Resource-Constrained Scheduling Algorithms for Stochastic Independent Tasks With Unknown Probability Distribution

Yiqin Gao, Yves Robert, Frédéric Vivien

Resource-Constrained Scheduling Algorithms for Stochastic Independent Tasks With Unknown Probability Distribution

Yiqin Gao[1], Yves Robert[2,3*] and Frédéric Vivien[2]

[1]Shanghai Jiao Tong University, China.
[2*]Laboratoire LIP, ENS Lyon & Inria, France.
[3]University of Tennessee Knoxville, USA.

*Corresponding author(s). E-mail(s): Yves.Robert@ens-lyon.fr;
Contributing authors: gaoyiqin95@gmail.com;
Frederic.Vivien@inria.fr;

Abstract

This work introduces scheduling algorithms to maximize the expected number of independent tasks that can be executed on a parallel platform within a given budget and under a deadline constraint. The main motivation for this problem comes from imprecise computations, where each job has a mandatory part and an optional part, and the objective is to maximize the number of optional parts that are successfully executed, in order to improve the accuracy of the results. The optional parts of the jobs represent the independent tasks of our problem. Task execution times are not known before execution; instead, the only information available to the scheduler is that they obey some (unknown) probability distribution. The scheduler needs to acquire some information before deciding for a cutting threshold: instead of allowing all tasks to run until completion, one may want to interrupt long-running tasks at some point. In addition, the cutting threshold may be reevaluated as new information is acquired when the execution progresses further. This work presents several algorithms to determine a good cutting threshold, and to decide when to re-evaluate it. In particular, we use the Kaplan-Meier estimator to account for tasks that are still running when making a decision. The efficiency of our algorithms is assessed through an extensive set of simulations with various budget and deadline values, and ranging over 13 probability

distributions. In particular, the AutoPerSurvival(40%,0.005) strategy is proved to have a performance of 77% compared to the upper bound even in the worst case. This shows the robustness of our strategy.

Keywords: stochastic tasks, independent tasks, imprecise computation, resource allocation, scheduling, Kaplan-Meier estimator

1 Introduction

This paper focuses on the design of scheduling strategies to maximize the expected number of successfully executed tasks on a parallel platform composed of identical processors. The execution time of the tasks is not known before execution. The only information known by the scheduler is that these execution times are independent and identically distributed (IID) random variables obeying the same probability distribution, but this distribution is unknown. The scheduler has both a deadline constraint d and a budget constraint b. At any time, and on each enrolled processor, the scheduler can decide whether to interrupt a long-running task T to start a new task T', with the hope that T' will have an execution time shorter than the remaining execution time of T. However, there is a big risk involved with such a decision because: (i) the time and budget spent to execute T until its interruption are completely lost; and (ii) T' may well happen to have an execution time longer than the remaining execution time of T.

In this non-clairvoyant setting, what is the optimal strategy? Intuitively, the scheduler must first decide how many processors to enroll. Then, the scheduler needs to acquire some information about task execution times by letting several tasks run until completion on each processor. At some point, the scheduler synthesizes the information acquired so far and decides for a scheduling policy. This policy could be either to allow all tasks to run until completion, or to define a cutting threshold τ after which every long-running task should be interrupted. The cutting threshold τ can be recomputed dynamically as the execution progresses until the deadline d is reached or the budget b is exhausted, whichever comes first. Each of the above decisions involves a complicated trade-off. Indeed, deciding for the threshold early avoids consuming a significant fraction of the deadline and of the budget before interrupting any task. But this can lead to an imprecise estimation because the threshold value is based on little information. On the contrary, deciding for the threshold later during the execution leads to making a more accurate decision, at the risk of having wasted resources unduly. This work introduces several strategies to determine a good threshold, and at the right moment in the execution.

This scheduling problem has the (somewhat non-standard) objective to maximize the expected number of successful tasks with a given budget and deadline. Not all tasks will be successfully executed in the end: some tasks will be interrupted, and some tasks will never be launched. This problem is very

closely related to *imprecise computations* [2, 9, 23]. In imprecise computations, it is not necessary for all tasks to be completely processed to obtain a meaningful result: tasks are divided into a mandatory and an optional part: while the execution of all mandatory parts is necessary, the execution of optional parts is decided by the user. Often the user has neither the time nor the budget to execute all optional parts, and they must select which ones to execute. Our work perfectly corresponds to the optimization of the processing of the optional parts.

Among domains where tasks may have optional parts (or some tasks may be entirely optional), one can cite recognition and mining applications [26], robotic systems [17], speech processing [11], and [21] also cite multimedia processing, planning and artificial intelligence, and database systems. In these applications, the processing times of the optional parts are of similar nature but are heavily data-dependent, hence it is very natural to model them via a probability distribution $\mathcal{D}$. However, this probability distribution $\mathcal{D}$ is unknown before processing, and can be only determined through sampling many tasks. Unfortunately, in our scheduling problem, letting the scheduler sample many tasks without interruption to learn, say, the mean and standard deviation of the distribution, can prove very costly: it will consume a significant part of the budget and will prove suboptimal for any distribution requiring a small cutting threshold τ, such as lognormal distributions (see below).

This paper builds upon our previous work [6] where we tackle the dramatically simpler problem where the distribution $\mathcal{D}$ is known. In that case, we proposed an analytical method to compute the optimal threshold τ. Section 5.1 provides background material on this method. For some distributions, the optimal strategy is to never interrupt any task ($\tau = +\infty$), while for some others, such as some lognormal distributions, there is an optimal cutting threshold.

Regardless, when the distribution $\mathcal{D}$ is known, the approach in [6] provides an asymptotically optimal solution. The main focus of this paper is to investigate efficient strategies when the distribution $\mathcal{D}$ is unknown. To the best of our knowledge, this work constitutes the first attempt to address this challenging problem.

The major contributions of this work are the following:

- We provide a detailed study of bimodal exponential distributions, for which we exhibit a wide range of cutting thresholds. Small changes in the parameters of the distribution have a big impact on the cutting threshold, which nicely illustrates the difficulty of the scheduling problem with unknown distributions.
- We design a set of scheduling heuristics that use different estimators of the cutting threshold τ, and that refine this estimation periodically as the execution progresses.
- We show how to use to the Kaplan-Meier estimator [20] to account for long-running tasks when estimating the threshold τ.
- We introduce several methods for deciding when to compute and recompute the threshold.

- We report a comprehensive set of simulation results that compare the heuristics for various budget and deadline values, using up to 13 different probability distributions.

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 provides an introduction to the Kaplan-Meier estimator. We detail the framework and objective in Section 4. In Section 5, we provide background on prior strategies for interrupting tasks when the distribution is known (Section 5.1), together with a set of new results for Exponential distributions (Section 5.2). We provide new scheduling heuristics when the distribution is unknown in Sections 6, 7 and 8: Section 6 is devoted to methods for computing the cutting threshold accurately, while Sections 7 and 8 focus on different heuristics of when to recompute it. In Sections 7 and 8, we assess also the performance of our heuristics through an extensive set of simulation parameters. Finally, we provide concluding remarks and directions for future work in Section 9.

2 Related work

We first overview studies dealing with bags of tasks in Section 2.1. Then we survey work related to non-clairvoyant scheduling in Section 2.2. Next we discuss related task models such as imprecise computations in Section 2.3.

2.1 Bags of tasks

A bag of tasks is an application composed of a set of independent tasks sharing some common characteristics: either all tasks have the same execution time or they are instances sampled from the same probability distribution. The survey [34] deals with resource optimization for bag of tasks applications, but does not consider the non-clairvoyant case. Several works devoted to bag-of-tasks processing explicitly target cloud computing [4, 7, 15, 29]. These works consider the classical clairvoyant model, in which we know the exact execution times of the tasks, or their probability distribution, or maybe only their range and standard deviation. Vecchiola et al. [36] consider a single application comprising independent tasks with deadlines but without any budget constraints. In their model, tasks are supposed to have different execution times but they only consider the average execution time of tasks rather than its probability distribution (this is left for future work). Moreover, they do not report on the amount of deadline violations. Mao et al. [25] consider both deadline and budget constrained provisioning and assume they know the tasks execution times up to some small variation (the largest standard deviation of a task execution time is at most 20% of its expected execution time). Hence, this work is more related to scheduling under uncertainties than to non-clairvoyant scheduling.

2.2 Non-clairvoyant scheduling

The work surveyed in Section 2.1 assumes a fully or semi clairvoyant set of task execution times, which is not realistic in many applicative scenarios. In contrast, our model considers a fully non-clairvoyant case, in which we have no information in advance about the execution times of our bag of tasks. Although this topic has received less attention, we can still find several references. For instance, Im et al. [18] and Pawan et al. [33] both worked on online algorithms. They assume that the size of arriving tasks is not known before completing them. In [18], a unified model is designed for several different scheduling problems, while [33] aims at minimizing flow-time and energy. In the work of Li [22], task execution times are unknown, and the objective is to minimize the makespan while using one or several multicore processors. A group of authors [27–29] has published several studies focusing on budget-constrained makespan minimization. They do not assume to know the distribution of execution times but try to learn it on the fly [27, 28]. This work differs from ours as these authors do not consider deadlines. For instance, in [29], the objective is to try to complete all tasks, possibly using replication on faster processors, and, in case the proposed solution fails to achieve this goal, to complete as many tasks as possible. The implied assumption is that all tasks can be completed within the budget. We implicitly assume the opposite: there are too many tasks to complete all of them by the deadline, and therefore we attempt to complete as many as possible; we avoid replication, which would be a waste of resources in our framework.

2.3 Task models

As mentioned in Section 1, our model assumes that some tasks may not be executed, which is very closely related to *imprecise computations* [2, 9, 23]. Furthermore, this task model also corresponds to the overload case of [3] where jobs can be *skipped* or *aborted*. Another related model, is that of *anytime tasks* [19] where a task can be interrupted at any time, with the assumption that the longer the running, the higher the quality of its output. Such a model requires a function relating the time spent to a notion of reward. Finally, we note that the general problem related to interrupting tasks falls into the scope of optimal stopping, the theory that consists in selecting a date to take an action, in order to optimize a reward [12].

3 The Kaplan-Meier estimator

In medical research, biostatisticians have to answer questions like: "What is the probability that a patient will still be alive 5 years after receiving a cancer diagnosis?" To answer such a question, biostatisticians analyse the data of many individual patients. Some of these data will be complete: they will have both the time of diagnosis and the time of death of the patient. However, at the time of the analysis, some patients enrolled in the dataset will still be alive.

The status of some other patients may be unknown because contact with them has been lost (e.g., they have moved away). In both cases observations are incomplete. One only knows the time of diagnosis and the last time the patient was known to be alive. Hence, one only knows a lower bound on the time the patient has survived after the diagnosis. These incomplete "lower-bound" data are called *right-censored data* and the question addressed by biostatisticians is that of *survival analysis with right-censored data*. This problem is exactly our scheduling problem, only the vocabulary changes:

- instead of survival times, we have execution times;
- instead of diagnosis times, we have start times;
- at the time of analysis, instead of patients still alive, we have tasks still running;
- at the time of analysis, instead of patients with unknown whereabouts, we have tasks that have been terminated by the scheduler before completion.

We can therefore use the well-known tool to solve survival analysis with right-censored data, namely the Kaplan-Meier estimator [1, 20]. We provide a detailed example in Section 6.2.

We refer the interested reader to [1] for a thorough overview of survival and event history analysis. Survival analysis with the Kaplan-Meier estimator is widely used in biostatistics [8, 16, 37], and in a variety of other domains such as engineering [31], economics [24], etc. To the best of our knowledge, this work is the first to use the Kaplan-Meier estimator for scheduling bags of tasks, let alone for any problem involving scheduling decisions. In core computer science, the Kaplan-Meier estimator seems having been used only for the analysis of software projects (see [30, 32] and the references therein).

