Resource-Constrained Scheduling of Stochastic Tasks With Unknown Probability Distribution

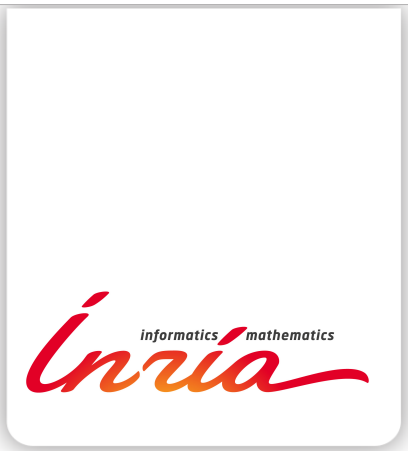
Yiqin Gao, Yves Robert, Frédéric Vivien

Resource-Constrained Scheduling of Stochastic Tasks With Unknown Probability Distribution

Yiqin Gao, Yves Robert, Frédéric Vivien

RESEARCH REPORT
N° 9373
November 2020
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--9373--FR+ENG



Resource-Constrained Scheduling of Stochastic Tasks With Unknown Probability Distribution

Yiqin Gao*, Yves Robert*†, Frédéric Vivien*

Project-Team ROMA

Research Report n° 9373 — November 2020 — 40 pages

Abstract: This work introduces scheduling strategies to maximize the expected number of independent tasks that can be executed on a cloud platform within a given budget and under a deadline constraint. 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 works presents several strategies 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 strategies is assessed through an extensive set of simulations with various budget and deadline values, and ranging over 14 probability distributions.

Key-words: stochastic tasks, independent tasks, budget, deadline, scheduling, Kaplan-Meier estimator.

* LIP, École Normale Supérieure de Lyon, CNRS & Inria, France
† University of Tennessee Knoxville, USA

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Ordonnancement de tâches stochastiques à distribution de probabilité inconnue

Résumé : Ce travail présente des stratégies d'ordonnancement permettant de maximiser le nombre attendu de tâches indépendantes pouvant être exécutées sur une plateforme de type *cloud* avec un budget donné et une contrainte de date limite. Le temps d'exécution des tâches est inconnu, on sait seulement qu'ils obéissent à une distribution de probabilité (inconnue). L'ordonnanceur peut décider à tout moment d'interrompre l'exécution d'une tâche (longue) en cours d'exécution et d'en lancer une nouvelle, mais le budget déjà utilisé pour la tâche interrompue est perdu. Le seuil d'interruption d'une tâche peut être recalculé au fur et à mesure que l'exécution progresse globalement. Ce travail présente plusieurs stratégies pour déterminer un bon seuil d'interruption, et pour décider quand le ré-évaluer. Nous utilisons l'estimateur de Kaplan-Meier pour prendre en compte les tâches en cours d'exécution au moment où la décision est prise. L'efficacité de nos stratégies est évaluée via un vaste ensemble de simulations, avec diverses valeurs de budget et de date limite, et portant sur 14 distributions de probabilité.

Mots-clés : tâches indépendantes, temps d'exécution stochastiques, ordonnancement, budget, date limite, estimateur de Kaplan-Meier.

1 Introduction

This paper focuses on the design of scheduling strategies to maximize the expected number of successfully executed tasks on a cloud platform composed of identical Virtual Machines (VMs, or processors[1]). The tasks are independent and their execution times are unknown. The only information known by the scheduler is that the task 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 VM, 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 VMs to enroll. Then, the scheduler needs to acquire some information about task execution times by letting several tasks run until completion on each VM. At some point, the scheduler synthesizes the information acquired so far and will decide 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. The main is to determine when and how to compute a first cutting threshold τ (with the possibility that $\tau = +\infty$, meaning that all tasks are allowed to run until completion). Again, there is a trade-off. Deciding for the threshold early can lead to an imprecise estimation because it is based on little information, but this would avoid to consume a significant fraction of the deadline and of the budget before interrupting any task. 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. Altogether, these are several complicated trade-offs to achieve. The key is to be able to compute a good threshold without bias, and this paper 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

[1]Throughout the text, we use both terms VM and processor indifferently.

will be interrupted, and some tasks will never be launched. This problem is very closely related to *imprecise computations* [16, 35, 3], particularly in the context of real-time computations. In imprecise computations, it is not necessary for all tasks to be completely processed to obtain a meaningful result. Most often, tasks in imprecise computations 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 not the time or 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 [41], robotic systems [28], speech processing [20], and [33] also cites 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 [13] 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 4.1 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 [13] 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 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 [32] to account for long-running tasks when estimating the threshold τ.
- We introduce several methods for deciding when to recompute the

threshold

- We report a comprehensive set of simulation results that compare the heuristics for various budget and deadline values, using up to 14 different probability distributions.

The rest of the paper is organized as follows. Section 2 surveys related work. We detail the framework and objective in Section 3. In Section 4, we provide background on prior strategies for interrupting tasks when the distribution is known (Section 4.1), together with a set of new results for Exponential distributions (Section 4.2). We provide new scheduling heuristics when the distribution is unknown in Sections 5 and 6: Section 5 is devoted to methods for computing the cutting threshold accurately, while Section 6 focuses on when to recompute it. We compare the heuristics in Section 7, assessing their performance for an extensive set of simulation parameters. Finally, we provide concluding remarks and directions for future work in Section 8.

2 Related work

This work falls under the scope of cloud computing since it targets the execution of independent tasks on a cloud platform under deadline and budget constraints. We overview cloud computing in Section 2.1 and bags of tasks in Section 2.2 Furthermore, task execution times obey a probability distribution which is unknown before execution, which is closely related to non-clairvoyant scheduling, which we survey in Section 2.3. Then in Section 2.4, we survey the closely related model of imprecise computations. Finally, we survey in Section 2.5 the Kaplan-Meier estimator, which we use in our heuristics to estimate task execution times.

2.1 Cloud computing

There exists a huge literature on cloud computing, see the geveral surveys [49, 48, 6, 5]. Resource provisioning and scheduling are key steps to the efficient execution of workflows on cloud platforms. Singh and Chana published a survey devoted solely to cloud resource provisioning [48], that is, the decision of which resources should be enrolled to perform the computations. Resource scheduling decides which computations should be processed by each of the enrolled resources and in which order they should be performed. The multi-objective scheduling problem that consists in meeting deadlines and either respecting a budget or minimizing the cost (or energy) has been extensively studied for deterministic workflows [10, 18, 4, 53, 39, 2, 8, 38, 23], but has received much less attention in a stochastic context. Indeed, most of the studies assume a *clairvoyant* setting: the resource provisioning and task

scheduling mechanisms know in advance, and accurately, the execution time of all tasks. A handful of additional studies also consider that tasks may fail [36, 45]. Among these articles, Poola et al. [45] differ as they assume that tasks have uncertain execution times. However, they assume they know these execution times with a rather good accuracy (the standard deviation of the uncertainty is 10% of the expected execution time). They are thus dealing with uncertainties rather than a true non-clairvoyant setting. The work in [11] targets stochastic tasks but is limited to taking static decisions (no task interruption). Some works are limited to a particular type of application like MapReduce [29, 51, 27]. For instance, Tian and Chen [51] consider MapReduce programs and can either minimize the financial cost while matching a deadline or minimize the execution time while enforcing a given budget. Our task model applies to compute-bound tasks because we do not account for communication times and instead assume that they are negligible in front of computation times. However, we refine the classical deterministic model by adding stochasticity to task execution times.

