completion time vector $\hat{\mathbf{c}} = (\hat{C}_1, \ldots, C_i + \delta/s_i, \ldots, C_{i'} - \delta/s_{i'}, \ldots, C_m)$ such that $C_{i'} - \delta/s_{i'} \leq C_i + \delta/s_i$. Then $\mathbf{c}$ and $\hat{\mathbf{c}}$ satisfy $\mathbf{c} \prec^w \hat{\mathbf{c}}$.

Proof. Let l (or respectively h) be the rank of processor i (or respectively i') in the vector $\mathbf{c}$ with respect to an increasing order.

Suppose that after the transfer operation the new rank of processors i and i' become respectively l^+ and h^-.

The difference between the partial sum of vector $\mathbf{c}$ and $\hat{\mathbf{c}}$ in the first case where $h < l$ is given by

1. $\forall k \mid 1 \leq k < h^-,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = 0.$
2. $\forall k \mid h^- \leq k < h,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = C_{(k)} - \hat{C}_{(h^-)} \geq 0$ (since $\hat{C}_{(k+1)} = C_{(k)}$).
3. $\forall k \mid h \leq k < l,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = \delta/s_{i'}.$
4. $\forall k \mid l \leq k \leq l^+,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = C_{(l)} + \delta/s_{i'} - \hat{C}(k) \geq 0$ (since $C_{(l)} + \delta/s_{i'} \geq C_{(l^+)} \geq \hat{C}_{(k)}$).
5. $\forall k \mid^+ l \leq k \leq m,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = \delta(1/s_{i'} - 1/s_i) \geq 0.$

In the second case where $l < h$ and $h < l^+$.

1. $\forall k \mid 1 \leq k < h^-,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = 0.$
2. $\forall k \mid h^- \leq k < l,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = C_{(k)} - \hat{C}_{(h^-)} \geq 0$ (since $C_{(k)} = \hat{C}_{(k+1)}$).
3. $\forall k \mid l \leq k < h,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = C_{(l)} - \hat{C}_{(h^-)} \geq 0.$
4. $\forall k \mid h \leq k < l^+,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = C_{(l)} + \delta/s_{i'} - \hat{C}_{(k)} \geq 0.$
5. $\forall k \mid l^+ \leq k \leq m,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = \delta(1/s_{i'} - 1/s_i) \geq 0.$

In the last case where $l < h$ and $l^+ < h$ we have the same analysis except for the sub cases 3 and 4.

1. $\forall k \mid 1 \leq k < h^-,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = 0.$
2. $\forall k \mid h^- \leq k < l,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = C_{(k)} - \hat{C}_{(h^-)} \geq 0$ (since $C_{(k)} = \hat{C}_{(k+1)}$).
3. $\forall k \mid l \leq k < l^+,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = C_{(l)} - \hat{C}_{(h^-)} \geq 0.$
4. $\forall k \mid l^+ \leq k < h,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = C_{(k)} + C_{(l)} - (\hat{C}_{(l^+)} + \hat{C}_{(h^-)}) \geq 0$ (since $C_{(k)} = \hat{C}_{(k+1)}$).
5. $\forall k \mid h \leq k \leq m,\ \sum_{i=1}^{k} C_{(i)} - \sum_{i=1}^{k} \hat{C}_{(i)} = \delta(1/s_{i'} - 1/s_i) \geq 0.$