Furthermore, the present study appears to be unique because it uses a fully non-clairvoyant framework and assumes an overall deadline in addition to a budget constraint. Our previous works [5, 14] had the same setting under homogeneous [5] or heterogeneous [14] platforms. But in these works, we assumed that the distribution of execution times was known in advance, while the key problem studied in the current paper is to learn the distribution of task execution times on the fly and to decide when interrupting unfinished tasks.

4 Problem definition

We consider a parallel platform composed of M identical processors. The execution time of a task on a processor obeys an unknown probability distribution $\mathcal{D}$. Without loss of generality, we assume that it costs one budget unit to execute a task for one second on any processor, and we have a total budget b and an overall deadline d. Thus the budget can also refer to the available *machine time*.

At any time, and on each enrolled processor, the scheduler can decide whether to interrupt a long-running task to start a new task, with the hope that the new one will have an execution time shorter than the remaining execution time of the old one. However, we assume that execution is non-preemptive: if

the execution of a task is interrupted, it cannot be restarted, and all the work done (and the budget spent) so far for that task is lost.

Our objective is to maximize the total number of tasks successfully completed under the budget and deadline limits. To drive the design of our scheduling policies, we use an instantaneous version of this objective, namely the yield, which is defined as the expected number of tasks completed per unit of budget spent. Because one unit of budget corresponds to one second of execution, the yield is also equal to the expected number of tasks completed per second.

All scheduling policies are required to have polynomial complexity. Since a solution to the problem is the list of the tasks that are executed, either partially or successfully (for each of these tasks, the scheduler made a decision), the size of the problem is proportional to that number of tasks. This number in turn is proportional to the budget (or deadline), divided by the expectation of the (unknown) probability distribution $\mathcal{D}$, since the average execution time until completion of a task is $\mu(\mathcal{D})$. Furthermore, the scheduling policies will make decisions and compute a cutting threshold several times during the whole execution; we require that the number of such decisions be constant, and they will typically be taken each time a prescribed percentage of the budget is spent. The motivation here is to cap the overhead incurred by the scheduler by forbidding to recompute a threshold each time a new task is executed.

5 Optimal strategies for known distributions

In Section 5.1, we recall previous results for known distributions, namely an asymptotically optimal policy for discrete distributions and its extension to continuous distributions [5, 6]. Then, in Section 5.2, we study the case where the distribution of task execution times is defined by a bimodal exponential distribution. This study shows how small changes in distribution parameters can lead to drastically different optimal scheduling policies.

5.1 Background on previous approaches

A scheduling policy has to decide whether all tasks should be allowed to run until completion, or whether some tasks should be interrupted and, in the latter case, which tasks and when? In [5, 6] we provided answers to this key question. We review the approach to determine a cutting threshold, first for discrete distributions of task execution times (Section 5.1.1), and then for continuous distributions (Section 5.1.2).

In addition to determining a cutting threshold, the scheduler should decide how many processors to enroll. With a budget of b and a deadline of d, we enroll $\lceil \frac{b}{d} \rceil$ processors. The rationale is that this is the minimum number of processors required to exhaust the budget. Because the policy on each processor is asymptotically optimal (see below) enrolling more processors will not be beneficial for large budgets, and could lead to waste due to budget fragmentation for smaller budgets.

5.1.1 Discrete distributions

We consider a discrete distribution $\mathcal{D}$ under which there are k possible task execution times, $w_1 < w_2 < ... < w_k$. A task has an execution time w_i with probability p_i, with $0 \leq p_i \leq 1$ and $\sum_{j=1}^{k} p_j = 1$. The simplest policies that interrupt task executions are the *fixed-threshold strategies*. A fixed-threshold strategy interrupts every not-yet-completed task at a predefined threshold τ, i.e., when the task has been executing for a time τ without completing. The yield of the fixed-threshold strategy of threshold τ is computed as follows:

$$\mathcal{Y}(\tau) = \begin{cases} 0 & \text{if } \tau < w_1 \\ \dfrac{\sum_{j=1}^{\mathbb{I}(\tau)} p_j}{\sum_{j=1}^{\mathbb{I}(\tau)} p_j w_j + \left(1 - \sum_{j=1}^{\mathbb{I}(\tau)} p_j\right)\tau} & \text{otherwise} \end{cases} \tag{1}$$

where $\mathbb{I}(\tau)$ is the index of the largest task execution time smaller than or equal to τ: $\mathbb{I}(\tau) = k$ if $\tau \geq w_k$, and $w_{\mathbb{I}(\tau)} \leq \tau < w_{\mathbb{I}(\tau)+1}$ otherwise. This complicated formula has an intuitive explanation: the probability of success with cutting threshold τ is $\sum_{j=1}^{\mathbb{I}(\tau)} p_j$, and the execution time is averaged as follows: some tasks have (successfully) executed in w_j seconds, with probability p_j, for each $j \leq \mathbb{I}(\tau)$, and the remaining tasks have been interrupted after τ seconds (with the remaining probability $\left(1 - \sum_{j=1}^{\mathbb{I}(\tau)} p_j\right)$). The following theorem states that the best fixed-threshold strategy is asymptotically optimal when the platform includes a single processor ($M = 1$) [5, 6].

Theorem 1 *Let $\tau_{opt} = \arg\max_{\tau \in \{w_1,...,w_k\}} \mathcal{Y}(\tau)$. If the platform includes a single processor, the fixed-threshold strategy of threshold τ_{opt} is asymptotically optimal among all possible strategies when the budget tends to infinity (the deadline being equal to the budget).*

With several processors available, we enroll $\lceil \frac{b}{d} \rceil$ processors and execute on each of them the fixed-threshold strategy of threshold τ_{opt}.

5.1.2 Continuous distributions

We now consider a continuous distribution $\mathcal{D}$ of task execution times whose cumulative distribution function is $F(x)$ and its probability density function $f(x)$. The execution time of a task is thus defined by a random variable X which follows $\mathcal{D}$. With these notations, the probability that the execution is no longer than a duration t is: $P(X \leq t) = F(t)$. Then, the equation of the yield of the fixed-threshold strategy of threshold τ is easily extrapolated from that for discrete distributions (Equation 1):

$$\mathcal{Y}(\tau) = \frac{F(\tau)}{\int_0^{\tau} x f(x) dx + \tau(1 - F(\tau))} \tag{2}$$

The optimal threshold is then, like previously:

$$\tau_{\text{opt}} = \arg\max_{\tau} \mathcal{Y}(\tau).$$

5.2 New results for exponential distributions

To illustrate the fact that small differences in the distribution of task execution times can lead to dramatically different optimal policies, we study the case where task execution times follow exponential distributions. We assume that the distribution is either unimodal or bimodal, but we formally express it as a bimodal case: the unimodal case will appear as a special case where both modes coincide.

Task execution times are thus defined by a bimodal exponential distribution of parameters λ and μ, chosen with respective weights p and $1-p$, where $0 \leq p \leq 1$. In other words, each time we need to generate a new task execution time, with probability p we generate an execution time using an exponential distribution of parameter λ and with probability $1-p$ we generate an execution time using an exponential distribution of parameter μ. Without loss of generality, we assume that $\mu \geq \lambda$. A potential problem with exponential distributions is that task execution times can be arbitrarily small. This seems unrealistic: independently of the task size, the system requires some time to load (part of) the code of the task and prepare for execution. Furthermore, the possibility of arbitrarily small execution times can lead to pathological situations (for instance, see Theorem 2 below). Therefore, one may want to add a positive constant time δ to the sum of the random variables (one can always set $\delta = 0$ if one does not want to add such a constant). In this context, δ can be seen as the lower bound on any task execution time. Altogether, this leads to the following density function for the distribution of probability:

$$f(t) = \begin{cases} 0 \text{ if } t < \delta \\ p\lambda e^{-\lambda(t-\delta)} + (1-p)\mu e^{-\mu(t-\delta)} \text{ otherwise.} \end{cases} \tag{3}$$

Theorem 2 defines the optimal cutting threshold for a fixed-threshold strategy for the distribution of task execution times whose density obeys Equation 3. Its proof can be found in the appendix.

Theorem 2 *When task execution times are defined by a bimodal exponential distribution plus a nonnegative constant, the optimal cutting threshold τ_{opt} and the optimal yield $\mathcal{Y}_{opt}$ are as follows:*

- *If the constant is null ($\delta = 0$)*
  - *If there is a single mode, any value for the threshold is optimal and $\mathcal{Y}_{opt} = \lambda$.*
  - *If the two modes are distinct, $\tau_{opt} = 0$ (tasks should be interrupted as soon as possible), and $\mathcal{Y}_{opt} = p\lambda + (1-p)\mu$.*
- *If the constant is* not *null ($\delta > 0$)*

- *If* $\delta\lambda p - p(1-p)\frac{\mu-\lambda}{\mu} \geq 0$, *then* $\tau_{opt} = +\infty$ *(tasks should never be interrupted), and* $\mathcal{Y}_{opt} = \frac{1}{\delta + \frac{p}{\lambda} + \frac{(1-p)}{\mu}}$.
- *If* $\delta\lambda p - p(1-p)\frac{\mu-\lambda}{\mu} < 0$, τ_{opt} *is the unique solution in* $]0; +\infty[$ *of the equation:*

$$p(1-p)\frac{\mu-\lambda}{\lambda\mu}\left(-\lambda\left(1-e^{-\mu l}\right)+\mu e^{-\mu l}\left(e^{\lambda l}-1\right)\right) + \delta\left(\lambda p+\mu(1-p)e^{-(\mu-\lambda)l}\right)=0$$

This equation should be solved numerically and its solution should be injected in Equation 2 to obtain the value of $\mathcal{Y}_{opt}$.

Certainly the most striking (and counter-intuitive) part of this theorem is the case where $\delta = 0$ (tasks can be arbitrarily small) and when the distribution is truly bimodal ($\lambda \neq \mu$ and $p(1-p) \neq 0$). The result states that $\tau_{\text{opt}} = 0$. This means that the lower the threshold, the better. But, obviously, a task must be launched for having any chance to complete. This result means that each task should be interrupted as soon as possible if it has not yet completed. When the constant δ is not null, however small it is, results are drastically different: depending on the relationships between the parameters (p, λ, μ, and δ), either no task should be interrupted or there is a single optimal cutting threshold (and it is not trivial: $0 < \tau_{\text{opt}} < +\infty$).

One may wonder whether Theorem 2 really matters, that is, whether the yield significantly varies with the cutting threshold. Consider two equiprobable modes ($p = 0.5$) with a constant $\delta = 0.001$, with $\lambda = 1$, and $\mu = 50$. If we never interrupt tasks the yield is approximatively 1.96. If we interrupt them with a cutting threshold of 0.01, the yield is 20.35, more than 10 times larger! There are distributions for which using the optimal cutting threshold has a dramatic impact on the performance of the system.

6 Threshold estimation for unknown distributions

As stated in Section 5, when the distribution of task execution times is known, the optimal policy is a fixed-threshold strategy that interrupts tasks, and the choice of the cutting threshold can have a very significant impact on the system performance. Now the question is: how do we find the optimal cutting threshold when the distribution is unknown?