2.2 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. There exists a survey about resource optimization for bag of tasks applications [50], but they did not pay attention to the non-clairvoyant case. Several works devoted to bag-of-tasks processing explicitly target cloud computing [25, 9, 43, 14]. Most of them [25, 9, 14] consider the classical clairvoyant model, in which we know the exact execution time or its distribution, or the uncertain model, in which we know its range or its standard deviation, while [43] targets a non-clairvoyant setting (see Section 2.3). Vecchiola et al. [52] 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; their contribution is therefore hard to assess. Mao et al. [40] 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.

2.3 Non-clairvoyant scheduling

The work surveyed so far assume a fully or semi clairvoyant set of task execution times, which is not always true in a realistic scenario. 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, Sungjin et al. [30] and Pawan et al. [47] both worked on online algorithms. They assume that the size of arriving tasks is not known before completing them. In [30], a unified model is designed for several different scheduling problems, while [47] aims at minimizing flow-time and energy. In the work of Li [34], task execution times are unknown, and the objective is to minimize the makespan while using one or several multicore processors. A group of authors [43, 42, 44] 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 [42, 44]. This work differs from ours as these authors do not consider deadlines. For instance, in [43], the objective is to try to complete all tasks, possibly using replication on faster VMs, 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.

2.4 Imprecise computations and anytime tasks

Our task model assumes that some tasks may not be executed. This model is very closely related to *imprecise computations* [16, 35, 3]. Furthermore, this task model also corresponds to the overload case of [7] where jobs can be *skipped* or *aborted*. Another related model, is that of *anytime tasks* [31] 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 [21].

2.5 Kaplan-Meier estimator (KMS)

We will show in Section 5.2 that our problem of estimation of task execution times is equivalent to a version of survival analysis called *survival analysis with right-censored data*. The solution to this problem is a famous statistical estimator: the Kaplan and Meier estimator [32]. Nowadays, survival analysis with the Kaplan-Meier estimator is widely used in biostatistics [26, 54, 15],

Table 1: Summary of notations.

b	budget
d	deadline
M	number of VMs in the platform
$\mathcal{D}$	probability distribution of task execution times
μ, σ	mean, standard deviation of $\mathcal{D}$

and in a variety of other domains such as engineering [46], economics [37], etc. A more comprehensive presentation about survival and event history analysis can be found in [1].

Altogether, 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 [12, 22] had the same setting under homogeneous [12] or heterogeneous [22] 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.

3 Problem definition

We consider a cloud platform composed of M identical virtual machines (VMs) or processors. The execution time of a task on a VM obeys an unknown probability distribution $\mathcal{D}$. Without loss of generality, we assume it costs one budget unit to execute a task for one second on any VM, and we have a total budget b and an overall deadline d. When considering the asymptotic behavior of policies, we assume that budget b and deadline d grow toward infinity, and obey the equation $b = Md$. Main notations are summarized in Table 1. We assume executions to be non-preemptive: if the execution of a task is interrupted, 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. This is equal to the expected success rate per second, as we spend one budget unit per second.

All our 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 at each execution of a new task.

4 Optimal strategies for known distributions

In Section 4.1, we recall previous results for known distributions, namely an asymptotically optimal policy for discrete distributions and its extension to continuous distributions [13, 12]. Then, in Section 4.2, 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.

4.1 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 [13, 12] 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 4.1.1), and then we move to continuous distributions (Section 4.1.2).

Note that in addition to decide for 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.

4.1.1 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:

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

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 ($M = 1$) [13, 12].

Theorem 1. *Let $\tau_{opt} = \arg\max_{\tau \in \{w_1, \ldots, w_k\}} \mathcal{Y}(\tau)$. 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 $\left\lceil \frac{b}{d} \right\rceil$ processors and execute on each of them the fixed-threshold strategy of threshold τ_{opt}.

4.1.2 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):

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

The optimal threshold is then, like previously:

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

4.2 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 will study the case where the distribution is either unimodal or bimodal. We will 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. (Obviously, 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 web supplementary material.

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.

5 Threshold estimation for unknown distributions

We have seen in Section 4 that when the distribution of task execution times is known, the optimal policy is a fixed-threshold strategy that interrupts tasks, and that 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 single option is to execute some tasks and record their execution times. We will consider the problem of deciding how many tasks to execute in Section 6. For the sake of the argument, let us assume 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. The question is then: how do we estimate the distribution of task execution times based on this data?

Figure 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 which would deliver 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. Section 5.1 details 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. 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 5.2, we show that our problem is exactly the statistical medical problem known as *survival analysis with right-censored data*, even if the concepts and wordings are quite different. We also show how to use its classical solution, the Kaplan-Meier estimator, to solve our problem more accurately.

5.1 The empirical distribution function

The naive approach only considers the execution times of completed tasks and uses the associated *empirical distribution function* [17], 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.

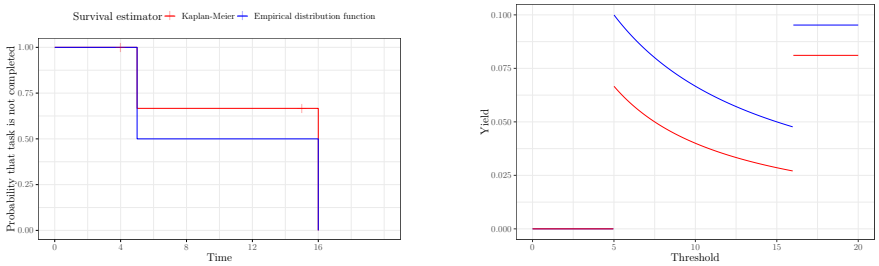


Figure 2: Probability of survival (left) and yield (right) for the toy example of Figure 1 when using the empirical distribution function (blue) or the Kaplan-Meier estimator (red).

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. This drawback induces a bias by ignoring long-running tasks.

5.2 Survival analysis and 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 ours, 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 tool to solve survival analysis with right-censored data, that is the Kaplan-Meier estimator [32, 1]. We refer the interested reader to [1] for a thorough overview of survival analysis.

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:

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

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

6 Taking decisions

In Section 5, we have shown how we can use data from the execution of tasks 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 tradeoff before launching any task. Then, we present a policy that tries to automatically infer that tradeoff.

6.1 *One-size-fits-all* policies

The simplest strategies will try to guess, without interrupting any task, the "right" tradeoff. Consider a strategy that spends 10% of the overall budget

running tasks up to completion before computing the optimal threshold: it can still hope to achieve a 90% overall efficiency. Indeed, it can achieve such a good performance just by applying the optimal policy for the actual distribution of task execution times during the remaining 90% of the budget. As this looks promising, this is the basis of our first two strategies:

1. we pick a priori a percentage p;
2. we run tasks on processors until we have spent the fraction $p \times b$ of the overall budget;
3. we compute the cutting threshold either using the empirical distribution function for strategy EMPIRICAL, or survival analysis for strategy SURVIVAL;
4. we 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 task average 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 to after having spent $2pb$, and so on if this extended budget is also too small.

There are two obvious limitations 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 do that periodically, each time we have spent another fraction pb of the budget, by recomputing the threshold considering all the available data. This gives us two new strategies PEREMPIRICAL and PERSURVIVAL.

The second limitation is due to the fact that when we compute the threshold, we have no idea how much the accumulated data is representative of the actual distribution of execution times. Therefore, we have no idea of the quality of the threshold that we compute. To remedy this, in the next section, we propose to automatically infer when to stop observing the distribution and to compute the threshold.

6.2 Automatic inference

We do not want to compute the threshold before ascertaining that the data we have 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 eventually take a decision before running out of the budget. We will 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 3). 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), we implement it only for survival analysis, leading to the new strategy AutoPerSurvival.

7 Experiments

This section assesses the performance of the different strategies introduced in the previous sections. The experimental settings are detailed in Section 7.1, and results are presented in Section 7.2. All strategies were implemented in R. The corresponding source code, and all the data, are publicly available in [24].