Thus, for all cases we obtain

$$\sum_{i=1}^{m} C_{(i)} - \sum_{i=1}^{m} \hat{C}_{(i)} \geq 0.$$

Therefore we obtain

$\mathbf{c} \prec^w \hat{\mathbf{c}}$. □

References

[1] N. Alon, Y. Azar, G.J. Woeginger, T. Yadid, Approximation schemes for scheduling on parallel machines, J. Sched. 1 (1) (1998) 55–66.
[2] D. Anderson, BOINC: a system for public-resource computing and storage, in: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA, 2004.
[3] I. Assayad, A. Girault, H. Kalla, Tradeoff exploration between reliability, power consumption, and execution time for embedded systems, Int. J. Softw. Tools Technol. Trans. (STTT) (2012) 1–17.
[4] R.E. Barlow, F. Proschan, L.C. Hunter, Mathematical Theory of Reliability, SIAM, 1996.
[5] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, M. Snir, Toward exascale resilience, Int. J. High Perform. Comput. Appl. 23 (4) (2009) 374.
[6] H. Casanova, A. Legrand, Y. Robert, Parallel Algorithms, Chapman & Hall, 2008.
[7] A.K. Chandra, C.K. Wong, Worst-case analysis of a placement algorithm related to storage allocation, SIAM J. Comput. 4 (1975) 249.
[8] C. Chekuri, M. Bender, An efficient approximation algorithm for minimizing makespan on uniformly related machines, J. Algorithms 41 (2) (2001) 212–224.
[9] EBI. Uniprotkb/swiss-prot homepage.
[10] T. Estrada, K. Reed, M. Taufer, Modeling job lifespan delays in volunteer computing projects, in: Proceedings of the IEEE/ACM International Symposium on Cluster Computing and Grid, 2009, pp. 331–338.
[11] M.R. Garey, D.S. Johnson, Strongnp-completeness results: motivation, examples, and implications, J. ACM (JACM) 25 (3) (1978) 499–508.
[12] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, WH Freeman & Co., New York, USA, 1979.
[13] A. Girault, H. Kalla, A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate, IEEE Trans. Dependable Secure Comput. 6 (4) (2009) 241–254.
[14] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell Syst. Tech. J. 45 (9) (1966) 1563–1581.
[15] R.L. Graham, E.L. Lawler, J.K. Lenstra, R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (2) (1979) 287–326.
[16] Doug Hains, Zach Cashero, Mark Ottenberg, Wim Bohm, Sanjay Rajopadhye, Improving CUDASW++, a parallelization of Smith–Waterman for cuda enabled devices, in: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Ph.D. Forum, IPDPSW, IEEE, 2011, pp. 490–501.
[17] E. Heien, D. Kondo, D. Anderson, Correlated resource models of Internet end hosts, in: 31st International Conference on Distributed Computing Systems, ICDCS, 2011.
[18] D.S. Hochbaum, D.B. Shmoys, Using dual approximation algorithms for scheduling problems theoretical and practical results, J. ACM (JACM) 34 (1) (1987) 144–162.
[19] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D.H.J. Epema, The grid workloads archive, Future Gener. Comput. Syst. 24 (7) (2008) 672–686.
[20] J.M. Jaffe, Efficient scheduling of tasks without full use of processor resources, Theoret. Comput. Sci. 12 (1) (1980) 1–17.
[21] B. Javadi, D. Kondo, J.M. Vincent, D.P. Anderson, Discovering statistical models of availability in large distributed systems: an empirical study of SETI@home, IEEE Trans. Parallel Distrib. Syst. 22 (11) (2010) 1896–1903.

