Scheduling stochastic tasks on heterogeneous cloud platforms under budget and deadline constraints

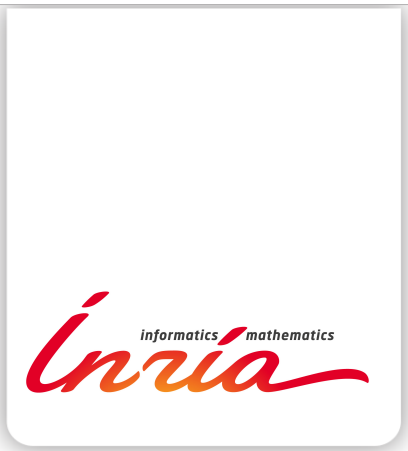
Yiqin Gao, Louis-Claude Canon, Frédéric Vivien, Yves Robert

Scheduling stochastic tasks on heterogeneous cloud platforms under budget and deadline constraints

Yiqin Gao, Louis-Claude Canon, Yves Robert, Frédéric Vivien

RESEARCH REPORT
N° 9260
February 2019
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--9260--FR+ENG



Scheduling stochastic tasks on heterogeneous cloud platforms under budget and deadline constraints

Yiqin Gao*, Louis-Claude Canon†, Yves Robert*‡, Frédéric Vivien*

Project-Team ROMA

Research Report n° 9260 — February 2019 — 34 pages

* LIP, École Normale Supérieure de Lyon, CNRS & Inria, France
† FEMTO-ST, Université de Bourgogne Franche-Comté, France
‡ University of Tennessee Knoxville, USA

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

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. The cloud platform is composed of several types of virtual machines (VMs), where each type has a unit execution cost that depends upon its characteristics. The amount of budget spent during the execution of a task on a given VM is the product of its execution length by the unit execution cost of that VM. The execution length of tasks follow an exponential, uniform or lognormal probability distribution whose mean and standard deviation both depend upon the VM type. Finally, there is a global available budget and a deadline constraint, and the goal is to successfully execute as many tasks as possible before the deadline is reached or the budget is exhausted (whichever comes first). On each VM, the scheduler can decide at any instant to interrupt the execution of a (long) running task and to launch a new one, but the budget already spent for the interrupted task is lost. The main questions are which VMs to enroll, and whether and when to interrupt tasks that have been executing for some time. We assess the complexity of the problem by showing its NP-completeness and providing a 2-approximation for the asymptotic case where budget and deadline both tends to infinity. We introduce several heuristics and compare their performance by running an extensive set of simulations.

Key-words: independent tasks, stochastic execution times, cloud platform, heterogeneity, budget, deadline.

Ordonnancement de tâches stochastiques sur plateformes hétérogènes avec contraintes de budget et d'échéance

Résumé : Ce travail présente des stratégies d'ordonnancement permettant de maximiser le nombre attendu de tâches indépendantes pouvant être exécutées sur une plateforme cloud dans le cadre d'un budget donné et d'une date limite contrainte. La plateforme cloud est composée de plusieurs types de machines virtuelles (VM), chaque type ayant un coût d'exécution unitaire qui dépend de ses caractéristiques. Le montant du budget dépensé lors de l'exécution d'une tâche sur une VM donnée est le produit de son temps d'exécution par le coût d'exécution unitaire de cette VM. Le temps d'exécution des tâches suit une distribution de probabilité exponentielle, uniforme ou log-normale dont la moyenne et l'écart type dépendent tous du type de la VM. Enfin, il existe un budget disponible global et une contrainte de date limite, et l'objectif est d'exécuter avec succès le plus grand nombre de tâches possible avant la date limite ou l'épuisement du budget (suivant lequel vient en premier). Sur chaque VM, le planificateur 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. Les principales questions sont de choisir les VM à utiliser et le moment pour interrompre les tâches en cours d'exécution. Nous évaluons la complexité du problème en montrant sa NP-complétude et en fournissant une 2-approximation pour le cas asymptotique où le budget et la date limite tendent vers l'infini. Nous introduisons plusieurs méthodes heuristiques et comparons leurs performances en exécutant un vaste ensemble de simulations.