7.1 Experimental methodology

The default settings are as follows. The deadline d can take the values 5, 10, 50, and 100. The cloud platform is composed of $M = 10$ identical VMs, each with a unitary cost. As stated in Section 3, the budget b is defined as $b = Md$. Then, the budget b is evenly shared among the VMs which all execute tasks until the deadline d. As discussed in Section 4.1, recall that a typical configuration enrolls $\lceil \frac{b}{d} \rceil$ VMs.

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 [19]. 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 four 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 the gamma distribution, the log-normal, the bimodal exponential, and the 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. The detailed parameters of the distributions are presented in Table 2. Due to space limitation, we will sometimes only report here the performance of 6 of these 14 distributions. The performance of the other distributions can be found in the web supplementary material.

Following the discussion in Section 4.2 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.

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's performance, the better the heuristic.

7.2 Results

We present the experimental results in two steps. First in Section 7.2.1, we present results for the *one-size-fits-all* heuristics: Empirical, Survival, PerEmpirical and PerSurvival, for all the distributions. PerSurvival turns out to be the best of the 4 heuristics in almost all studied cases. This is why, in Section 7.2.2, we compare PerSurvival and AutoPerSurvival.

Before assessing the performance of the heuristics, we consider the distributions under study. Figure 3 presents, for each distribution, 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 in the figure, tasks should never be interrupted. For the following distributions, tasks should be interrupted, and sometimes quite early. Table 3 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

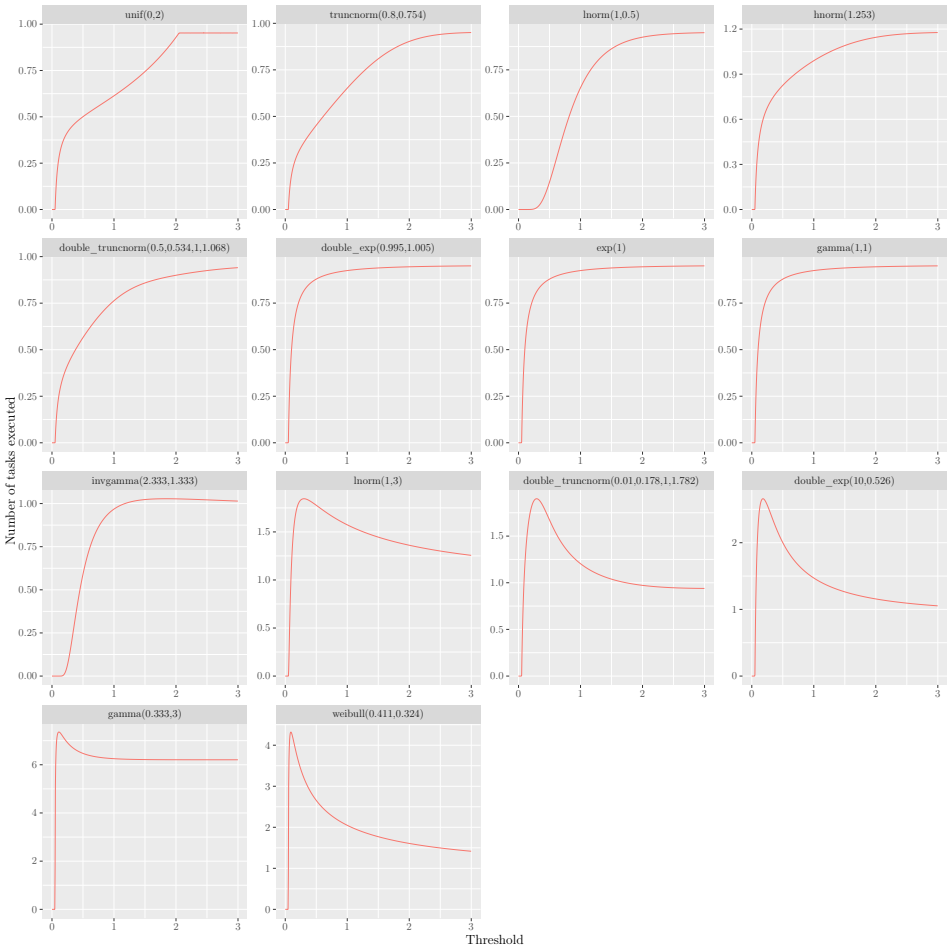


Figure 3: Theoretical yield when varying cutting threshold for each distribution.

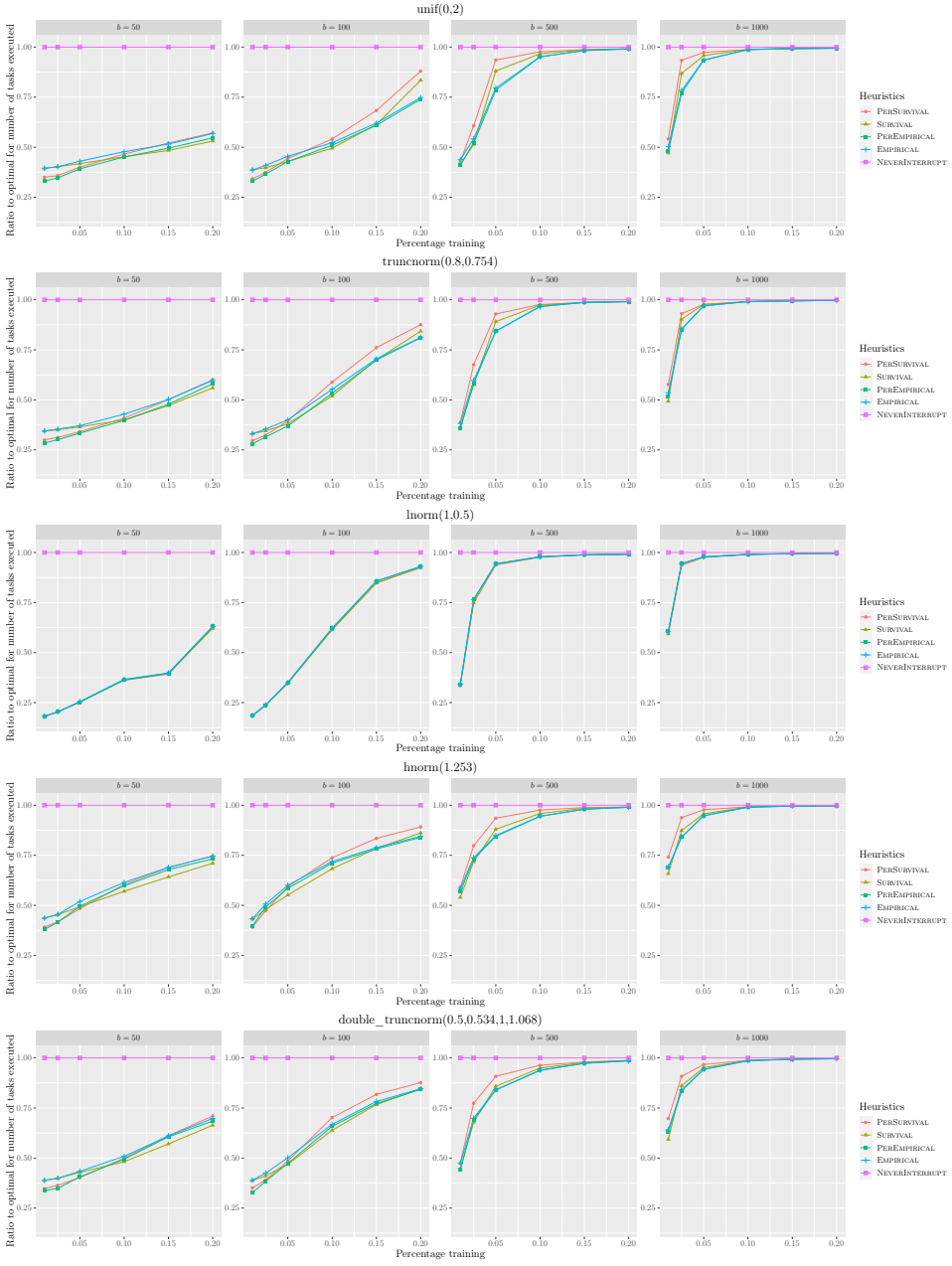


Figure 4: Ratio to Oracle of number of tasks successfully executed using different heuristics when varying $p$for each distribution (1/3).