In order to acquire information on the distribution of task execution times, the unique option is to execute some tasks and record their execution times. We consider the problem of deciding how many tasks to execute in Sections 7 and 8. For the sake of the argument, let us assume for now that we have already launched the execution of several tasks, that some executions have already completed, some are still running, and some were interrupted. For instance, in the toy example presented on Figure 1 we have two processors, four tasks, and

we want to take a decision at time 20. One task has executed for 5 seconds, one for 16; two tasks have not yet completed (the tasks in red), having run, respectively, for 15 and 4 seconds so far. How do we estimate the distribution of task execution times based on this data?

Fig. 1 Toy example with two processors, two successfully completed tasks (in blue) and two not-yet-completed tasks (in red) at time 20.

There are two types of approaches. In the first type, we would try to guess some characteristics of the distribution. For instance, we could claim that "task execution times likely follow an exponential distribution". Then, we would look for the exponential distribution that better fits the data, for instance using a maximum likelihood estimation. If our initial guess was lucky, we should end up with a good result. However, the underlying distribution may be either a lognormal distribution, or a multimodal one, or even not resemble any of the most used probability distributions. Rather than relying on potentially unlucky guesses, we aim at designing a robust approach delivering high quality results regardless of the underlying distribution. Therefore, our approach belongs to the second type of approaches, sometimes called "nonparametric" statistics. We are not going to make any assumption on the underlying distribution.

In Section 6.1, we start from a naive approach that only considers the execution times of tasks that have completed. This approach has the advantage of simplicity. However, as exemplified by the toy example on Figure 1, it can ignore a significant share of the data, and in particular long-running tasks. As discussed in Section 3, the question on how to take into account tasks that have not yet completed has been thoroughly research in the field of medical research. In Section 6.2, we show how to use the Kaplan-Meier estimator to solve our scheduling problem more accurately.

6.1 The empirical distribution function

The naive approach only considers the execution times of completed tasks and uses the associated *empirical distribution function* [35], along with Equation (1). Consider an example where there are k different task execution times, $w_1 < w_2 < ... < w_k$, and where n_i tasks have the execution time w_i. Then, using the empirical distribution function, a task has an execution time w_i with probability $p_i = \frac{n_i}{\sum_{j=1}^{k} n_i}$. Using these probabilities, we search in the set $\{w_1, ..., w_k\}$ the value maximizing the yield, using Equation 1.

The main advantage of this approach is its simplicity. The toy example on Figure 1 illustrates its main drawback: there maybe many tasks whose information is ignored, namely the tasks that have not yet completed or that

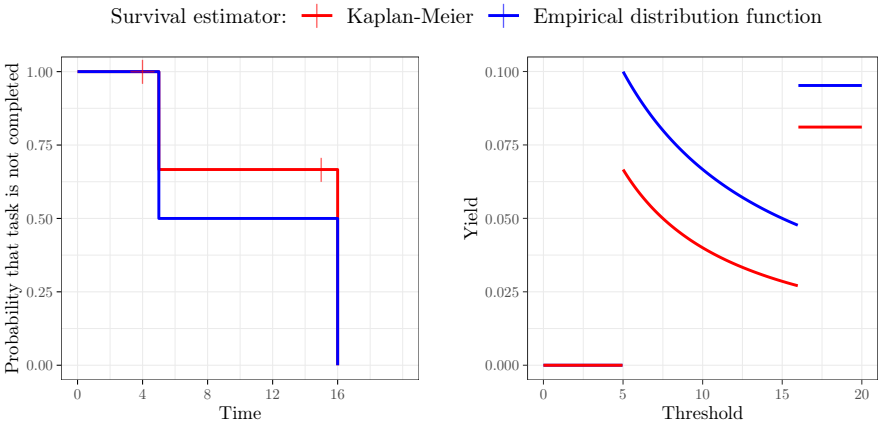

Fig. 2 Probability of survival (left) and yield (right) for the toy example of Figure 1 when using the empirical distribution function (blue) or the Kaplan-Meier estimator (red).

are already interrupted. This drawback induces a bias by ignoring interrupted or long-running tasks.

6.2 Using the Kaplan-Meier estimator for survival analysis

This section shows how to use the Kaplan-Meier estimator initially designed for survival analysis with right-censored data [1, 20]. Consider an example where there are k different task execution times, $w_1 < w_2 < ... < w_k$. Here, execution times can be the execution times of tasks that have completed, like the values 5 and 16 in the example of Figure 1. They can also be censored execution times, like the values 4 and 15 in that example. Let d_i be the number of tasks that *die* at time w_i, that is, the number of tasks whose execution time is exactly w_i. Let r_i be the number of individual *at risks* just prior to time w_i, that is, the number of tasks whose execution time is greater than or equal to w_i. The survival function, $\mathcal{S}(t)$, is the probability that life is longer than t: $\mathcal{S}(t) = Pr(X > t)$. The Kaplan-Meier estimator gives us:

$$\mathcal{S}(t) = \prod_{w_i \leq t} \left(1 - \frac{d_i}{r_i}\right). \tag{4}$$

Using this estimator, we can then rewrite Equation 1 as:

$$\mathcal{Y}(t) = \frac{1 - \mathcal{S}(t)}{\sum_{j=1}^{\mathbb{I}(t)} (\mathcal{S}(w_{i-1}) - \mathcal{S}(w_i)) w_j + \mathcal{S}(w_{\mathbb{I}(t)}) t}$$

where $\mathbb{I}(t)$ is the index of the largest task execution time smaller than or equal to t: $\mathbb{I}(t) = k$ if $t \geq w_k$, and $w_{\mathbb{I}(t)} \leq t < w_{\mathbb{I}(t)+1}$ otherwise (with $w_0 = 0$ and $\mathcal{S}(w_0) = 1$).

We illustrate this estimator with the toy example of Figure 1:

w_i	d_i	r_i	$1 - \frac{d_i}{r_i}$	$\prod_{j \leq i} \left(1 - \frac{d_j}{r_j}\right)$
4	0	4	1	1
5	1	3	$\frac{2}{3}$	$\frac{2}{3}$
15	0	2	1	$\frac{2}{3}$
16	1	1	0	0

The resulting function is presented in red on the left-hand side of Figure 2, alongside the probabilities associated to the empirical distribution function (in blue). Red ticks indicate the presence of censored data. For the empirical distribution function, the probability that the execution time of a task exceeds 5 seconds is 50%, while it is 66.6% for the Kaplan-Meier estimator. When we plug these different probability functions in Equation 1, we obtain the yields depicted on the right-hand side of Figure 2. In this toy example, the empirical distribution function claims that the optimal cutting threshold is 5, when the survival analysis claims that it is 16.

Note that, in the product of Equation (4), only the times corresponding to actual (non-censored) execution times matter. Execution times that only correspond to censored times each contribute a value of 1 in the product (see the table above). Note also that if there is no censored data, we have $r_{i-1} - r_i = d_{i-1}$ and $\mathcal{S}(t)$ simplifies into

$$\mathcal{S}(t) = \prod_{w_i \leq t} \frac{r_i - d_i}{r_i} = \prod_{w_i \leq t} \frac{r_{i+1}}{r_i} = \frac{r_j}{r_0}$$

where j is the smallest index such that $w_j > t$. In other words, when there is no censored data, the empirical distribution function and the Kaplan-Meier estimator coincide.

We can use the survival function to compute the mean and variance of the execution times. Recall that $\mathcal{S}(t) = Pr(X > t)$, hence $Pr(X = w_j) = Pr(X \in]w_{j-1}, w_j]) = \mathcal{S}(w_{j-1}) - \mathcal{S}(w_j))$ for $1 \leq j \leq k$ (with $w_0 = 0$ and $\mathcal{S}(w_0) = 1$, as stated above). We derive that:

$$\mu = \mathbb{E}[X] = \sum_{j=1}^{k} (\mathcal{S}(w_{j-1}) - \mathcal{S}(w_j))w_j + \mathcal{S}(w_k)w_k.$$

$$\sigma^2 = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \sum_{j=1}^{k} (\mathcal{S}(w_{j-1}) - \mathcal{S}(w_j))w_j^2 + \mathcal{S}(w_k)w_k^2 - \mu^2.$$

7 Static algorithm

In Section 6, we have shown how to use available data from task execution times to define the best cutting threshold. In this section, we focus on how and when to acquire the data needed to compute a cutting threshold, possibly many different times as the execution progresses.

In order to acquire information on the distribution of task execution times, the only solution is to execute some tasks and to record their execution times. In this process, we have to make a classical trade-off. On the one hand, we should execute a sufficiently large number of tasks until completion, in order to be sure that the set of observed execution times is indeed representative of the underlying distribution. On the other hand, we should execute as few tasks as possible before making a decision, to avoid wasting a significant share of the budget on running tasks until completion if the optimal threshold is a "short" one.

We start by designing policies that try to guess the good trade-off before launching any task, and we assess their performance through simulation. In Section 8, we refine the approach and present a policy that tries to automatically infer the cutting trade-off.

7.1 *One-size-fits-all* heuristics

The simplest strategies try to guess, without interrupting any task, the "right" trade-off. Consider a strategy that spends 10% of the overall budget running tasks up to completion before computing the optimal threshold: one can still hope to achieve a 90% overall efficiency if the threshold has been accurately evaluated. This is the basis of the first two strategies:

1. pick a priori a percentage p;
2. run tasks on processors until having spent the fraction $p \times b$ of the overall budget;
3. compute the cutting threshold either using the empirical distribution function for strategy EMPIRICAL, or survival analysis for strategy SURVIVAL;
4. then apply the cutting threshold on all tasks until the budget is exhausted.

For 2), recall that we enroll $\lceil \frac{b}{d} \rceil$ processors, hence up to rounding artefacts, the fraction pb of the whole budget is spent when the fraction pd of the deadline is reached on each processor.

When the average task execution time is large and the observation budget pb is small, it may happen that no task has completed when the observation budget is exhausted. In such a case, we delay the computation of the threshold until having spent $2pb$, and so on if this extended budget is still too small.

There exists an obvious limitation to these first two strategies. First, once a threshold is computed, it is applied until the end. However, in the meantime, new tasks complete and some are interrupted, and we gather more information on the distribution. We should take the new available information into account. We propose to recompute the threshold periodically, each time we have spent

another fraction pb of the budget, by accounting for all the available data. This leads to two new strategies PerEmpirical and PerSurvival, which we assess below.

7.2 Performance evaluation

This section assesses the performance of the *one-size-fits-all* heuristics introduced in Section 7.1. The experimental settings are detailed in Section 7.2.1, and results are presented in Section 7.2.2. All heuristics were implemented in R. The corresponding source code, and all the data, are publicly available in [13].

7.2.1 Experimental methodology

The default settings are as follows. The deadline d can take the values 5, 10, 50, and 100. The parallel platform is composed of $M = 10$ identical processors, each with a unitary cost. As discussed in Section 5.1, recall that a typical configuration enrolls $\lceil \frac{b}{d} \rceil$ processors. We therefore define the budget b as $b = Md$. Then, the budget b is evenly shared among the processors which all execute tasks until the deadline d.

We use different standard probability distribution functions to generate task execution times, namely uniform, exponential, log-normal, half-normal, truncated normal (truncated on $[0, +\infty)$), gamma, inverse-gamma, and Weibull distributions. In addition, multimodal distributions have been advocated to model jobs, file and object sizes [10]. Therefore, we also consider two types of bimodal distributions, either based on truncated normal distributions or on exponential distributions. For all the bimodal distributions, the two modes are equiprobable. For three of the distributions, we consider two different sets of parameters to illustrate different potential behaviors associated to the same type of distribution. These distributions are log-normal, bimodal exponential, and bimodal truncated normal. To enable a direct comparison between all different distributions, we choose their parameters so that all distributions achieve a mean equal to 1. Following the discussion in Section 5.2 about avoiding arbitrarily small task execution times, we add a constant $\delta = 0.05$ to all randomly generated task execution times. Therefore, for all the distributions under study, execution times will always have an average value of 1.05. The detailed parameters of the distributions are presented in Table 1. Due to space limitation, we will sometimes only report here the performance of 6 of these 13 distributions. The performance of the other distributions can be found in the appendix.