Mots-clés : tâches indépendantes, temps d'exécution stochastiques, plateforme cloud, hétérogénéité, budget, date limite.

1 Introduction

This paper deals with the following problem: given a cloud platform and a bag of tasks, how to maximize the number of successful task executions, given a budget and a deadline. The cloud platform is composed of several Virtual Machine (VM) types, each with a different unit cost and computing capacity. The execution time of the tasks follow a different probability distribution on each VM type to account for their different performance. In the paper, we use three widely used distributions, namely exponential, uniform and lognormal. For instance the expectation of the distribution could be inversely proportional to the raw speed of the VM, while the standard deviation can account for the interplay between task profiles and VM parameters, such as memory usage, communication pattern, etc.

A representative instance of this problem is the framework of *imprecise* computations [2, 15, 28]. Each task has a mandatory part, and an optional part. Once the mandatory parts have all been executed, there remains some budget and time to execute some optional parts. Each optional part refines the outcome of its associated mandatory part, and one aims at executing as many optional parts as possible. Because the optional parts are heavily data-dependent, their execution times are not deterministic but rather follow a probability distribution as stated above. Other examples are provided in Section 6, which is devoted to related work.

With a single VM, the problem is to decide whether, and when, to interrupt a long-lasting task, with the hope to launch a new one that would execute faster. Of course this is a risky decision, becauses (i) the budget already spent to execute the current task will be lost if it gets interrupted, and (ii) there is no guarantee that the new task will complete faster than the interrupted one. This problem was studied in our previous work [9, 11], which showed that there exists an optimal threshold at which each running task should be interrupted. Interrupting each yet unsuccessful task when it reaches this optimal threshold is shown to maximize the expected *success rate* on the VM, i.e., the average number of tasks successfully executed per time unit.

With several VMs of different types, the problem becomes dramatically more complicated, because we have to decide how many VMs to enroll, and of which type. In addition to success rate, the unit cost of the VM plays an important role and must be accounted for. In fact, the key parameter is the *yield*, defined as the ratio of the success rate over the unit cost: it gives the expected number of successful tasks per budget unit. Intuitively, one would like to sort available VMs by non-decreasing yields, and greedily enroll them in this order. With this greedy algorithm, there remains to determine how many VMs to enroll. We show how to determine this number and call Greedy the resulting greedy algorithm with the optimal number of VMs. Unfortunately, Greedy is not optimal. In fact, we show that the

problem to decide which VM to enroll is NP-complete, but that GREEDY is guaranteed to be a 2-approximation. These results lay the foundation for the complexity of the problem with several VMs. On the practical side, we compare GREEDY with a variety of other heuristics, using an extensive set of simulations, and observe that it always achieve a close-to-optimal performance, which is a clear indicator that it is the heuristic of choice for the optimization problem.

The rest of the paper is organized as follows. We detail the framework and objective in Section 2, and recall prior results for a single VM in Section 3. We provide complexity results (NP-completeness and 2-approximation algorithm) in Section 4. We compare these heuristics in Section 5, assessing their performance for an extensive set of simulation parameters. Section 6 surveys related work. Finally, we provide concluding remarks and directions for future work in Section 7. The algorithms are implemented in R and the related code, data and analysis are available in [20].

Platform	
$\mathcal{P}$	platform
M	number of VMs
VM_i	the i-th VM
c_i	unit cost of VM_i
ℓ_i	cutting threshold for task interruption on VM_i
$\mathcal{S}_i$	success rate of VM_i, computed using ℓ_i
$\mathcal{Y}_i$	yield of VM_i, where $\mathcal{Y}_i = \frac{\mathcal{S}_i}{c_i}$
$\mathcal{Y}^{tot}$	total platform yield
k	number of VM categories
m_j	number of VMs of type j (hence $M = \sum_{j=1}^{k} m_j$)
Tasks	
$\mathcal{D}_i$	probability distribution of execution times on VM_i
μ_i, σ_i	mean, standard deviation of $\mathcal{D}_i$
Constraints	
b	budget
d	deadline
K	ratio b/d

Table 1: Summary of main notations.