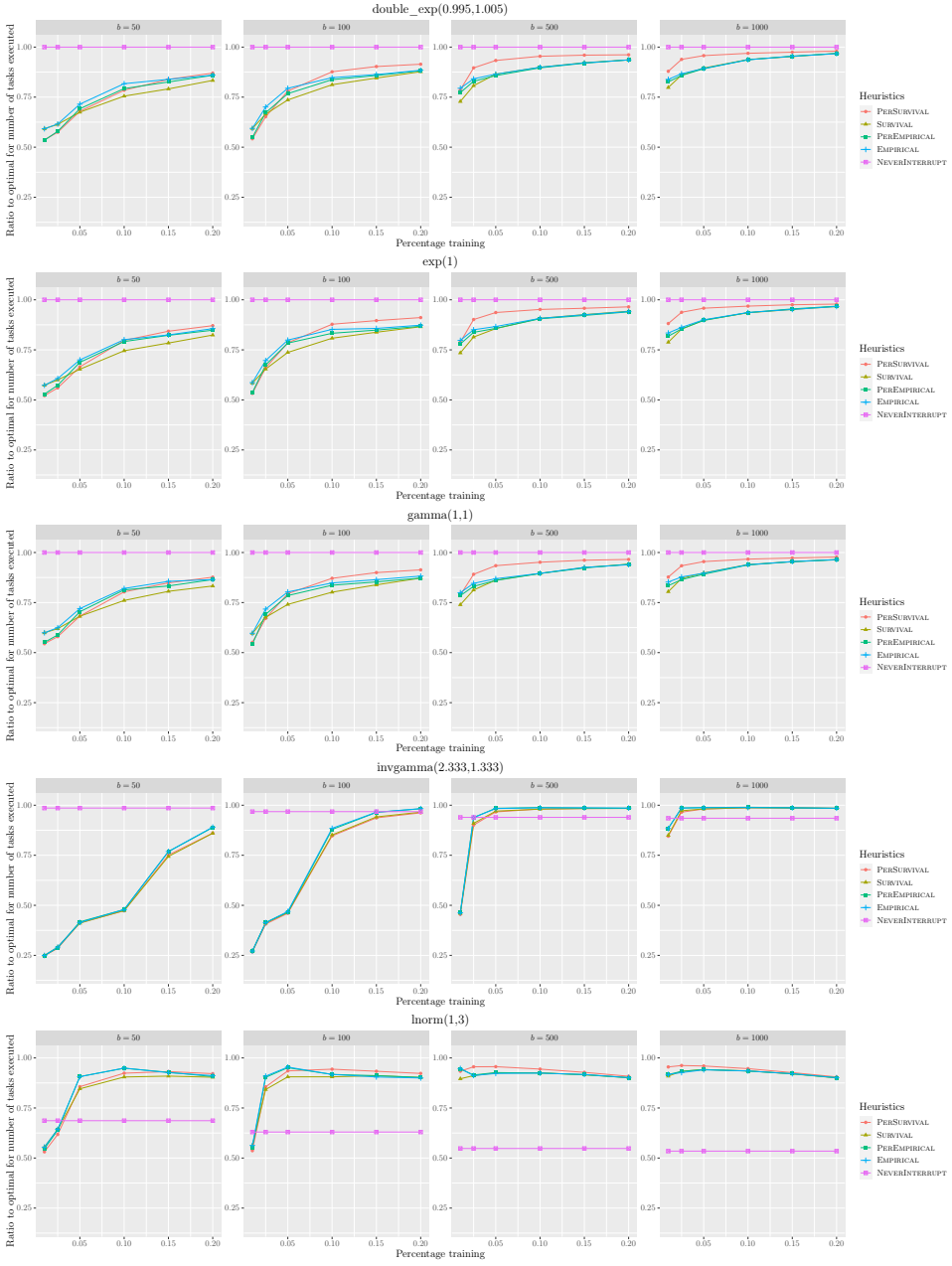


Figure 5: Ratio to Oracle of number of tasks successfully executed using different heuristics when varying $p$for each distribution (2/3).

Figure 6: Ratio to Oracle of number of tasks successfully executed using different heuristics when varying $p$for each distribution (3/3).

Table 2: 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	$\lambda_1 = \frac{1}{1.005} \approx 0.995, \lambda_2 = \frac{1}{0.995} \approx 1.005$
		$\lambda_1 = \frac{1}{0.1} = 10, \lambda_2 = \frac{1}{1.9} \approx 0.526$
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	$k = 1, \theta = 1$
		$k = \frac{1}{3} \approx 0.333, \theta = 3$
hnorm(σ)	Half-normal	$\sigma = \sqrt{\frac{\pi}{2}} \approx 1.253$
invgamma(α, β)	Inverse Gamma	$\alpha = \frac{7}{3} \approx 2.333, \beta = \frac{4}{3} \approx 1.333$
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	$k \approx 0.411, \lambda = \frac{1}{\Gamma(1+\frac{1}{k})} \approx 0.324$

to ease the comparison of the behaviors of the different strategies for the different distributions, all graphs and tables report results with distributions ordered as in Figure 3.

7.2.1 *One-size-fits-all* heuristics

In Figures 4, 5, and 6, 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 evolve 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), exp(1) and gamma(1,1)), 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),

Table 3: 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)	∞
gamma(1,1)	∞
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

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 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 deduce 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 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 with respect to 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 average task execution time will be, what the cutting-threshold will be and hence, how to adequately chose the value of p. The AUTOPERSURVIVAL policy aims at alleviating this problem.

7.2.2 AUTOPERSURVIVAL vs. PERSURVIVAL

In Figures 7, 8 and 9, 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 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