Figure 3 presents, for each distribution under study, the theoretical yield achievable as a function of the cutting threshold. In Figure 3, we have ordered the distributions by non-increasing values of their cutting threshold. One can see that different distributions, or the same distribution with different parameters, lead to different shapes of the yield function. For the first distributions

Table 1 Symbol and parameters for the distributions used in the simulations. For all distributions μ is the mean and σ the standard deviation, except for the truncated normal and half-normal distributions where μ and σ are the mean and standard deviation of the original normal distribution.

Symbol	Distribution	Parameters
double_exp(λ_1, λ_2)	Bimodal exponential	$\lambda_1 = \frac{1}{1_{1005}} \approx 0.995, \lambda_2 = \frac{1}{0.995} \approx 1.005$ $\lambda_1 = \frac{1}{0.1} = 10, \lambda_2 = \frac{1}{1.9} \approx 0.526$
double_truncnorm(μ_1, σ_1, μ_2, σ_2)	Bimodal truncated normal	$\mu_1 = 0.5, \sigma_1 \approx 0.534, \mu_2 = 1, \sigma_2 \approx 1.068$ $\mu_1 = 0.01, \sigma_1 \approx 0.178, \mu_2 = 1, \sigma_2 \approx 1.782$
exp(λ)	Exponential	$\lambda = 1$
gamma(k, θ)	Gamma	$k = \frac{1}{3} \approx 0.333, \theta = 3$
hnorm(σ)	Half-normal	$\sigma = \sqrt{\frac{\pi}{2}} \approx 1.253$
invgamma(α, β)	Inverse Gamma	$\alpha = \frac{1}{3} \approx 2.333, \beta = \frac{4}{3} \approx 1.333$
lnorm(μ, σ)	Log-normal	$\mu = 1, \sigma = 0.5$ $\mu = 1, \sigma = 3$
truncnorm(μ, σ)	Truncated normal	$\mu = 0.8, \sigma \approx 0.754$
unif(a, b)	Uniform	$a = 0, b = 2$
weibull(k, λ)	Weibull	$k \approx 0.411, \lambda = \frac{1}{\Gamma(1+\frac{1}{k})} \approx 0.324$

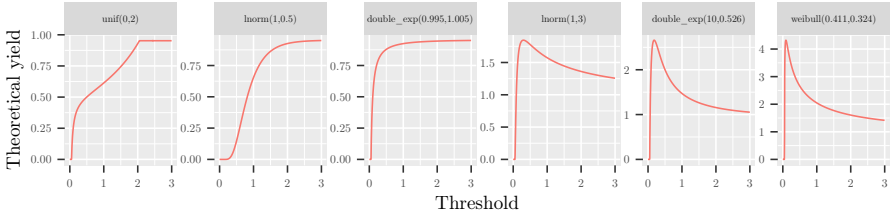

Fig. 3 Theoretical yield when varying cutting threshold for each distribution.

in the figure, tasks should never be interrupted. For the following distributions, tasks should be interrupted, and sometimes quite early. Table 2 reports the optimal cutting threshold for each distribution. This variety of situations makes it challenging to determine a good cutting threshold when the distribution is unknown. In the remainder of this section, in order to ease the comparison of the behavior of the scheduling strategies for the different distributions, all graphs and tables report results with distributions ordered as in Figure 3.

For each simulation setting, we generate 1000 random instances (i.e., sets of task execution times). In addition, we compare the result of the proposed strategies with two reference heuristics. NeverInterrupt is the baseline heuristic which let all tasks run up to completion. Oracle knows in advance the distribution used to generate task execution times and computes the optimal threshold using that knowledge. Oracle is thus an upper bound on the performance of any strategy. Therefore, the closer to Oracle performance, the better the heuristic.

Table 2 Optimal cutting threshold for each distribution

Distribution	Optimal Threshold
unif(0,2)	∞
truncnorm(0.8, 0.754)	∞
lnorm(1,0.5)	∞
hnorm(1.253)	∞
double_truncnorm(0.5,0.534,1,1.068)	∞
double_exp(0.995,1.005)	∞
exp(1)	∞
invgamma(2.333,1.333)	1.842
lnorm(1,3)	0.300
double_truncnorm(0.01,0.178,1,1.782)	0.290
double_exp(10,0.526)	0.180
gamma(0.333,3)	0.110
weibull(0.411,0.324)	0.090

7.2.2 Results

In Figure 4 (and Figure 8 in the appendix), we plot the ratio of the number of tasks successfully executed by each heuristic, over the value achieved by Oracle. Hence, the closer to 1, the better. We plot the performance of each heuristic while varying the percentage p of the budget spent for the observation phase (namely $p = 1\%$, 2.5%, 5%, 10%, 15%, or 20%), and for the four different values of the budget b.

We observe that the performance of the different heuristics is strongly correlated to the shape of the yield functions, as illustrated by Figure 3. In particular, the performance of the heuristics evolves according to our ordering of the distributions. When the optimal threshold is infinite (i.e., for unif(0,2), truncnorm(0.8, 0.75), lnorm(1,0.5), hnorm(1.25), double_truncnorm(0.5,0.5,0.53,1,1.07), double_exp(0.5,1,1.01), and exp(1)), NeverInterrupt has the same performance as Oracle. Also, the performance of the other heuristics increases with p. This is easily explained, since the behavior of the heuristics during the observation phase is, by definition, that of NeverInterrupt. Moreover, the longer the observation phase, the higher the probability that the accumulated data will be of good quality and lead to deriving an efficient threshold.

When the optimal threshold is finite (i.e., for invgamma(2.33,1.33), gamma(0.33,3), lnorm(1,3), double_truncnorm(0.5,0.01,0.18,1,1.78), double_exp(0.5,10,0.53) and weibull(0.41,0.32)), NeverInterrupt performs predictably worse. The lower the optimal threshold, the lower the performance of NeverInterrupt. Also, the larger the budget, the lower the performance of NeverInterrupt, even if the decrease is not always significant. For the other heuristics, the best value for p decreases. This is once again easily explained, because with larger values of p, the observation phase is longer, and thus the budget spent in a suboptimal mode is larger. The graphs are not decreasing from the start because a significant number of tasks must complete

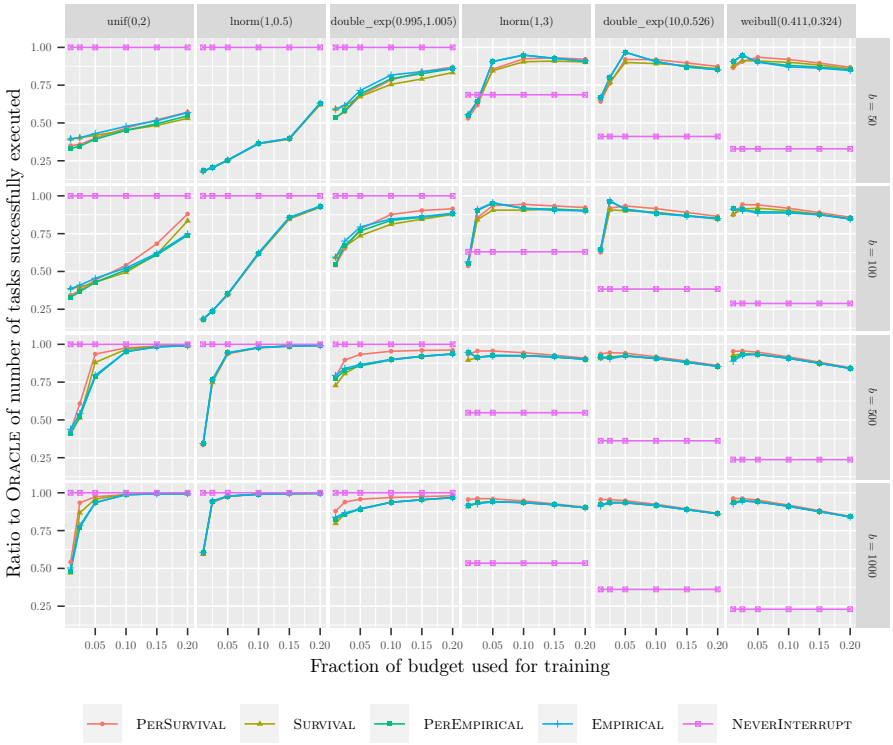

Fig. 4 Ratio to Oracle of number of tasks successfully executed using different heuristics when varying p for each distribution.

to make a decision close to the optimal one, rather than one that is heavily influenced by the random nature of the very few completion times available.

When the budget is large with respect to the average task execution time (e.g., $b = 1000$), many tasks complete before the end of the observation phase and we can infer a relatively precise threshold. Hence, the four heuristics perform globally well. For instance, when $p = 10\%$, all heuristics achieve a performance that is at least 90% that of Oracle, whatever the distribution. For some distributions, some heuristics achieve a performance of 95% of this theoretical optimal. This is true even for distributions that, theoretically, need to be cut early, such as Weibull. Because we have enough budget to obtain a high-quality threshold after the observation period (which costs 10% of the budget), for the rest of the execution time (90% of the budget), we achieve a performance close to the optimal. Therefore the overall results are very good although we do not interrupt tasks during the observation phase.

When the budget is either $b = 500$ or $b = 1000$, PerSurvival achieves the best performance, or a performance equivalent to that of the best of the four heuristics, except for the inverse-gamma distribution. For inverse-gamma,

PerSurvival is sometimes very slightly below PerEmpirical for the same percentage p. However, these two heuristics achieve the same peak performance for that distribution.

On the contrary, when the budget is small with respect to the average task execution time (e.g., $b = 50$), the performance of all heuristics worsens. When $b = 50$ and $p = 10\%$, each of the 10 processors executes tasks for only 0.5 seconds during the observation phase. Hence, the threshold should be computed after very few tasks are completed, if any. It should therefore not be a surprise that the results are then far from optimal. The best performance is achieved either for the distributions which have a small optimal threshold —and then the performance is rather good whatever the value of p— or when the value of p is large —which compensates from the fact that the budget is small. PerSurvival remains the best heuristic when $b = 100$; when $b = 50$ there is no obvious heuristic of choice.

In conclusion, when the budget b is large with respect to the average task execution time, the four basic and periodic heuristics achieve a good performance (at least 90% of the optimal) if we choose carefully the parameter p (e.g., $p = 10\%$). Then, among the four heuristics, PerSurvival achieves the best performance overall and also in most instances. When the budget is small, the performance of the heuristics worsens. The main reason is that for a same value of p, there are no longer enough completed tasks to make a relevant decision for the threshold. When the budget is small, p should be large if tasks should never be interrupted and p should be small if tasks must be interrupted quickly.

Obviously, before running any task, we do not know what the distribution of task execution times will be, what the cutting-threshold will be and, hence, how to adequately chose the value of p. In the next section, we will present the AutoPerSurvival policy which aims at mitigating this problem.

8 Dynamic algorithm

For the heuristics presented in Section 7: when we compute the threshold, we have no idea to what extent the accumulated data is representative of the actual distribution of execution times. Hence, we have no idea of the quality of the resulting threshold. To remedy this, in this section, we propose to automatically infer when to stop observing the distribution and to compute the threshold. As for the experiments, we compare the new heuristic with PerSurvival, the best static heuristic of Section 7, and we observe a dramatic performance improvement in the robustness of the heuristic.