2 Problem definition

This section details the framework and the scheduling objective. See Table 1 for a summary of main notations.

2.1 Platform and tasks

We aim at scheduling a set of independent stochastic tasks on a cloud platform. The cloud platform is composed of a set of different VMs, each with their own characteristics. In the abstract formulation of the problem, there is a set $\mathcal{P} = \{\text{VM}_1, \text{VM}_2, \ldots, \text{VM}_M\}$ of M VMs. Each VM has a unit cost: c_i is the amount of budget spent per second for executing a task on VM_i. The execution time of a task on VM_i obeys a probability distribution $\mathcal{D}_i$ which is chosen as a probability distribution whose mean and standard deviation both depend on the characteristics of VM_i. The rationale for such a framework is the following. First, we assume that task execution times are data-dependent, as is the case in many applications (see Section 6), and therefore exhibit stochastic behaviors which can be nicely modeled by a probability distribution. Second, task execution times cannot be easily encapsulated as a mere function of the number of cores of their host VM, because many parameters such as memory usage and communication patterns must be taken into account. Therefore, it would not make sense to consider a unique probability distribution and simply scale it by a unique parameter, say the number of cores of each VM, to induce actual execution times on that VM. Instead, we use a different probability distribution for each VM, with values of mean and standard deviation accounting for heterogeneity sources. It makes sense to assume that the mean μ_i of $\mathcal{D}_i$, which is the expectation of execution times on VM_i, is somewhat related to the number of cores $nbcores_i$ of VM_i. In the experimental section (Section 5), we explore scenarios where mean values μ_i are inversely proportional to core counts $nbcores_i$, but we vary standard deviations σ_i to account for a wide range of heterogeneity degrees.

We focus on probability distributions that are those typically used in the literature, namely exponential, uniform, and lognormal distributions. More specifically:

- The exponential distribution enjoys the memoryless property, which implies that tasks can be interrupted at any instant without any impact on the overall performance (see Proposition 1 in Section 3.2 for more details). In this simplified setting, the scheduling decisions have no impact, and the problem reduces to selecting the best set of VMs. Thereby we can precisely assess the impact of resource selection on performance;
- The uniform distribution is a natural and simple choice when execution times lie within a bounded interval, which is the case for applications like dense and sparse linear algebra kernels;
- Finally, a lognormal distribution is a natural candidate because it has been advocated to model file sizes [17], and we assume that task costs could naturally obey this distribution too. Moreover, the lognormal distribution is positive, it has a tail that extends to infinity and the

logarithm of the data values are normally distributed.

We recall the density function of these three distributions in Table 2. Note that to ensure a given expected value μ and standard deviation σ, we set $a = \mu - \sigma\sqrt{3}$ and $b = \mu + \sigma\sqrt{3}$ for the uniform distribution, and $\alpha = \log(\mu) - \log(\sigma^2/\mu^2 + 1)/2$ and $\beta = \sqrt{\log(\sigma^2/\mu^2 + 1)}$ for the lognormal distribution. Note also that using a standard deviation $\sigma = 3$ for the lognormal distribution corresponds to a high level of heterogeneity. To see this intuitively, take a discrete distribution with 11 equally probable costs, 10 of value 0.1 and 1 of value 10: its expected value is $\mu = 1$ while its standard deviation is $\sigma \approx 2.85$.