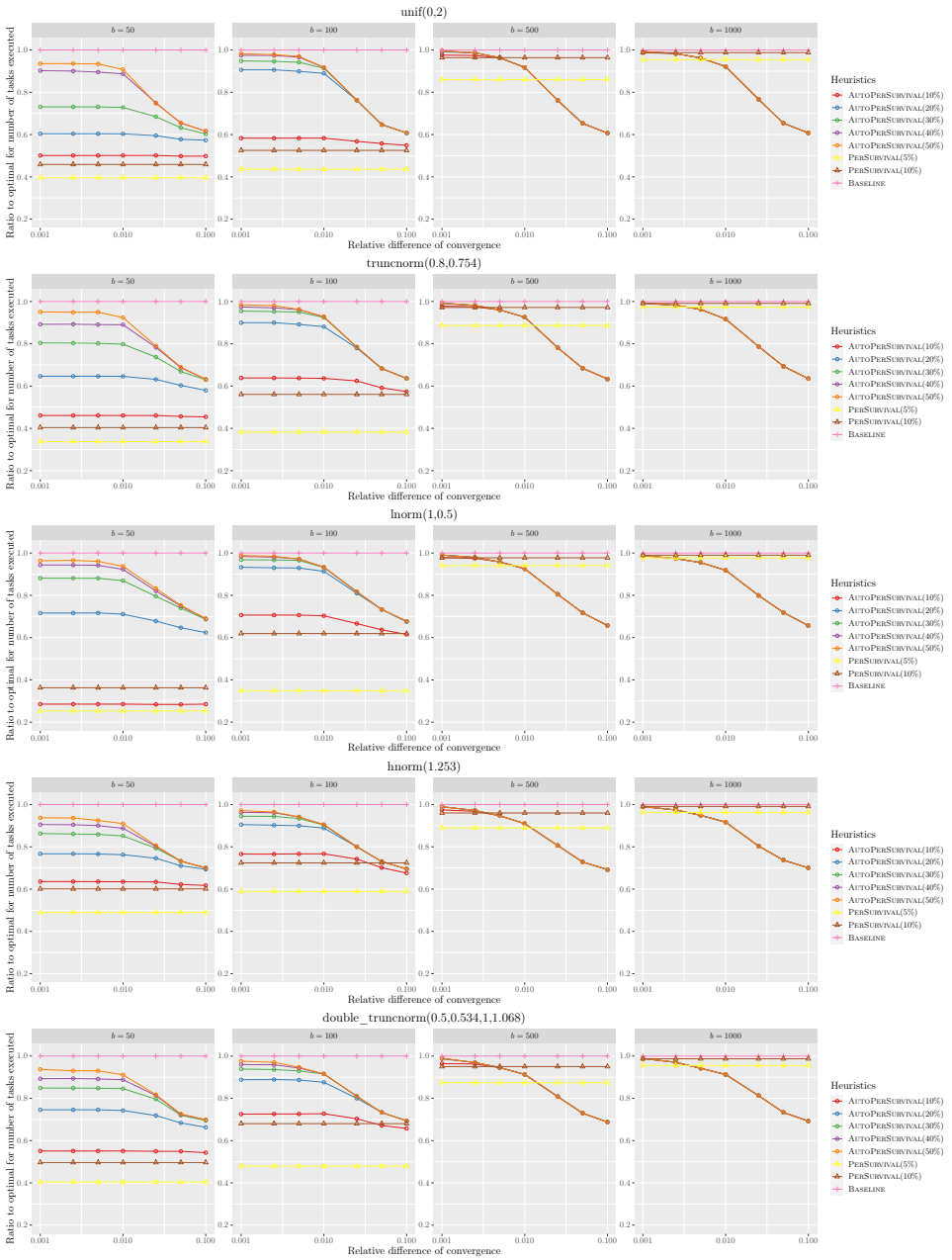


Figure 7: Number of successfully executed tasks for each distribution using AutoPerSurvival (different p_{max} when varying ϵ) and PerSurvival (different p) (1/3).

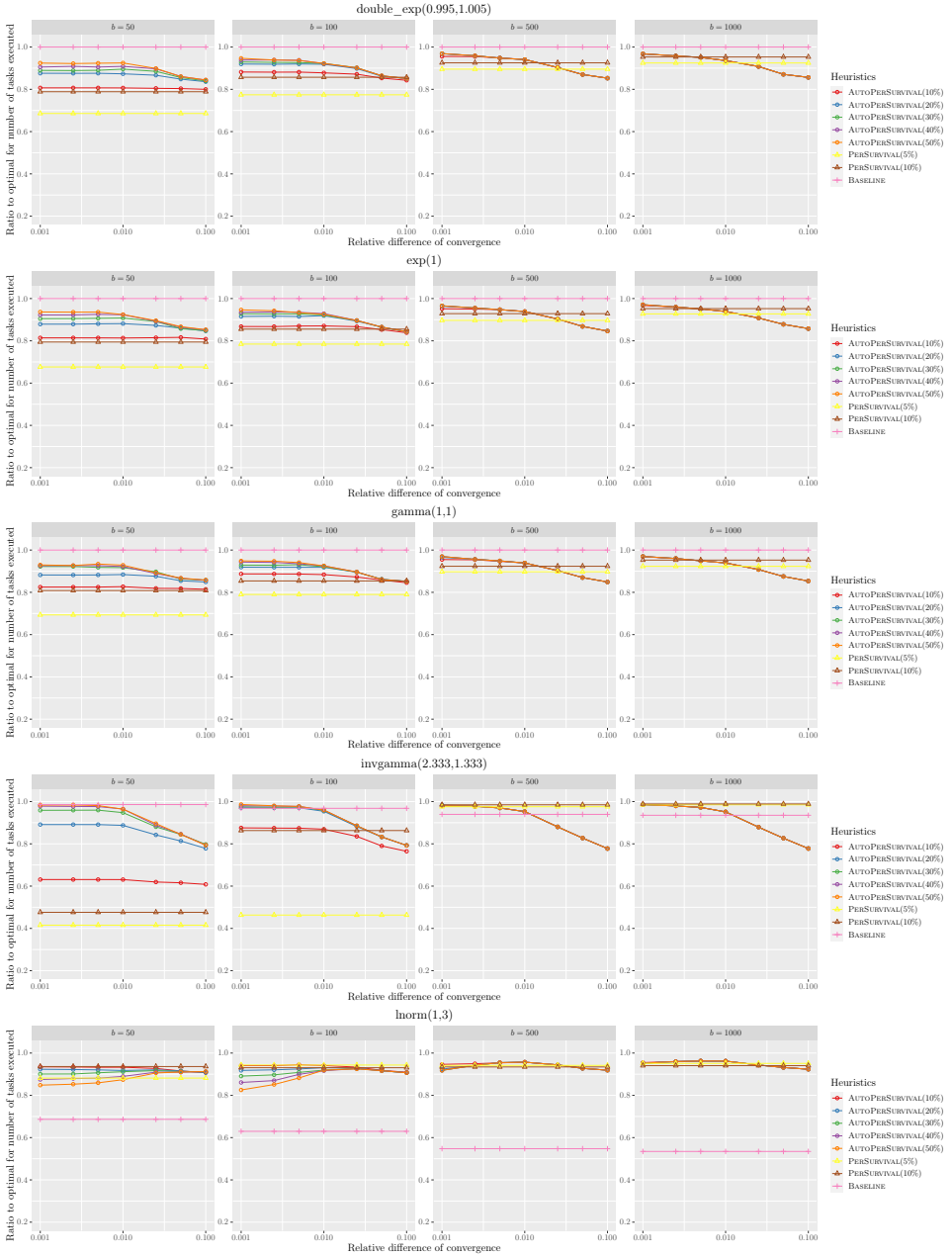


Figure 8: Number of successfully executed tasks for each distribution using AutoPerSurvival (different p_{max} when varying ϵ) and PerSurvival (different p) (2/3).