8.1 Automatic inference heuristics

We do not want to compute the threshold before ascertaining that the data acquired on the distribution of task execution times is "good enough". However, we do not want to spend the whole budget trying to acquire information. Hence, we decide to rely on two parameters fixed a priori: a percentage p_{max}

of the overall budget and a precision ϵ. The precision ϵ will guarantee that we have a good enough approximation of the data distribution because the mean value and standard deviation of the empirical distribution function have converged (up to the precision ϵ). In addition, the percentage p_{max} will be a large value guaranteeing that in extreme cases, we will take a decision eventually, before exhausting the budget. We compute the threshold as soon as one of the two following conditions is met: either observing convergence of the empirical distribution function, or having spent a fraction p_{max} b of the overall budget. In practice, each time a task completes, we recompute the mean value and standard deviation of the distribution. If both new values have a relative difference less than ϵ from previous ones, we assume the approximation of the distribution to have converged.

Once we have computed a cutting threshold, say after having spent a budget qb, we recompute it periodically each time we have spent $\max\{0.01, q\}b$ of the budget. We add the max for the cases where the budget is very large and the convergence very fast, in order to keep the number of decisions constant (as stated in Section 4). Obviously the new strategy can be implemented for both the empirical distribution function and the survival analysis. However, because of the superiority of the survival analysis (shown in Section 7.2.2), we implement it only for survival analysis, leading to the new strategy AutoPerSurvival.

8.2 Performance evaluation

In this section, we assess the performance of the automatic inference heuristics introduced previously. The experimental methodology is the same as in Section 7.2.1. We compare first AutoPerSurvival with PerSurvival heuristic. And then, we show the stability of AutoPerSurvival while varying different parameters. Finally, we summarize all simulation results.

8.2.1 AutoPerSurvival vs. PerSurvival

In Figure 5 (and Figure 9 in the appendix), we compare the performance of AutoPerSurvival for different values of p_{max} (namely p_{max}= 10%, 20%, 30%, 40%, or 50%) when varying ϵ (namely, ϵ= 0.0010, 0.0025, 0.0050, 0.0100, 0.0250, 0.0500, and 0.1000). We added the performance of PerSurvival using different values for p as a reference. In each figure we plot the ratio of the number of tasks successfully executed by each heuristic, over the value achieved by Oracle.

In all graphs, we observe that the performance of AutoPerSurvival is influenced by the interplay of the two parameters ϵ and p_{max}. When the value of ϵ is very small, we need a very large (in expectation) number of launched tasks to meet the ϵ criteria. This, in turn, will require to spend a large amount of budget for the observation phase. If ϵ is sufficiently small, on most instances this requirement will exceed the upper limit set by p_{max} on the budget spent during the observation phase. Hence, if ϵ is sufficiently small, the behavior

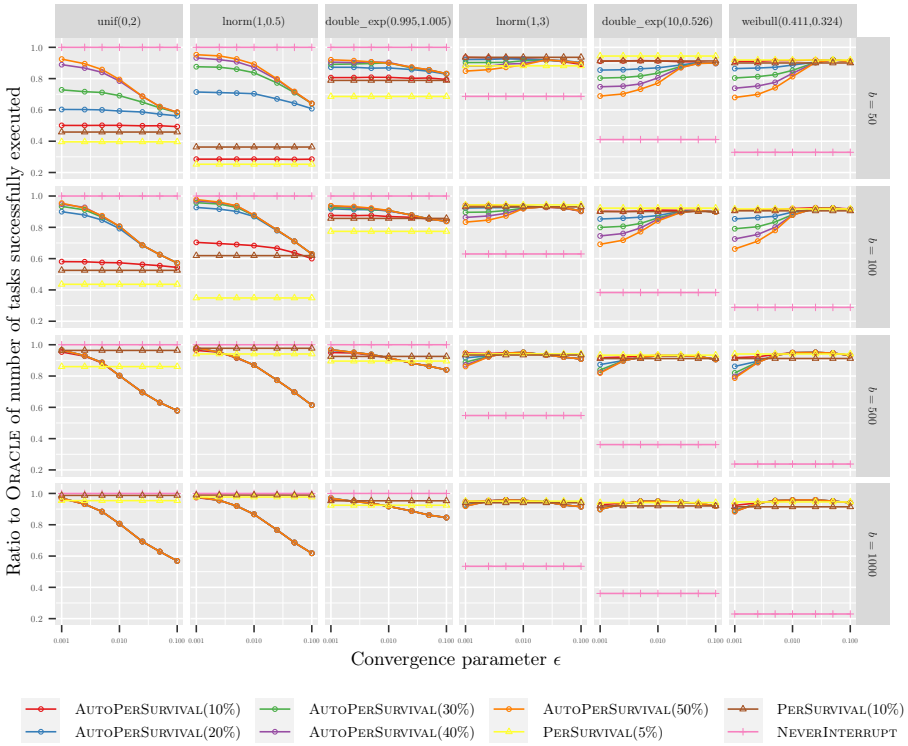

Fig. 5 Ratio to Oracle of number of tasks successfully executed using either AutoPerSurvival (for different values of p_{max} and when varying ϵ) or PerSurvival (with different values of p).

of AutoPerSurvival is only dictated by the value of p_{max}. For instance, when $b = 50$, this is the case for most of the distributions when $\epsilon \leq 0.0025$. However, when the value of ϵ gets larger, convergence is reached sooner. Then an approximation of the data distribution deemed "good enough" (with respect to ϵ) is obtained before spending a share p_{max} of the budget. In that case, p_{max} does not play any role, and only ϵ has an influence on the observation period, and thus on the performance. For instance, when $b = 100$, this is the case for weibull(0.411,0.324) when $\epsilon \geq 0.0250$. However, for the uniform distribution when $b = 100$, note that $p_{max} = 10\%$ still plays a role when $\epsilon = 0.1000$ which explains why AutoPerSurvival(10%, 0.1000) has a performance lower than that of the other AutoPerSurvival variants.

Furthermore, we see that the evolution of the performance depends upon the optimal cutting threshold. When the optimal cutting threshold is infinite, the smaller ϵ, the better the performance. Indeed, during the observation period, the optimal NeverInterrupt strategy is implemented, and, later on, the cutting-threshold strategy is applied. This is particularly true when we

have enough time before convergence (large values of p_{max} and of b). In such a case, there is no performance penalty in having a large observation period during which tasks are not interrupted. In contrast, for distributions with a short optimal cutting threshold, small ϵ values (and longer observation periods) waste more budget without interrupting tasks, and the performance decreases when ϵ decreases.

Globally, when the budget and deadline are large enough, AutoPerSurvival (when $\epsilon \leq 0.0050$) performs similarly to PerSurvival, and they both have a good performance (around 90% in nearly all cases). In this case, all p_{max} values perform equally well. However, when the budget and deadline decrease, we already know that PerSurvival performs worse, and we observe that the performance of AutoPerSurvival is strongly correlated to the value of p_{max} and ϵ. Among the parameters tested, AutoPerSurvival(40%, 0.005) is a good choice, because it can successfully execute more than 77% of the tasks of the optimal heuristic Oracle, regardless of the distribution and the budget (deadline) values. In other words, using AutoPerSurvival(40%, 0.005) will always lead to good results, contrarily to all *one-size-fits-all* heuristics.

8.2.2 Stability of performance while varying μ, σ, and M

In Figure 6 (and Figure 10 in the appendix) we assess the performance of the different heuristics under a log-normal distribution of task execution times when $b = 50$ (and $b = 1000$ for Figure 10 in the appendix) for different values of the average task execution time (μ), of the standard deviation (σ,) and of the number of processors in the platform (M). We use a log-normal distribution because it has been advocated to model file sizes [10], and thus task costs can also be assumed to follow this distribution. For the heuristics, we choose the parameters which achieved the best performance in the previous simulations: AutoPerSurvival is used with the parameters $p_{max} = 40\%$ and $\epsilon = 0.005$. For the four *one-size-fits-all* strategies, we use the same value to define the observation phase: $p = 10\%$.

When the budget is big enough ($b = 1000$), all heuristics perform similarly and close to the optimal in all configurations. AutoPerSurvival may perform slightly better than the four other heuristics in most of the cases but the differences are minimal.

Figure 6 presents the more interesting case of a small budget $b = 50$ with respect to the average task execution time. The first row of subgraphs show the influence of the average task execution time, μ, on the performance of heuristics. Remark that for $b = 50$, $p = 10\%$, and $M = 10$, the observation phase for *one-size-fits-all* heuristics only lasts for 0.5 second, during which one expects that very few processors will be able to complete a task. This gets even more true when μ increases, and explains that the performance of the heuristics is decreasing. Nevertheless, the performance of AutoPerSurvival decreases more slowly than that of the other heuristics. For instance, when $\mu = 3$, the four *one-size-fits-all* heuristics already achieve a rather bad performance while

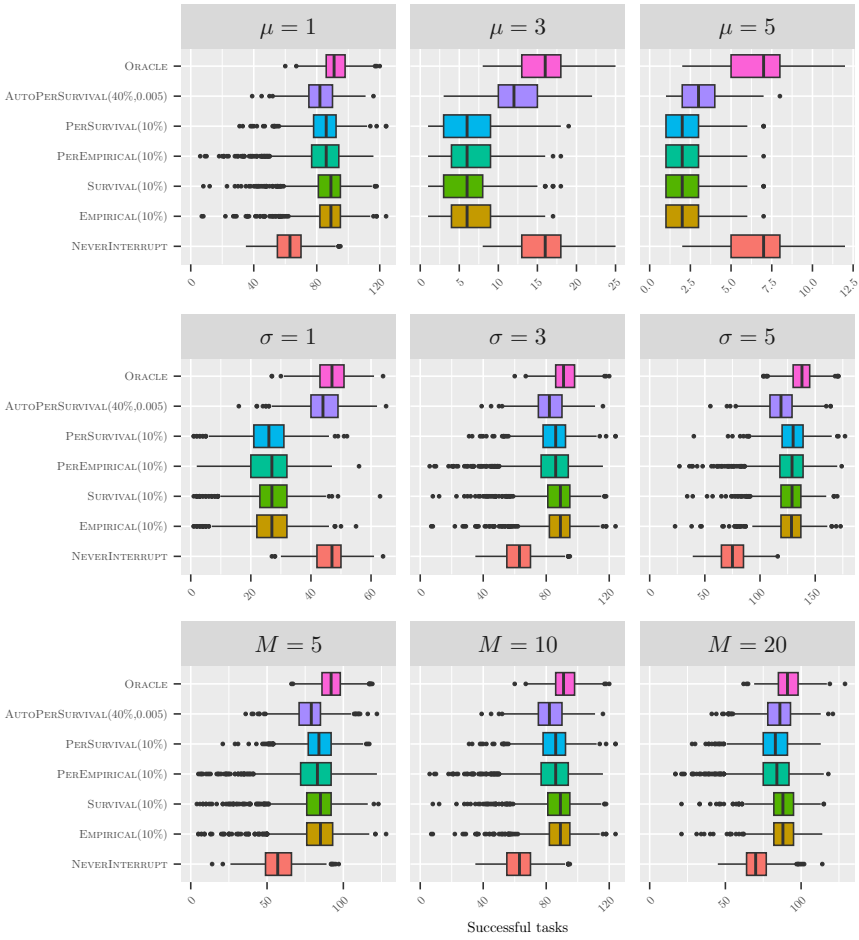

Fig. 6 Number of successfully executed tasks for the different heuristics with a budget $b = 50$ when task execution times follow a log-normal distribution. Unless otherwise specified, the expectation is $\mu = 1$, the standard deviation is $\sigma = 3$, and the number of processors is $M = 10$.