Distribution	Density	Support	Mean	Std. Dev.
$\exp(\lambda)$	$f(x) = \lambda e^{-\lambda x}$	$x \in [0, \infty)$	$\frac{1}{\lambda}$	$\frac{1}{\lambda}$
uniform(a, b)	$f(x) = \frac{1}{b-a}$	$x \in [a, b]$	$\frac{a+b}{2}$	$\frac{b-a}{\sqrt{12}}$
lognormal(α, β)	$f(x) = \frac{e^{-\frac{(\log(x)-\alpha)^2}{2\beta^2}}}{x\beta\sqrt{2\pi}}$	$x \in [0, \infty)$	$e^{\alpha+\frac{\beta^2}{2}}$	$e^{\alpha+\frac{\beta^2}{2}}\sqrt{e^{\beta^2}-1}$

Table 2: Probability density functions.

Finally, in many experimental cloud platforms, there is only a reduced set of different VM types, with several available identical VMs per type. We let k be the number of types and m_j be the number of available VMs for type j, where $\sum_{j=1}^{k} m_j = M$.

2.2 Constraints and optimization objective

The user has a limited budget b and an execution deadline d. The optimization problem is to select a subset of VMs and to maximize the expected number of tasks that can be successfully completed on these VMs until the deadline is reached or the totality of the budget is spent. More precisely, the optimization problem $\textsc{Opt}(\mathcal{P}, b, d)$ is the following:

- Decide which VMs to launch: it can be any subset of $\mathcal{P}$;
- Each VM in $\mathcal{P}$ executes tasks continuously, as soon as it is started and until the deadline or the budget is exceeded, whichever comes first;
- At any time and on each VM, decide whether to interrupt the task that is currently executing and launch a new one: each task can be deleted by the scheduler at any time before completion;
- The execution of each task is non-preemptive. In a non-preemptive execution, interrupted tasks cannot be relaunched, and the time/budget spent computing until interruption is completely lost.

3 Cutting threshold

In this section, we recall previous results obtained with one VM and no deadline. To help the reader understand the approach, we make a brief detour through discrete distributions before coming back to continuous ones.

3.1 Discrete distributions

In this section, we review results obtained in [11]. We consider a simple problem instance with a single VM, a budget b, no deadline ($d = b$) and a unit cost c. We assume that task execution times on that VM follow a discrete probability distribution $\mathcal{D} = \{(p_i, w_i)\}_{1 \leq i \leq k}$. There are k possible execution times $w_1 < w_2 < \cdots < w_k$ (expressed in seconds) and a task has an execution time w_i with probability p_i, where $\sum_{i=1}^{k} p_i = 1$. The w_i's are also called *thresholds*, because they represent instants at which we should take a decision: if the current task did not complete successfully, then either we continue its execution (if the remaining budget allows for it), or we interrupt the task and start a new one. Of course the discrete distribution of the thresholds is somewhat artificial: in practice, we have continuous distributions for task execution times. With continuous distributions, at any instant, we do not know for sure that the task will continue executing until some fixed delay. On the contrary with discrete distributions, we know that the execution will continue (at least) until the next threshold. However, we build on the results obtained for discrete distributions to design efficient strategies for continuous distributions.

For $1 \leq i \leq k$, the i-th *fixed-threshold* strategy, or FTS_i, interrupts every unsuccessful task at threshold w_i, i.e., when the task has been executing for w_i seconds without completing. There are k such strategies, one per threshold. Informally, our criterion to select the best one is to maximize the *success rate*

$$\mathcal{S} = \frac{\text{expected number of tasks completed}}{\text{total execution time}},$$

or equivalently, the *yield*

$$\mathcal{Y} = \frac{\text{expected number of tasks completed}}{\text{budget}} = \frac{\mathcal{S}}{c}.$$