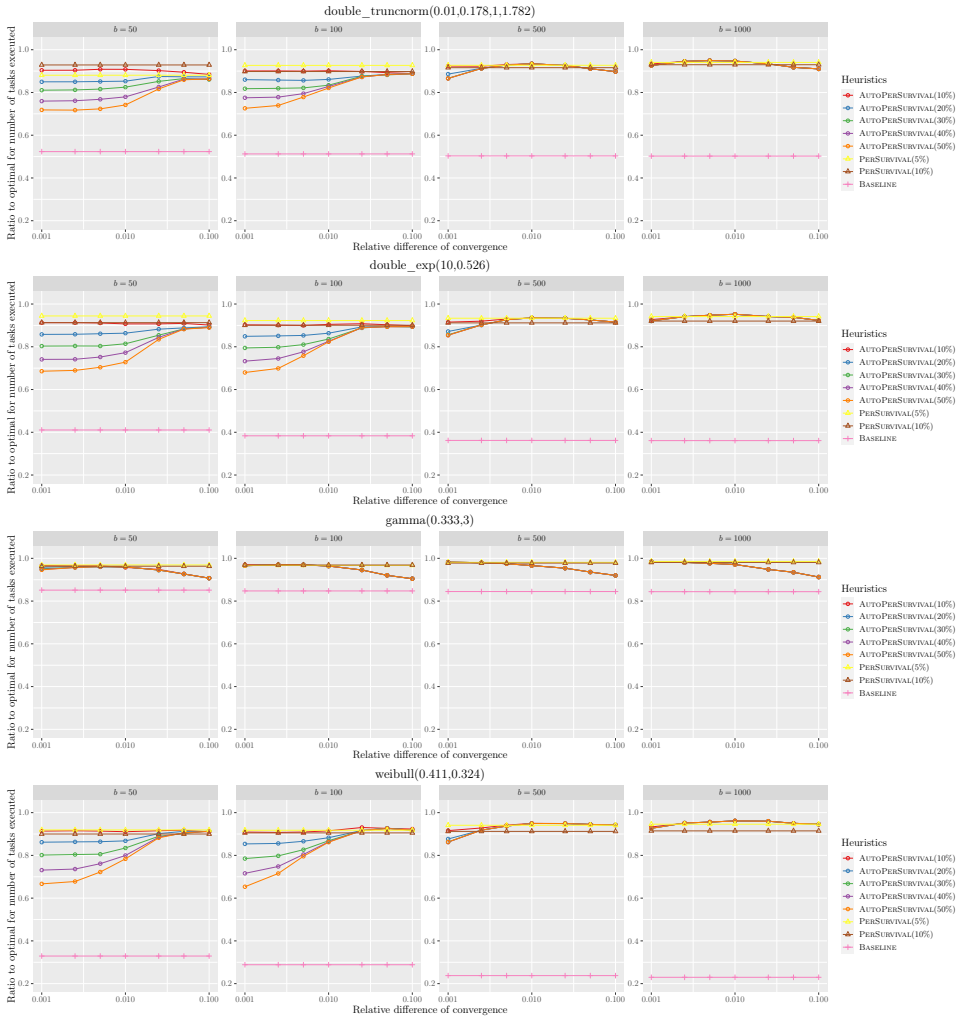


Figure 9: Number of successfully executed tasks for each distribution using AutoPerSurvival (different p_{max} when varying ϵ) and PerSurvival (different p) (3/3).

budget spent during the observation phase. Hence, if ϵ is sufficiently small, the behavior of AutoPerSurvival is only dictated by the value of p_{max}. For instance, when $b = 50$, this is the case for the uniform distribution when $\epsilon \leq 0.0050$, and for the Weibull distribution 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 lnorm(1,3) 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 $\epsilon \leq 0.0100$) performs similarly to PerSurvival, and they both have a good performance (larger than 90%). 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%, 0.01) 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%, 0.01) will always lead to good results, contrarily to all *one-size-fits-all* heuristics.

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

Figures 10 and 11 assess the performance of the different heuristics under a log-normal distribution of task execution times when $b = 1000$ (Figure 10) and $b = 50$ (Figure 11) for different values of the average task execution time (μ), of the standard deviation (σ,) and of the number of VMs in the platform (M). We use a log-normal distribution because it has been advocated to

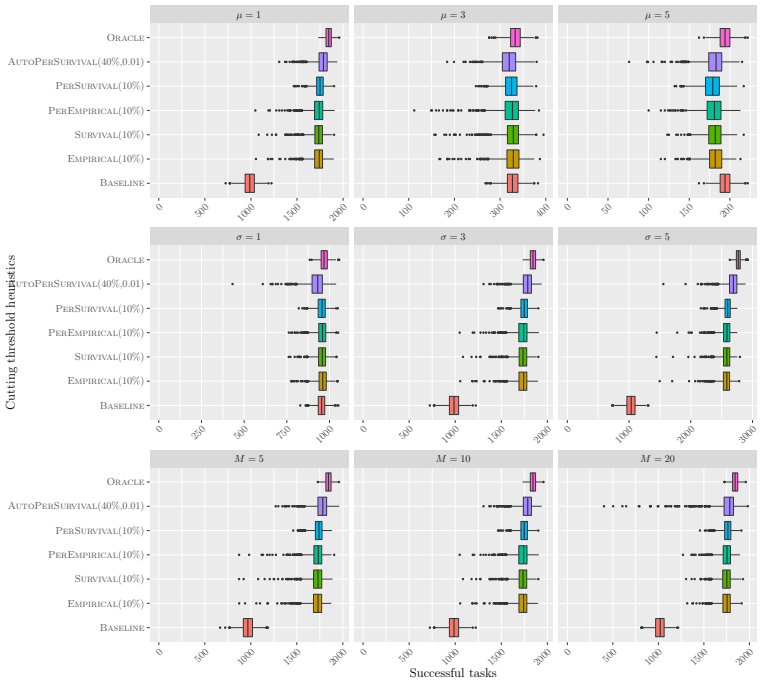


Figure 10: Number of successfully executed tasks for the different heuristics with a budget $b = 1000$ when task execution times follow a log-normal distribution. Unless otherwise specified, the expectation is $\mu = 1$, the standard deviation is $\sigma = 3$, and the number of VMs is $M = 10$.

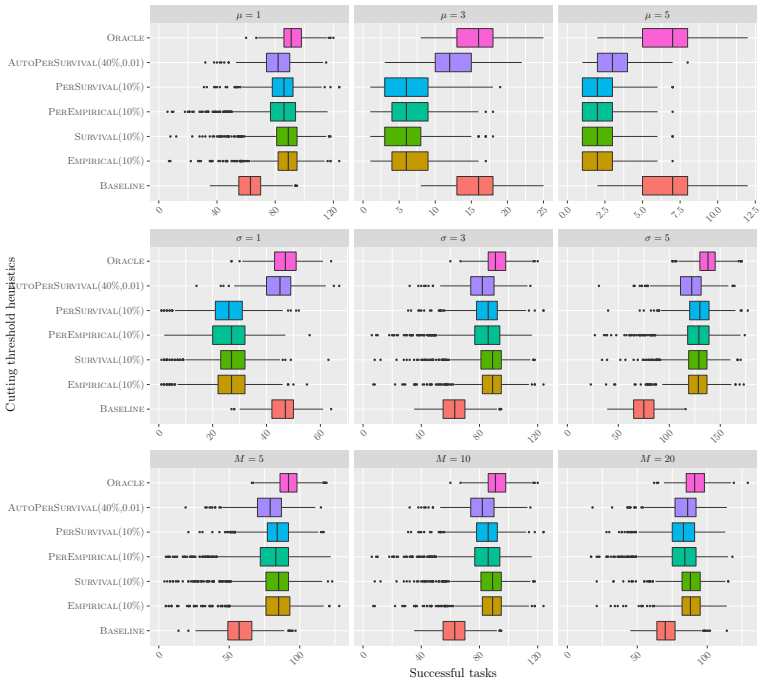


Figure 11: Number of successfully executed tasks for the different heuristics with a budget $b = 50$ when task execution times follow a log-normal distribution. Unless otherwise specified, the expectation is $\mu = 1$, the standard deviation is $\sigma = 3$, and the number of VMs is $M = 10$.

model file sizes [19], 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 $\epsilon = 0.01$. For the four *one-size-fits-all* strategies, we use the same value to define the observation phase: $p = 10\%$.

In Figure 10, as 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 11 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 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 $\sigma = 1$.

The third row of subgraphs show that varying the number of VMs has no significant impact on the performance of the heuristics: all scenarios achieve near optimal performance. Overall, AutoPerSurvival (40%, 0.01) is a very robust heuristic, which overcomes the other heuristics in all settings, and which, in the most adverse scnearios, exhibits a graceful degradation of performance with respect to the theoretical optimal.