AutoPerSurvival remains quite close to the optimal. The fact that AutoPerSurvival automatically adapts the length of its observation phase to the quality of the information that it gathers (here mainly the number of tasks that complete), enables it to achieve a graceful degradation of performance.

The second row of subgraphs shows the impact of the standard deviation σ. When σ varies, the optimal cutting-threshold varies. This is illustrated by the performance of NeverInterrupt which decreases when σ increases, showing that the optimal threshold also decreases. All heuristics have similar performance when $\sigma = 3$ and $\sigma = 5$. However, only AutoPerSurvival achieves near optimal performance when $\sigma = 1$.

Table 3 Ratio to ORACLE of number of tasks successfully executed for each heuristic and each distribution with $\mu = 1$, $b = 50$ and $d = 5$

	NEVERINTERRUPT	AUTOPERSURVIVAL(40%,0.005)	PERSURVIVAL (10%)	SURVIVAL (10%)	PEREMPIRICAL (10%)	EMPIRICAL (10%)
unif(0,2)	1.0000±0.0085	0.8402±0.0162	0.4659±0.0125	0.4522±0.0150	0.4513±0.0159	0.4775±0.0153
truncnorm(0.8,0.754)	1.0000±0.0089	0.8432±0.0167	0.4089±0.0134	0.4002±0.0151	0.3981±0.0157	0.4288±0.0151
lnorm(1,0.5)	1.0000±0.0073	0.9074±0.0126	0.3620±0.0256	0.3646±0.0257	0.3633±0.0257	0.3667±0.0258
lnorm(1.253)	1.0000±0.0095	0.8675±0.0149	0.6046±0.0139	0.5701±0.0183	0.5974±0.0186	0.6146±0.0182
double_truncnorm(0.5,0.534,1,1.068)	1.0000±0.0097	0.8647±0.0156	0.4989±0.0146	0.4818±0.0172	0.4940±0.0176	0.5088±0.0165
double_exp(0.995,1.005)	1.0000±0.0133	0.9066±0.0155	0.7846±0.0162	0.7552±0.0224	0.7927±0.0213	0.8172±0.0209
exp(1)	1.0000±0.0133	0.9047±0.0161	0.7973±0.0179	0.7456±0.0242	0.7916±0.0236	0.8010±0.0227
invgamma(2.333,1.333)	0.9858±0.0121	0.9521±0.0134	0.4729±0.0159	0.4738±0.0166	0.4790±0.0160	0.4788±0.0158
lnorm(1,3)	0.6865±0.0114	0.8927±0.0117	0.9238±0.0125	0.9048±0.0171	0.9484±0.0139	0.9493±0.0137
double_truncnorm(0.01,0.178,1,1.782)	0.5233±0.0072	0.7837±0.0116	0.9190±0.0105	0.8831±0.0162	0.9386±0.0112	0.9420±0.0105
double_exp(10,0.526)	0.4107±0.0067	0.7663±0.0109	0.9190±0.0088	0.8925±0.0131	0.9068±0.0097	0.9057±0.0097
gamma(0.333,3)	0.8513±0.0054	0.9552±0.0062	0.9658±0.0053	0.9592±0.0062	0.9603±0.0053	0.9610±0.0055
weibull(0.411,0.324)	0.3291±0.0063	0.7760±0.0104	0.9198±0.0079	0.9014±0.0103	0.8802±0.0105	0.8711±0.0108

The third row of subgraphs show that varying the number of processors has no significant impact on the performance of the heuristics: all scenarios achieve near optimal performance. Overall, AUTOPERSURVIVAL(40%, 0.005) is a very robust heuristic, which overcomes the other heuristics in all settings, and which, in the most adverse scenarios, exhibits a graceful degradation of performance with respect to the theoretical optimal.

8.2.3 Summary

To summarize our findings, we present two tables showing the number of tasks successfully executed by each heuristic for each distribution expressed as a fraction of the optimal performance (of ORACLE). We present results for a large budget (Table 4 in the appendix, $b = 1000$ and $d = 100$) and a small one (Table 3, $b = 50$ and $d = 5$) with respect to the average task execution time ($\mu = 1$). We use the same parameters as previously: $\epsilon = 0.005$, $p_{max} = 40\%$, and $p = 10\%$.

With large values of budget and deadline, all heuristics perform well. Indeed, with the chosen parameters, all heuristics achieve at least 88% of the performance of the optimal. Among the *one-size-fits-all* heuristics, PERSURVIVAL performs best and is the most robust, but the difference between these heuristics is not always significant. On average, the performance of AUTOPERSURVIVAL and PERSURVIVAL are pretty similar.

Table 3 presents the result when budget and deadline are small. In this case all *one-size-fits-all* heuristics achieve very low performance, below 40% for each of them (for the lnorm(1,0.5) distribution for example). On the contrary, AUTOPERSURVIVAL always achieves good to very good performance: its worse case is 77% of the optimal. Once again, this shows the great robustness of AUTOPERSURVIVAL(40%, 0.005).

9 Conclusion

In this work, we have studied the problem of maximizing the number of stochastic independent tasks successfully executed on a parallel platform under deadline and budget constraints. When task execution times obey a probability distribution that is known before execution, previous results showed that long-running tasks must be interrupted at some optimal cutting threshold τ, and

provided techniques to determine its value. Some probability distributions call for a very short threshold τ while others have a large or infinite one. The main challenge in this study is that the probability distribution of task execution times is unknown to the scheduler. We designed a set of scheduling heuristics to estimate the cutting threshold τ, some of which making use of the Kaplan-Meier estimator. We also assessed different decision mechanisms to compute and refine the threshold as the execution progresses. On the practical side, extensive simulations show that our best heuristic AutoPerSurvival(40%, 0.005) achieves good performance for a wide spectrum of probability distributions and parameter sets. In the worst scenario, it can execute 77% of tasks that an omniscient oracle (knowing the distribution) would be able to complete.

Future work will be dedicated to considering heterogeneous processors, still under the assumption that the distribution of task execution times is unknown on the different processors. Indeed, some cloud providers provide different categories of processors with different computer power and nominal cost, and execution times are not directly proportional to cost nor power. This heterogeneity will dramatically complicate the selection of a good processor subset, and the estimation of the cutting threshold for each of them.

Acknowledgments

We would like to thank the reviewers for their comments and suggestions, which greatly helped improve the final version of the paper. The work of Yiqin Gao was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

Conflict of interest

There is no conflict of interest for this work.

References

[1] Aalen O, Borgan O, Gjessing H (2008) Survival and event history analysis: a process point of view. Springer Science & Business Media

[2] Amirijoo M, Hansson J, Son SH (2006) Specification and management of qos in real-time databases supporting imprecise computations. IEEE Trans Computers 55(3):304–319

[3] Buttazzo G (2012) Handling overload conditions in real-time systems. In: Babamir SM (ed) Real-Time Systems, Architecture, Scheduling, and Application. InTech, Rijeka, chap 7, https://doi.org/10.5772/37265, URL https://doi.org/10.5772/37265

[4] Cai Z, Li X, Ruiz R, et al (2017) A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds. Future Generation Computer Systems 71:57 – 72. https://doi.org/https://doi.org/10.1016/j.future.2017.01.020, URL http://www.sciencedirect.com/science/article/pii/S0167739X17300870

[5] Canon LC, Kong Win Chang A, Robert Y, et al (2018) Scheduling independent stochastic tasks under deadline and budget constraints. In: SBAC-PAD. IEEE

[6] Canon LC, Kong Win Chang A, Robert Y, et al (2019) Scheduling independent stochastic tasks under deadline and budget constraints. Int J High Performance Computing Applications 34(2):246–264

[7] Casanova H, Gallet M, Vivien F (2010) Non-clairvoyant scheduling of multiple bag-of-tasks applications. In: Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference, pp 168–179, https://doi.org/10.1007/978-3-642-15277-1_17, URL https://doi.org/10.1007/978-3-642-15277-1_17

[8] Chambless LE, Diao G (2006) Estimation of time-dependent area under the roc curve for long-term risk prediction. Statistics in Medicine 25(20):3474–3486. https://doi.org/10.1002/sim.2299, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.2299, https://arxiv.org/abs/https://onlinelibrary.wiley.com/doi/pdf/10.1002/sim.2299

[9] Chung JY, Liu JWS, Lin KJ (1990) Scheduling periodic jobs that allow imprecise results. IEEE Trans Computers 39(9):1156–1174

[10] Feitelson D (2014) Workload modeling for computer systems performance evaluation. Version 103 pp 1–607

[11] Feng W, Liu JWS (1993) An extended imprecise computation model for time-constrained speech processing and generation. In: Proc. IEEE Workshop on Real-Time Applications, pp 76–80, https://doi.org/10.1109/RTA.1993.263112

[12] Ferguson TS (2008) Optimal stopping and applications. UCLA Press

[13] Gao Y (2020) Resource-constrained scheduling of stochastic tasks with unknown probability distribution. https://doi.org/10.6084/m9.figshare.13187135.v1, URL https://figshare.com/articles/software/Resource-Constrained_Scheduling_of_Stochastic_Tasks_With_Unknown_Probability_Distribution/13187135/1

[14] Gao Y, Canon L, Robert Y, et al (2019) Scheduling independent stochastic tasks on heterogeneous cloud platforms. In: 2019 IEEE International

Conference on Cluster Computing (CLUSTER), pp 1–11

[15] Grekioti A, Shakhlevich NV (2014) Scheduling bag-of-tasks applications to optimize computation time and cost. In: PPAM, LNCS, vol 8385. Springer

[16] Guyot P, Ades AE, Ouwens MJ, et al (2012) Enhanced secondary analysis of survival data: reconstructing the data from published kaplan-meier survival curves. BMC Medical Research Methodology 12(1):9. https://doi.org/10.1186/1471-2288-12-9, URL https://doi.org/10.1186/1471-2288-12-9

[17] Hassan H, Simó J, Crespo A (2001) Flexible real-time mobile robotic architecture based on behavioural models. Engineering Applications of Artificial Intelligence 14(5):685 – 702. https://doi.org/https://doi.org/10.1016/S0952-1976(01)00029-X, URL http://www.sciencedirect.com/science/article/pii/S095219760100029X

[18] Im S, Kulkarni J, Munagala K (2017) Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. J ACM 65(1). https://doi.org/10.1145/3136754, URL https://doi.org/10.1145/3136754

[19] Jumel F, Simonot-Lion F (2003) Management of anytime tasks in real time applications. In: XIV Workshop on Supervising and Diagnostics of Machining Systems, Karpacz/Pologne, URL https://hal.inria.fr/inria-00099612

[20] Kaplan EL, Meier P (1958) Nonparametric estimation from incomplete observations. Journal of the American Statistical Association 53(282):457–481. https://doi.org/10.1080/01621459.1958.10501452

[21] Kobayashi H, Yamasaki N (2004) Rt-frontier: a real-time operating system for practical imprecise computation. In: 10th IEEE Real-Time and Embedded Tech. Appl. Symp., pp 255–264, https://doi.org/10.1109/RTTAS.2004.1317271

[22] Li K (2019) Non-clairvoyant scheduling of independent parallel tasks on single and multiple multicore processors. Journal of Parallel and Distributed Computing 133:210 – 220. https://doi.org/https://doi.org/10.1016/j.jpdc.2018.06.001, URL http://www.sciencedirect.com/science/article/pii/S0743731518303988