Indeed, the $\mathcal{S}$ ratio measures the average number of successful tasks per time unit, while the $\mathcal{Y}$ ratio measures the same quantity per budget unit. Formally, for a given budget b, the execution time is $\frac{b}{c}$ (no deadline), and we would like to compute

$$\mathcal{S}_i(b) = \frac{N_i(b)}{\frac{b}{c}} \quad \text{and} \quad \mathcal{Y}_i(b) = \frac{N_i(b)}{b} \tag{1}$$

where $N_i(b)$ is the expected number of tasks that are successfully completed when using strategy FTS_i that interrupts all unsuccessful tasks after w_i

seconds, and proceeds until the budget b has been spent. It turns out that we can compute the limit $\mathcal{S}_i$ of $\mathcal{S}_i(b)$ when the budget b tends to infinity [11]:

$$\lim_{b\to\infty} \mathcal{S}_i(b) = \mathcal{S}_i \stackrel{\text{def}}{=} \frac{\sum_{j=1}^{i} p_j}{\sum_{j=1}^{i} p_j w_j + (1 - \sum_{j=1}^{i} p_j) w_i}. \tag{2}$$

Intuitively, the numerator is the success probability when interrupting at threshold i, and the denominator is the expected execution time: $p_j w_j$ for a success at threshold $j \leq i$, and $(1 - \sum_{j=1}^{i} p_j) w_i$ for an unsuccessful execution before threshold i.

The optimal fixed-threshold strategy FTS_{max} is defined as the strategy FTS_i whose ratio $\mathcal{S}_i$ is maximal. If several strategies FTS_i achieve the maximal ratio $\mathcal{R}_{max}$, we pick the one with smallest w_i (to improve success rate when the budget is limited and truncation must occur). Formally FTS_{max} is the strategy FTS_{i_0} where

$$i_0 = \min_{1\leq i\leq k} \{i \big| \mathcal{S}_i = \max_{1\leq j\leq k} \mathcal{S}_j\}.$$

It can be shown that FTS_{max} is asymptotically optimal among all possible scheduling strategies (see [9] for details).

3.2 Continuous distributions

We now consider the same problem instance as in Section 3.1 but with a continuous distribution $\mathcal{D}$, with expected value μ and standard deviation σ. Let $F(x)$ be its cumulative distribution function and $f(x)$ its probability density function.

We introduce the OptRatio scheduling strategy inspired by the asymptotically optimal strategy for discrete distributions. OptRatio interrupts all (unsuccessful) tasks at time

$$\ell^{max} = \arg\max_{l} \mathcal{S}(l) \;\; \textit{where} \;\; \mathcal{S}(l) = \frac{F(l)}{\int_0^l x f(x) dx + l(1 - F(l))}. \tag{3}$$

The idea behind OptRatio is that it maximizes the ratio of the probability of success (namely $F(l)$) to the expected execution time spent for a single task, when each task is interrupted at time l (i.e., $\int_0^l x f(x) dx$ for the cases when the task terminates sooner than l and $\int_l^\infty l f(x) dx = l(1 - F(l))$ otherwise). This is a continuous extension of Equation (1), and we expect OptRatio to perform well for large budgets. Intuitively, the continuous distributions under study can be uniformly and arbitrarily closely approximated by discrete distributions, hence we conjecture that OptRatio is asymptotically optimal, although we have no formal proof of this result.

We prove the following characterization of ℓ^{max} for exponential and uniform distributions:

Proposition 1. *For the distribution* $\exp(\lambda)$, $\mathcal{S}(l) = \lambda = \frac{1}{\mu}$ *for all values of* l, *and* ℓ^{max} *can be chosen arbitrarily (interrupt tasks at any time). For the distribution uniform*(a, b), $\ell^{max} = b$ *and* $\mathcal{S}(\ell^{max}) = \frac{2}{a+b} = \frac{1}{\mu}$ *(never interrupt tasks).*