[22] E. Jeannot, E. Saule, D. Trystram, Optimizing performance and reliability on heterogeneous parallel systems: approximation algorithms and heuristics, J. Parallel Distrib. Comput. (2012).
[23] G. Lin, R. Rajaraman, Approximation algorithms for multiprocessor scheduling under uncertainty, in: Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, 2007, pp. 25–34.
[24] J.W.S. Liu, C.L. Liu, Bounds on Scheduling Algorithms for Heterogeneous Computing Systems, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1974.
[25] Yongchao Liu, Douglas L. Maskell, Bertil Schmidt, CUDASW++: optimizing Smith–Waterman sequence database searches for CUDA-enabled graphics processing units, BMC Res. Notes 2 (1) (2009) 73.
[26] Yongchao Liu, Adrianto Wirawan, Bertil Schmidt, CUDASW++ 3.0: accelerating Smith–Waterman protein database search by coupling CPU and GPU SIMD instructions, BMC Bioinformatics 14 (1) (2013) 117.
[27] G. Malewicz, Parallel scheduling of complex dags under uncertainty, in: Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2005, pp. 66–75.
[28] A.W. Marshall, I. Olkin, Theory of Majorization and its Applications, Academic Press, New York, 1979.
[29] NCBI. RefSeq homepage.
[30] D. Nurmi, J. Brevik, R. Wolski, Modeling machine availability in enterprise and wide-area distributed computing environments, 3648 (2005) 612.
[31] C.H. Papadimitriou, M. Yannakakis, On the approximability of trade-offs and optimal access of web sources, in: IEEE Proceedings of the 41st Annual Symposium on Foundations of Computer Science, 2000, pp. 86–92.
[32] R.K. Sahoo, M.S. Sivasubramaniam, M. Squillante, Y. Zhang, Failure data analysis of a large-scale heterogeneous server environment, in: Proceedings of DSN'04, 2004.
[33] B. Schroeder, G.A. Gibson, A large-scale study of failures in high-performance computing systems, IEEE Trans. Dependable Secure Comput. 7 (4) (2010) 337–351.
[34] Jaideeep Singh, Ipseeta Aruni, Accelerating Smith–Waterman on heterogeneous CPU–GPU systems, in: 2011 5th International Conference on Bioinformatics and Biomedical Engineering, iCBBE, IEEE, 2011, pp. 1–4.

Mohamed Slim Bouguerra was born in 1983 in Tunis, Tunisia.

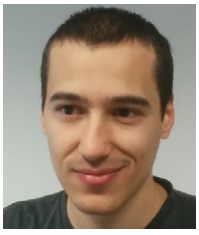
He obtained the Master degree in computer science from Ecole Supérieur des Sciences et Techniques de Tunis in 2008.

Slim holds a Ph.D. degree in computer science obtained at Grenoble University in 2012 and he is currently working as a postdoc at Argonne National Laboratory.

His research interests include fault-tolerance, reliability optimization and scheduling for large-scale parallel processing.

Derrick Kondo is a tenured research scientist at INRIA, France. He received his Bachelor's at Stanford University in 1999, and his Master's and Ph.D. at the University of California at San Diego in 2005, all in computer science. His general research interests are in the areas of reliability, fault-tolerance, statistical analysis, job and resource management.

His research projects are supported by national, European, and industrial grants. In 2009, he received a Young Researcher Award (similar to NSF's CAREER Award). He received an Amazon Research Award in 2010, and Google Research Award in 2011. He is the co-founder of the Failure Trace Archive, which serves as a public repository of failure traces and algorithms for distributed systems. He has won numerous best paper awards at IEEE/ACM conferences for work in these projects.



Fernando Mendonca graduated from the University of Brasilia in 2010.

During his Master Degree, he worked with Alba Cristina de Melo. His work was in the area of High Performance Computing using Heterogeneous Platforms. It involved the design of algorithms to parallelize DNA sequence comparisons on parallel platforms composed of CPUs and GPUs.

Currently, he is enrolled as a Ph.D. student in the University of Grenoble, France with Denis Trystram. His study subject is the design of efficient scheduling algorithms targeting the future exascale platforms.

Denis Trystram is a professor at Grenoble Institute of Technology since 1991 and is now a distinguished professor in this Institute.

Since 2010, he is a senior member of the Institut Universitaire de France. He is the vice-director of the LIG laboratory in Grenoble where he is leading a research group on resource optimization on parallel and distributed platforms.

Denis' current research activities concern the design and analysis of efficient approximation algorithms for multi-objective scheduling problems and Game Theory applied to parallel and distributed processing. He is interested in problems concerning reliability and energy optimization.

Denis is involved in the editorial board of Parallel Computing, Journal of Parallel and Distributed Computing and other journals. He also served 4 years in IEEE TPDS and in the program committees of the major international conferences in the field. He has published several books, more than 80 papers in international journals and twice more contributions in international conferences.