7.2.4 Conclusion

To summarize our findings, we finally present two tables showing the number of tasks completed 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 of the web supplementary material, $b = 1000$ and $d = 100$) and a small one (Table 5, $b = 50$ and $d = 5$) with respect to the

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

	NEVERINTERRUPT	AUTOPERSURVIVAL (40%,0.01)	PERSURVIVAL (10%)	SURVIVAL (10%)	PEREMPIRICAL (10%)	EMPIRICAL (10%)
unif(0,2)	1.0000	0.9217	0.9880	0.9887	0.9873	0.9881
truncnorm(0.8, 0.754)	1.0000	0.9170	0.9911	0.9914	0.9913	0.9914
lnorm(1,0.5)	1.0000	0.9186	0.9894	0.9909	0.9903	0.9909
lnorm(1.253)	1.0000	0.9168	0.9920	0.9920	0.9903	0.9903
double_truncnorm(0.5,0.534,1,1.068)	1.0000	0.9127	0.9881	0.9873	0.9860	0.9859
double_exp(0.995,1.005)	1.0000	0.9360	0.9690	0.9379	0.9370	0.9367
exp(1)	1.0000	0.9388	0.9685	0.9352	0.9362	0.9367
gamma(1,1)	1.0000	0.9391	0.9669	0.9397	0.9381	0.9391
invgamma(2.333,1.333)	0.9350	0.9512	0.9876	0.9871	0.9897	0.9881
lnorm(1,3)	0.5345	0.9620	0.9464	0.9352	0.9344	0.9351
double_truncnorm(0.01,0.178,1,1.782)	0.5023	0.9470	0.9338	0.9229	0.9259	0.9259
double_exp(10,0.526)	0.3610	0.9522	0.9236	0.9151	0.9169	0.9162
gamma(0.333,3)	0.8440	0.9711	0.9810	0.9801	0.9803	0.9799
weibull(0.411,0.324)	0.2296	0.9613	0.9183	0.9108	0.9109	0.9109

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

	NEVERINTERRUPT	AUTOPERSURVIVAL (40%,0.01)	PERSURVIVAL (10%)	SURVIVAL (10%)	PEREMPIRICAL (10%)	EMPIRICAL (10%)
unif(0,2)	1.0000	0.8872	0.4659	0.4522	0.4513	0.4775
truncnorm(0.8, 0.754)	1.0000	0.8904	0.4089	0.4002	0.3981	0.4288
lnorm(1,0.5)	1.0000	0.9230	0.3620	0.3646	0.3633	0.3667
lnorm(1.253)	1.0000	0.8874	0.6046	0.5701	0.5974	0.6146
double_truncnorm(0.5,0.534,1,1.068)	1.0000	0.8881	0.4989	0.4818	0.4940	0.5088
double_exp(0.995,1.005)	1.0000	0.9086	0.7846	0.7552	0.7927	0.8172
exp(1)	1.0000	0.9235	0.7973	0.7456	0.7916	0.8010
gamma(1,1)	1.0000	0.9221	0.8033	0.7604	0.8133	0.8203
invgamma(2.333,1.333)	0.9858	0.9647	0.4729	0.4738	0.4790	0.4788
lnorm(1,3)	0.6865	0.8896	0.9238	0.9048	0.9484	0.9493
double_truncnorm(0.01,0.178,1,1.782)	0.5233	0.7794	0.9190	0.8831	0.9386	0.9420
double_exp(10,0.526)	0.4107	0.7725	0.9190	0.8925	0.9068	0.9057
gamma(0.333,3)	0.8513	0.9599	0.9658	0.9592	0.9603	0.9610
weibull(0.411,0.324)	0.3291	0.7995	0.9198	0.9014	0.8802	0.8711

average task execution time ($\mu = 1$). Obviously, we use the same heuristic parameters than previously: $\epsilon = 0.01$, $p_{max} = 40\%$, and $p = 10\%$.

Table 4 shows that, with large values of budget and deadline, all heuristics perform well. Indeed, with the chosen parameters all heuristic achieve at least 91% 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 5 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 log-normal distribution). On the contrary, AUTOPERSURVIVAL always achieves good to very good performance: its worse case is 79% of the optimal. Once again, this shows the great robustness of AUTOPERSURVIVAL (40%, 0.01).

8 Conclusion

In this work, we have studied the problem of maximizing the number of tasks successfully executed on a cloud 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 difficulty 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 recompute the threshold as the execution progresses. On the practical side, extensive simulations show that our best heuristic AutoPerSurvival (40%, 0.01) achieves good performance for a wide spectrum of probability distributions and parameter sets. In the worst scenario, it can execute 79% of tasks that an omniscient oracle (knowing the distribution) would be able to complete.

Future work will be dedicated to considering heterogeneous VMs, still under the assumption that the distribution of task execution times is unknown on the different VMs. Indeed, some cloud providers provide different categories of VMs 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 VM subset, and the estimation of the cutting threshold on each of them.

Acknowledgment

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).

References

[1] Odd Aalen, Ornulf Borgan, and Hakon Gjessing. *Survival and event history analysis: a process point of view*. Springer Science & Business Media, 2008.

[2] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick H.J. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158 – 169, 2013. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

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

[4] V. Arabnejad, K. Bubendorfer, and B. Ng. Budget distribution strategies for scientific workflow scheduling in commercial clouds. In *12th IEEE International Conference on e-Science*, pages 137–146, Oct 2016.

[5] AR. Arunarani, D. Manjula, and Vijayan Sugumaran. Task scheduling techniques in cloud computing: A literature survey. *Future Generation Computer Systems*, 91:407 – 415, 2019.

[6] Mohammad Ubaidullah Bokhari, Qahtan Makki, and Yahya Kord Tamandani. A survey on cloud computing. In D. Mishra V. Aggarwal, V. Bhatnagar, editor, *Big Data Analytics*, volume 654 of *Advances in Intelligent Systems and Computing*. Springer, 2018.

[7] Giorgio Buttazzo. Handling overload conditions in real-time systems. In Seyed Morteza Babamir, editor, *Real-Time Systems, Architecture, Scheduling, and Application*, chapter 7. InTech, Rijeka, 2012.

[8] Eun-Kyu Byun, Yang-Suk Kee, Jin-Soo Kim, and Seungryoul Maeng. Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8):1011 – 1026, 2011.

[9] Zhicheng Cai, Xiaoping Li, Rubén Ruiz, and Qianmu Li. 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, 2017.

[10] R. N. Calheiros and R. Buyya. Meeting deadlines of scientific workflows in public clouds with tasks replication. *IEEE Transactions on Parallel and Distributed Systems*, 25(7):1787–1796, July 2014.

[11] Yves Caniou, Eddy Caron, Aurélie Kong Win Chang, and Yves Robert. Budget-aware scheduling algorithms for scientific workflows with stochastic task weights on heterogeneous iaas cloud platforms. In *27th International Heterogeneity in Computing Workshop HCW 2013*. IEEE Computer Society Press, 2018.

[12] Louis-Claude Canon, Aurélie Kong Win Chang, Yves Robert, and Frédéric Vivien. Scheduling independent stochastic tasks under deadline and budget constraints. In *SBAC-PAD*. IEEE, 2018.

[13] Louis-Claude Canon, Aurélie Kong Win Chang, Yves Robert, and Frédéric Vivien. Scheduling independent stochastic tasks under deadline and budget constraints. *Int. J. High Performance Computing Applications*, 34(2):246–264, 2019.

[14] Henri Casanova, Matthieu Gallet, and Frédéric Vivien. Non-clairvoyant scheduling of multiple bag-of-tasks applications. In *Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference*, pages 168–179, 2010.