[23] Liu JWS, Lin KJ, Shih WK, et al (1991) Algorithms for scheduling imprecise computations. In: van Tilborg AM, Koob GM (eds) Foundations of Real-Time Computing: Scheduling and Resource Management. Springer, pp 203–249, https://doi.org/10.1007/978-1-4615-3956-8_8, URL https:

//doi.org/10.1007/978-1-4615-3956-8_8

[24] Lopez O (2012) A generalization of the kaplan–meier estimator for analyzing bivariate mortality under right-censoring and left-truncation with applications in model-checking for survival copula models. Insurance: Mathematics and Economics 51(3):505 – 516. https://doi.org/https://doi.org/10.1016/j.insmatheco.2012.07.009, URL http://www.sciencedirect.com/science/article/pii/S0167668712000881

[25] Mao M, Li J, Humphrey M (2010) Cloud auto-scaling with deadline and budget constraints. In: 2010 11th IEEE/ACM International Conference on Grid Computing. IEEE, pp 41–48, https://doi.org/10.1109/GRID.2010.5697966

[26] Meng J, Chakradhar S, Raghunathan A (2009) Best-effort parallel execution framework for recognition and mining applications. In: IPDPS. IEEE, https://doi.org/10.1109/IPDPS.2009.5160991

[27] Oprescu AM, Kielmann T (2010) Bag-of-tasks scheduling under budget constraints. In: 2010 IEEE Second International Conference on Cloud Computing Technology and Science, pp 351–359, https://doi.org/10.1109/CloudCom.2010.32

[28] Oprescu AM, Kielmann T, Leahu H (2011) Budget estimation and control for bag-of-tasks scheduling in clouds. Parallel Processing Letters 21(02):219–243. https://doi.org/10.1142/S0129626411000175, URL https://www.worldscientific.com/doi/abs/10.1142/S0129626411000175, https://arxiv.org/abs/https://www.worldscientific.com/doi/pdf/10.1142/S0129626411000175

[29] Oprescu AM, Kielmann T, Leahu H (2012) Stochastic tail-phase optimization for bag-of-tasks execution in clouds. In: Fifth Int. Conf.s on Utility and Cloud Computing. IEEE, pp 204–208, https://doi.org/10.1109/UCC.2012.23

[30] Samoladas I, Angelis L, Stamelos I (2010) Survival analysis on the duration of open source projects. Information and Software Technology 52(9):902–922

[31] Scanniello G (2011) Source code survival with the kaplan meier. In: 2011 27th IEEE International Conference on Software Maintenance (ICSM), pp 524–527

[32] Scanniello G (2011) Source code survival with the Kaplan Meier estimator. In: 27th IEEE International Conference on Software Maintenance (ICSM). IEEE, pp 524–527

[33] Singh P, Khan B, Vidyarthi A, et al (2019) Energy-aware online non-clairvoyant scheduling using speed scaling with arbitrary power function. Applied Sciences 9(7). https://doi.org/10.3390/app9071467, URL https://www.mdpi.com/2076-3417/9/7/1467

[34] Thai L, Varghese B, Barker A (2018) A survey and taxonomy of resource optimisation for executing bag-of-task applications on public clouds. Future Generation Computer Systems 82:1 – 11. https://doi.org/https://doi.org/10.1016/j.future.2017.11.038, URL http://www.sciencedirect.com/science/article/pii/S0167739X17305071

[35] der Vaart V (1998) Asymptotic Statistics. Cambridge University Press

[36] Vecchiola C, Calheiros RN, Karunamoorthy D, et al (2012) Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka. Future Generation Computer Systems 28(1):58 – 65. https://doi.org/https://doi.org/10.1016/j.future.2011.05.008, URL http://www.sciencedirect.com/science/article/pii/S0167739X11000896

[37] Xie J, Liu C (2005) Adjusted kaplan–meier estimator and log-rank test with inverse probability of treatment weighting for survival data. Statistics in Medicine 24(20):3089–3110. https://doi.org/10.1002/sim.2174, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.2174, https://arxiv.org/abs/https://onlinelibrary.wiley.com/doi/pdf/10.1002/sim.2174

Appendix

9.1 Proof of Theorem 2

We start by recalling some classical results for a unimodal exponential distribution of rate λ, whose probability density function and cumulative distribution function are respectively denoted by $f(x)$ and $F(x)$ (here we do not add any constant or, in other words, $\delta = 0$):

- Probability that the random variable is no greater than l: $P(X \leq l) = F(l) = 1 - e^{-\lambda l}$.
- Average value of a random variable which is no greater than l: $\mathbb{E}(X \leq l) = \int_{x=0}^{l} xf(x)dx = -le^{-\lambda l} + \frac{1}{\lambda}\left(1 - e^{-\lambda l}\right)$.
- Average value of the time spent executing a task whose execution time is defined by an exponential distribution of parameter λ when the execution is cut at time l if the task has not completed by that time:

$$\left(-le^{-\lambda l} + \frac{1}{\lambda}\left(1 - e^{-\lambda l}\right)\right) + l(1 - F(l)) = \frac{1}{\lambda}\left(1 - e^{-\lambda l}\right).$$

When the cutting threshold l is smaller than or equal to the constant time δ, the yield is obviously null: $\mathcal{Y}(l) = 0$. Therefore, we only consider the yield for cutting thresholds greater than or equal to δ. To ease the writing, let $y(l) = \mathcal{Y}(l + \delta)$. Using Equation 2, we derive the expression of the yield when the cutting threshold is set at $l + \delta$:

$$\begin{aligned} y(l) &= \mathcal{Y}(l + \delta) \\ &= \frac{p\left(1 - e^{-\lambda l}\right) + (1 - p)\left(1 - e^{-\mu l}\right)}{\delta + p\left(\frac{1}{\lambda}\left(1 - e^{-\lambda l}\right)\right) + (1 - p)\left(\frac{1}{\mu}\left(1 - e^{-\mu l}\right)\right)} \\ &= \frac{p\left(1 - e^{-\lambda l}\right) + (1 - p)\left(1 - e^{-\mu l}\right)}{\delta + \frac{p}{\lambda}\left(1 - e^{-\lambda l}\right) + \frac{(1-p)}{\mu}\left(1 - e^{-\mu l}\right)} \end{aligned}$$

In order to identify the maximum of the function $y(l)$, we differentiate it:

$$\begin{aligned} y'(l) = &\frac{\left(\lambda p e^{-\lambda l} + \mu(1-p)e^{-\mu l}\right)\left(\delta + \frac{p}{\lambda}\left(1 - e^{-\lambda l}\right) + \frac{(1-p)}{\mu}\left(1 - e^{-\mu l}\right)\right)}{\left(\delta + \frac{p}{\lambda}\left(1 - e^{-\lambda l}\right) + \frac{(1-p)}{\mu}\left(1 - e^{-\mu l}\right)\right)^2} \\ &- \frac{\left(p\left(1 - e^{-\lambda l}\right) + (1-p)\left(1 - e^{-\mu l}\right)\right)\left(p e^{-\lambda l} + (1-p)e^{-\mu l}\right)}{\left(\delta + \frac{p}{\lambda}\left(1 - e^{-\lambda l}\right) + \frac{(1-p)}{\mu}\left(1 - e^{-\mu l}\right)\right)^2} \end{aligned}$$

Because we are only interested by the sign of $y'(l)$, we focus on its numerator, that we denote by $g(l)$:

$$\begin{aligned}
g(l) &= \left(\lambda p e^{-\lambda l} + \mu(1-p)e^{-\mu l}\right)\left(\delta + \tfrac{p}{\lambda}\left(1-e^{-\lambda l}\right) + \tfrac{(1-p)}{\mu}\left(1-e^{-\mu l}\right)\right) \\
&\quad - \left(p\left(1-e^{-\lambda l}\right) + (1-p)\left(1-e^{-\mu l}\right)\right)\left(pe^{-\lambda l} + (1-p)e^{-\mu l}\right) \\
&= \delta\left(\lambda p e^{-\lambda l} + \mu(1-p)e^{-\mu l}\right) \\
&\quad + p(1-p)\left(\left(\tfrac{\lambda}{\mu} - 1\right)e^{-\lambda l}\left(1-e^{-\mu l}\right) + \left(\tfrac{\mu}{\lambda} - 1\right)e^{-\mu l}\left(1-e^{-\lambda l}\right)\right) \\
&= e^{-\lambda l}\left(\delta\left(\lambda p + \mu(1-p)e^{-(\mu-\lambda)l}\right)\right. \\
&\qquad \left. + p(1-p)\tfrac{\mu-\lambda}{\lambda\mu}\left(-\lambda\left(1-e^{-\mu l}\right) + \mu e^{-\mu l}\left(e^{\lambda l} - 1\right)\right)\right).
\end{aligned}$$

We focus on the expression

$$\begin{aligned}
h(l) &= -\lambda\left(1 - e^{-\mu l}\right) + \mu e^{-\mu l}\left(e^{\lambda l} - 1\right) \\
&= -\lambda + \lambda e^{-\mu l} + \mu e^{-(\mu-\lambda)l} - \mu e^{-\mu l}.
\end{aligned}$$

We differentiate h (with respect to l):

$$\begin{aligned}
h'(l) &= -\lambda\mu e^{-\mu l} - \mu(\mu-\lambda)e^{-(\mu-\lambda)l} + \mu^2 e^{-\mu l} \\
&= \mu e^{-\mu l}(\mu-\lambda)\left(1 - e^{\lambda l}\right).
\end{aligned}$$

Because $\mu - \lambda$, λ and l are all nonnegative, $1 - e^{\lambda l} \leq 0$ and $h'(l)$ is nonpositive. Therefore, $h(l)$ is a non-increasing function.

Clearly, $g(l)$ and $k(l) = g(l)e^{\lambda l}$ have the same sign. From what precedes, $k(l) = g(l)e^{\lambda l}$ is the sum of two non-increasing functions, the functions $\delta\left(\lambda p + \mu(1-p)e^{-(\mu-\lambda)l}\right)$ and $p(1-p)\frac{\mu-\lambda}{\lambda\mu}h(l)$. The maximum of $k(l)$ is thus reached for $l = 0$ and its minimum is reached when l tends toward $+\infty$:

$$\begin{aligned}
k(0) &= \delta\left(\lambda p + \mu(1-p)\right); \\
\lim_{l\to+\infty} k(l) &= \delta\lambda p + p(1-p)\frac{\mu-\lambda}{\lambda\mu}\left(-\lambda\right) \\
&= \delta\lambda p - p(1-p)\frac{\mu-\lambda}{\mu} \\
&= p\left(\delta\lambda - (1-p)\frac{\mu-\lambda}{\mu}\right).
\end{aligned}$$

To conclude, we have several cases to consider:

1. $\delta = 0$ (arbitrarily small execution times are allowed). Then, $k(0) = 0$ and $\lim_{l\to+\infty} k(l) \leq 0$. We have two subcases to consider:
   (a) $p(1-p)(\mu-\lambda) = 0$. In this case, we have a *monomodal* exponential distribution. Then, $y'(l)$ is null, $\mathcal{Y}(l) = y(l)$ is constant and equal to λ. In other words, in this case the yield is optimal whatever the value chosen for the threshold.
   (b) $p(1-p)(\mu-\lambda) \neq 0$. In this case, $\mathcal{Y}(l) = y(l)$ is a decreasing function and its maximum is achieved when $l = 0$, and the optimum yield is

then $\lim_{l\to 0} \mathcal{Y}(l)$. To compute this limit we use the equivalent to e^{-x} in 0 which is $1-x$. We obtain:

$$\begin{aligned}\mathcal{Y}(l) &= \frac{p\left(1-e^{-\lambda l}\right)+(1-p)\left(1-e^{-\mu l}\right)}{c+\frac{p}{\lambda}\left(1-e^{-\lambda l}\right)+\frac{(1-p)}{\mu}\left(1-e^{-\mu l}\right)} \\ &\approx \frac{p\left(\lambda l\right)+(1-p)\left(\mu l\right)}{\frac{p}{\lambda}\lambda l+\frac{(1-p)}{\mu}\mu l} \\ &= p\lambda+(1-p)\mu.\end{aligned}$$

Remark: this counter-intuitive result means that the shortest the threshold, the better. It means that, in practice, the scheduler should stop each task as soon as it is started. Obviously, this is not achievable in practice. This peculiar property is a consequence of allowing execution times to be arbitrarily small, as the remainder of this case study will illustrate.