Proof. The distribution $\exp(\lambda)$ is memoryless, so it is no surprise that $\mathcal{S}(l)$ is constant. We can check that analytically: we have $F(x) = 1 - e^{-\lambda x}$ and $f(x) = \lambda e^{-\lambda x}$. Hence:

$$\begin{aligned}\mathcal{S}(l) &= \frac{F(l)}{\int_0^l xf(x)dx + l(1-F(l))} \\ &= \frac{1-e^{-\lambda l}}{\lambda \int_0^l xe^{-\lambda x}dx + le^{-\lambda l}}.\end{aligned}$$

$$\begin{aligned}\int_0^l xe^{-\lambda x}dx &= \left[x(-\frac{1}{\lambda}e^{-\lambda x})\right]_0^l - \int_0^l (-\frac{1}{\lambda}e^{-\lambda x})dx \\ &= -\frac{1}{\lambda}\left[xe^{-\lambda x}\right]_0^l - (-\frac{1}{\lambda})\left[-\frac{1}{\lambda}e^{-\lambda x}\right]_0^l \\ &= -\frac{1}{\lambda}le^{-\lambda l} - \frac{1}{\lambda^2}(e^{-\lambda l} - 1).\end{aligned}$$

$$\begin{aligned}\mathcal{S}(l) &= \frac{1-e^{-\lambda l}}{\lambda[-\frac{1}{\lambda}le^{-\lambda l} - \frac{1}{\lambda^2}(e^{-\lambda l}-1)] + le^{-\lambda l}} \\ &= \frac{1-e^{-\lambda l}}{-le^{-\lambda l} + \frac{1}{\lambda}(1-e^{-\lambda l}) + le^{-\lambda l}} \\ &= \lambda.\end{aligned}$$

For the distribution uniform(a, b), the intuition is that it is never worth interrupting a task because all completion times are equiprobable, so better capitalize on prior execution. It is also easy to check that analytically: we have $F(x) = \frac{x-a}{b-a}$ and $f(x) = \frac{1}{b-a}$. Hence:

$$\begin{aligned}\mathcal{S}(l) &= \frac{F(l)}{\int_0^l xf(x)dx + l(1-F(l))} \\ &= \frac{\frac{l-a}{b-a}}{\int_a^l x\frac{1}{b-a}dx + l(1-\frac{l-a}{b-a})} \\ &= \frac{l-a}{\int_a^l xdx + l(b-l)} \\ &= \frac{2(l-a)}{l^2 - a^2 + 2bl - 2l^2} \\ &= \frac{2(l-a)}{-l^2 + 2bl - a^2}.\end{aligned}$$

To find $\mathcal{S}(\ell^{max})$, let us compute the differentiate $\mathcal{S}'(l)$:

$$\begin{aligned}
\mathcal{S}'(l) &= \frac{2(l-a)'(-l^2+2bl-a^2) - 2(l-a)(-l^2+2bl-a^2)'}{(-l^2+2bl-a^2)^2} \\
&= \frac{2(-l^2+2bl-a^2) - 2(l-a)(-2l+2b)}{(-l^2+2bl-a^2)^2} \\
&= \frac{-2l^2+4bl-2a^2+4l^2-4bl-4la+4ab}{(-l^2+2bl-a^2)^2} \\
&= \frac{2l^2-4la-2a^2+4ab}{(-l^2+2bl-a^2)^2} \\
&= \frac{2(l-a)^2+4a(b-a)}{(-l^2+2bl-a^2)^2}.
\end{aligned}$$

We have $\mathcal{S}'(l) \geq 0$ when $a \leq l \leq b$. Hence $\mathcal{S}(\ell^{max}) = \mathcal{S}(b)$, where

$$\begin{aligned}
\mathcal{S}(\ell^{max}) &= \mathcal{S}(b) \\
&= \frac{2(b-a)}{-b^2+2b^2-a^2} \\
&= \frac{2(b-a)}{b^2-a^2} \\
&= \frac{2}{a+b}.
\end{aligned}$$

□

However, for a lognormal(α, β) distribution, we do not have computed ℓ^{max} analytically, but we provide a program [12] to compute it numerically. For example when $\mu = 1$ and $\sigma = 3$ we find $\ell^{max} \approx 0.1$. Altogether, either analytically or numerically, we compute the maximum success rate $\mathcal{S}(\ell^{max})$ and maximum yield $\mathcal{Y}(\ell^{max}) = \frac{\mathcal{S}(\ell^{max})}{c}$ of the VM for the continuous distribution $\mathcal{D}$.