[15] Lloyd E. Chambless and Guoqing Diao. Estimation of time-dependent area under the roc curve for long-term risk prediction. *Statistics in Medicine*, 25(20):3474–3486, 2006.

[16] J. Y. Chung, J. W. S. Liu, and K. J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Trans. Computers*, 39(9):1156–1174, 1990.

[17] Van der Vaart. *Asymptotic Statistics*. Cambridge University Press, 1998.

[18] H. M. Fard, R. Prodan, and T. Fahringer. A truthful dynamic workflow scheduling mechanism for commercial multicloud environments. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1203–1212, June 2013.

[19] D Feitelson. Workload modeling for computer systems performance evaluation. *Version 1.0.3*, pages 1–607, 2014.

[20] W. Feng and J. W. S. Liu. An extended imprecise computation model for time-constrained speech processing and generation. In *Proc. IEEE Workshop on Real-Time Applications*, pages 76–80, May 1993.

[21] Thomas S Ferguson. *Optimal stopping and applications*. UCLA Press, 2008.

[22] Y. Gao, L. Canon, Y. Robert, and F. Vivien. Scheduling independent stochastic tasks on heterogeneous cloud platforms. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11, 2019.

[23] Y. Gao, Yanzhi Wang, S. K. Gupta, and M. Pedram. An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems. In *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, September 2013.

[24] Yiqin Gao. Resource-constrained scheduling of stochastic tasks with unknown probability distribution, Nov 2020.

[25] Anastasia Grekioti and Natalia V. Shakhlevich. Scheduling bag-of-tasks applications to optimize computation time and cost. In *PPAM*, volume 8385 of *LNCS*. Springer, 2014.

[26] Patricia Guyot, A. E. Ades, Mario JNM Ouwens, and Nicky J. Welton. Enhanced secondary analysis of survival data: reconstructing the data from published kaplan-meier survival curves. *BMC Medical Research Methodology*, 12(1):9, Feb 2012.

[27] Ibrahim Abaker Targio Hashem, Nor Badrul Anuar, Mohsen Marjani, Ejaz Ahmed, Haruna Chiroma, Ahmad Firdaus, Muhamad Taufik Abdullah, Faiz Alotaibi, Waleed Kamaleldin Mahmoud Ali, Ibrar Yaqoob, and Abdullah Gani. Mapreduce scheduling algorithms: a review. *The Journal of Supercomputing*, 76(7):4915–4945, Jul 2020.

[28] Houcine Hassan, José Simó, and Alfons Crespo. Flexible real-time mobile robotic architecture based on behavioural models. *Engineering Applications of Artificial Intelligence*, 14(5):685 – 702, 2001.

[29] Eunji Hwang and Kyong Hoon Kim. Minimizing cost of virtual machines for deadline-constrained mapreduce applications in the cloud. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, GRID '12, pages 130–138, Washington, DC, USA, 2012. IEEE Computer Society.

[30] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. *J. ACM*, 65(1), December 2017.

[31] Fabrice Jumel and Françoise Simonot-Lion. Management of anytime tasks in real time applications. In *XIV Workshop on Supervising and Diagnostics of Machining Systems*, Karpacz/Pologne, 2003.

[32] E. L. Kaplan and Paul Meier. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53(282):457–481, 1958.

[33] H. Kobayashi and N. Yamasaki. Rt-frontier: a real-time operating system for practical imprecise computation. In *10th IEEE Real-Time and Embedded Tech. Appl. Symp.*, pages 255–264, May 2004.

[34] Keqin Li. Non-clairvoyant scheduling of independent parallel tasks on single and multiple multicore processors. *Journal of Parallel and Distributed Computing*, 133:210 – 220, 2019.

[35] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. In André M. van Tilborg and Gary M. Koob, editors, *Foundations of Real-Time Computing: Scheduling and Resource Management*, pages 203–249. Springer, 1991.

[36] Ke Liu, Hai Jin, Jinjun Chen, Xiao Liu, Dong Yuan, and Yun Yang. A compromised-time-cost scheduling algorithm in swindew-c for instance-intensive cost-constrained workflows on a cloud computing platform. *Int. J. High Performance Computing Applications*, 24(4):445–456, 2010.

[37] Olivier Lopez. 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, 2012.

[38] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11. IEEE, Nov 2012.

[39] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. *Future Gen. Comp. Syst.*, 48:1–18, 2015.

[40] M. Mao, J. Li, and M. Humphrey. Cloud auto-scaling with deadline and budget constraints. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 41–48. IEEE, October 2010.

[41] Jiayuan Meng, S. Chakradhar, and A. Raghunathan. Best-effort parallel execution framework for recognition and mining applications. In *IPDPS*. IEEE, 2009.

[42] A. M. Oprescu and T. Kielmann. Bag-of-tasks scheduling under budget constraints. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 351–359, November 2010.

[43] A. M. Oprescu, T. Kielmann, and H. Leahu. Stochastic tail-phase optimization for bag-of-tasks execution in clouds. In *Fifth Int. Conf.s on Utility and Cloud Computing*, pages 204–208. IEEE, November 2012.

[44] Ana-Maria Oprescu, Thilo Kielmann, and Haralambie Leahu. Budget estimation and control for bag-of-tasks scheduling in clouds. *Parallel Processing Letters*, 21(02):219–243, 2011.

[45] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao. Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In *AINA 2014*, pages 858–865, May 2014.

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

[47] Pawan Singh, Baseem Khan, Ankit Vidyarthi, Hassan Haes Alhelou, and Pierluigi Siano. Energy-aware online non-clairvoyant scheduling using speed scaling with arbitrary power function. *Applied Sciences*, 9(7), 2019.

[48] Sukhpal Singh and Inderveer Chana. Cloud resource provisioning: survey, status and future research directions. *Knowledge and Information Systems*, 49(3):1005–1069, December 2016.

[49] Sukhpal Singh and Inderveer Chana. A survey on resource scheduling in cloud computing: Issues and challenges. *J. Grid Comp.*, 14(2):217–264, 2016.

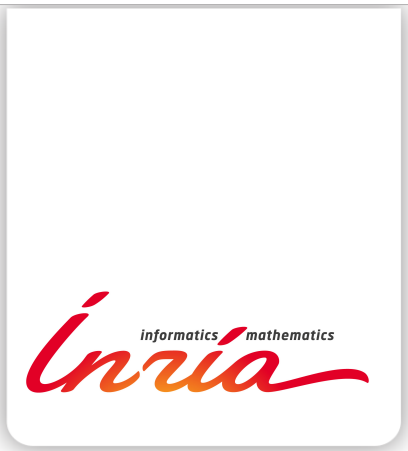
[50] Long Thai, Blesson Varghese, and Adam Barker. A survey and taxonomy of resource optimisation for executing bag-of-task applications on public clouds. *Future Generation Computer Systems*, 82:1 – 11, 2018.

[51] F. Tian and K. Chen. Towards optimal resource provisioning for running mapreduce programs in public clouds. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 155–162. IEEE, July 2011.

[52] Christian Vecchiola, Rodrigo N. Calheiros, Dileban Karunamoorthy, and Rajkumar Buyya. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka. *Future Generation Computer Systems*, 28(1):58 – 65, 2012.

[53] C. Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li. End-to-end delay minimization for scientific workflows in clouds under budget constraint. *IEEE Transactions on Cloud Computing*, 3(2):169–181, April 2015.

[54] Jun Xie and Chaofeng Liu. Adjusted kaplan–meier estimator and log-rank test with inverse probability of treatment weighting for survival data. *Statistics in Medicine*, 24(20):3089–3110, 2005.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée

655 avenue de l'Europe Montbonnot

38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399