2. $\delta > 0$. Once again, we have two subcases to consider:
   - $p(\delta\lambda-(1-p)\frac{\mu-\lambda}{\mu}) \geq 0$. Then $k(l)$, and thus $g(l)$ and $y'(l)$ are non-negative for all values of l. y and $\mathcal{Y}$ are thus increasing and we should never abort the execution of a running task (threshold $= +\infty$). The optimum yield in this case is then:

$$\lim_{l\to+\infty} \mathcal{Y}(l) = \frac{1}{\delta+\frac{p}{\lambda}+\frac{(1-p)}{\mu}}.$$

   - $p(\delta\lambda-(1-p)\frac{\mu-\lambda}{\mu}) < 0$ which can be rewritten $\delta < (1-p)\frac{\mu-\lambda}{\lambda\mu}$. In this subcase, we have $y'(l)$ which is a decreasing function, with $y'(0) > 0$ and $\lim_{l\to+\infty} k(l) < 0$ which implies that $y'(l)$ and $\mathcal{Y}'(l)$ are negative when l is sufficiently large. Therefore, $\mathcal{Y}(l+\delta)$ is first increasing and then decreasing, and has a unique maximum. This maximum is achieved for l satisfying $k(l) = 0$, which could only be solved numerically.

9.2 Additional graphs and statistics

We report here the theoretical threshold and the performance for the distributions that were not illustrated in the core of the article. We also report the performance of heuristics when the budget is large with respect to the average task execution time ($b = 1000$).

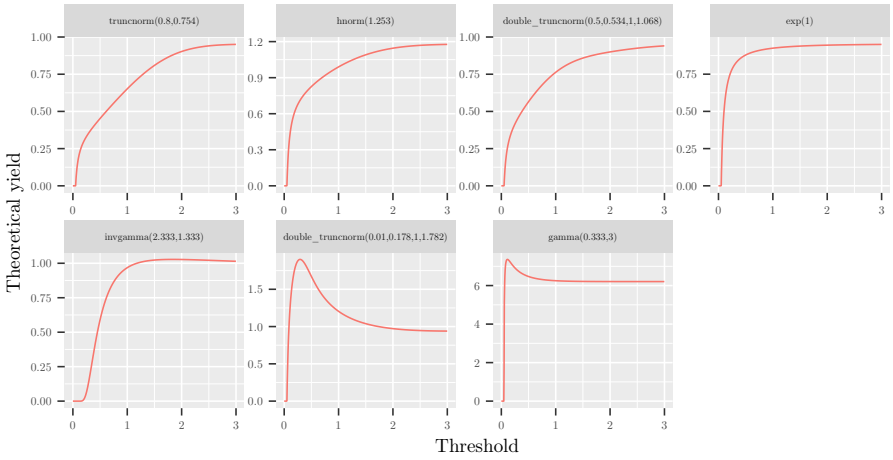

Fig. 7 Theoretical yield when varying cutting threshold for each distribution.

Fig. 8 Ratio to Oracle of number of tasks successfully executed using different heuristics when varying p for each distribution.

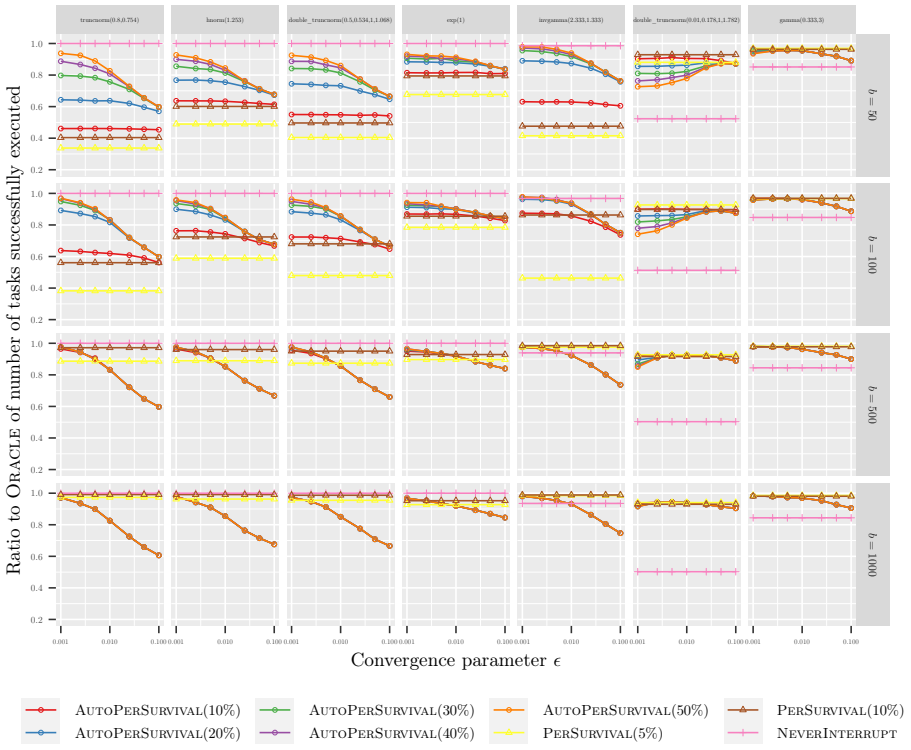

Fig. 9 Ratio to ORACLE of number of tasks successfully executed using either AUTOPERSURVIVAL (for different values of p_{max} and when varying ϵ) or PERSURVIVAL (with different values of p).

Table 4 Ratio to ORACLE of number of tasks successfully executed for each heuristic and each distribution with $\mu = 1$, $b = 1000$ and $d = 100$.

	NEVERINTERRUPT	AUTOPERSURVIVAL(40%,0.005)	PERSURVIVAL(10%)	SURVIVAL(10%)	PEREMPIRICAL(10%)	EMPIRICAL(10%)
unif(0,2)	1.000±0.0017	0.8839±0.0120	0.9880±0.0020	0.9887±0.0020	0.9873±0.0026	0.9881±0.0027
truncnorm(0.8,0.754)	1.000±0.0017	0.8989±0.0118	0.9911±0.0019	0.9914±0.0020	0.9913±0.0019	0.9914±0.0020
lnorm(1,0.5)	1.000±0.0015	0.9200±0.0108	0.9894±0.0017	0.9909±0.0017	0.9903±0.0017	0.9909±0.0017
lnorm(1.253)	1.000±0.0020	0.9100±0.0093	0.9920±0.0021	0.9920±0.0029	0.9903±0.0034	0.9903±0.0035
double_truncnorm(0.5,0.534,1,1.068)	1.000±0.0020	0.9123±0.0090	0.9881±0.0023	0.9873±0.0036	0.9860±0.0040	0.9859±0.0039
double_exp(0.995,1.005)	1.000±0.0030	0.9378±0.0057	0.9690±0.0038	0.9379±0.0089	0.9370±0.0089	0.9367±0.0089
exp(1)	1.000±0.0030	0.9349±0.0061	0.9685±0.0039	0.9352±0.0091	0.9362±0.0091	0.9367±0.0091
invgamma(2.333,1.333)	0.9350±0.0034	0.9535±0.0080	0.9876±0.0027	0.9871±0.0029	0.9897±0.0027	0.9881±0.0028
lnorm(1,3)	0.5345±0.0038	0.9590±0.0044	0.9464±0.0029	0.9352±0.0044	0.9344±0.0043	0.9351±0.0044
double_truncnorm(0.01,0.178,1,1.782)	0.5023±0.0014	0.9446±0.0048	0.9338±0.0030	0.9229±0.0046	0.9259±0.0044	0.9259±0.0043
double_exp(10,0.526)	0.3610±0.0014	0.9499±0.0038	0.9236±0.0024	0.9151±0.0036	0.9169±0.0035	0.9162±0.0034
gamma(0.333,3)	0.8440±0.0012	0.9732±0.0051	0.9810±0.0011	0.9801±0.0011	0.9803±0.0011	0.9799±0.0012
weibull(0.411,0.324)	0.2296±0.0018	0.9538±0.0046	0.9183±0.0022	0.9108±0.0027	0.9109±0.0027	0.9109±0.0027

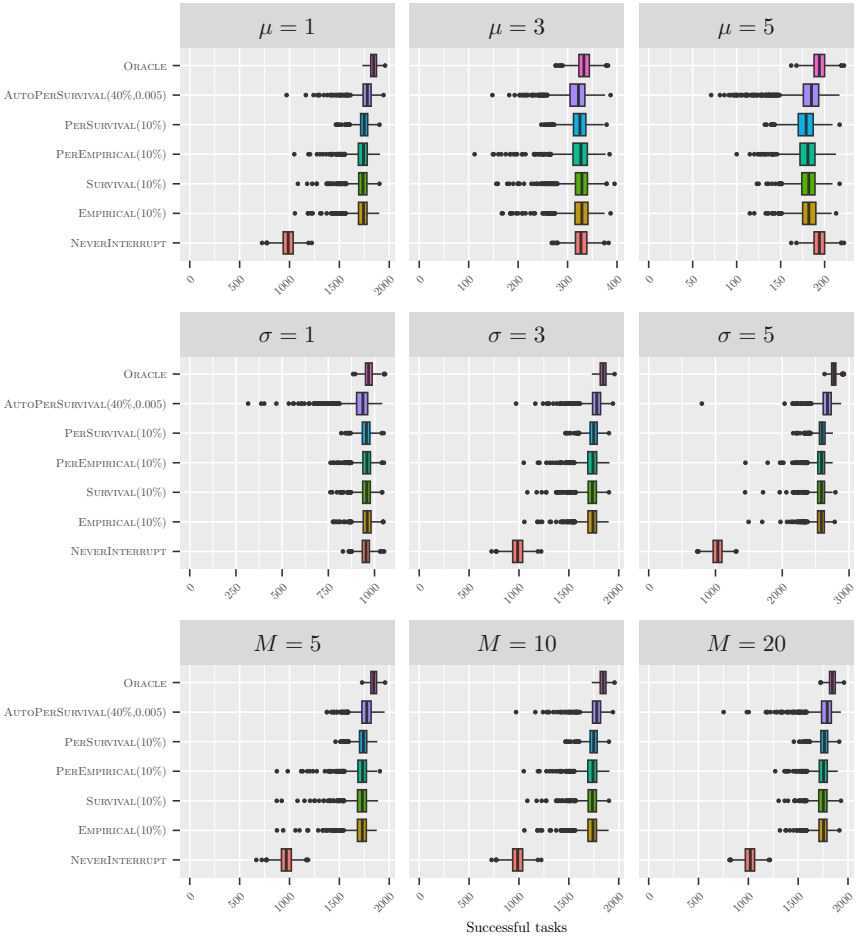

Fig. 10 Number of successfully executed tasks for the different heuristics with a budget $b = 1000$ when task execution times follow a log-normal distribution. Unless otherwise specified, the expectation is $\mu = 1$, the standard deviation is $\sigma = 3$, and the number of processors is $M = 10$.