4 Complexity results

In this section, we present complexity results with several VMs, assuming a large budget and a large deadline. We start by formulating the asymptotic optimization problem in Section 4.1. We assess its complexity in Section 4.1.1. Then we introduce a greedy polynomial heuristic in Section 4.1.2, and show that it is a 2-approximation.

4.1 Problem instance with $b = Kd$

We aim at generalizing the asymptotic results obtained for a single VM (Section 3) to the case of several VMs. We used to have no deadline for

the single VM case. However, with several VMs, if there is no deadline, the best solution is to use a single VM, namely the one with highest yield $\mathcal{Y}(\ell^{max})$. Introducing a deadline makes parallelism unavoidable, and raises the question of selecting which VMs to enroll. In the following, we assume that budget and deadline are proportional:

$$b = Kd$$

for some constant $K > 1$, and aim at deriving asymptotic results when b tends to infinity under that constraint. Intuitively, K represents the total cost per time unit available until deadline d and it denotes the potential parallelism.

Consider a given VM $\text{VM}_i \in \mathcal{P}$. Given the distribution $\mathcal{D}_i$ of task execution times on VM_i, we compute the maximum threshold ℓ_i^{max} at which to interrupt tasks, and derive the success rate $\mathcal{S}_i = \mathcal{S}(\ell_i^{max})$ and yield $\mathcal{Y}_i = \frac{\mathcal{S}_i}{c_i}$, where c_i is the unit cost of VM_i.

Now assume that we enroll a subset $\mathcal{Q} = \{\text{VM}_i, i \in Q\}$ of VMs from $\mathcal{P}$. Here, Q simply represents the subset of $\{1, 2, \ldots, M\}$ that records the indices of enrolled VMs. These VMs will work continuously until the budget is exhausted or the deadline has been reached, whichever comes first. If the VMs in $\mathcal{Q}$ work for a duration t, the total budget spent is $t \times \sum_{i \in Q} c_i$ hence

$$t = \min\left(d, \frac{b}{\sum_{i \in Q} c_i}\right) = \min\left(\frac{b}{K}, \frac{b}{\sum_{i \in Q} c_i}\right) = \frac{b}{\max(K, \sum_{i \in Q} c_i)}.$$

Asymptotically, each VM_i, with $i \in Q$, is successfully executing $\mathcal{S}_i$ task per time unit, hence the total yield of subset $\mathcal{Q}$ is

$$\mathcal{Y}^{tot} = \frac{\sum_{i \in Q} \mathcal{S}_i}{\max(K, \sum_{i \in Q} c_i)}. \tag{4}$$

We are ready to define the asymptotic optimization problem with several VMs:

Definition 1 (OptHetero)**.** *Given the set $\mathcal{P}$ of available VMs and the constraint $b = Kd$, determine the subset $\mathcal{Q}$ of $\mathcal{P}$ so that the value of $\mathcal{Y}^{tot}$ in Equation* (4) *is maximized. Here, each VM in $\mathcal{Q}$ obeys the* OptRatio *strategy and interrupts all tasks at time ℓ_i^{max}, with expected success rate $\mathcal{S}_i$.*

4.1.1 NP-completeness

In this section, we show that the decision problem associated to OptHetero is NP-complete. For simplicity, we use the same name for the decision and optimization problems.

Theorem 1. OptHetero *is NP-complete.*

Proof. The decision problem is the following: given the set $\mathcal{P}$ of available VMs and the constraint $b = Kd$, and given a bound on the total yield $\mathcal{Z}$, can we find a subset $\mathcal{Q}$ of $\mathcal{P}$ with total yield $\mathcal{Y}^{tot} \geq \mathcal{Z}$? The problem obviously belongs to the class NP, a certificate being the subset of enrolled VMs, and whose yield can be computed in linear time. For the completeness, we make a reduction from SubsetSum, a well-known NP-complete problem [22]. Consider an instance $\mathcal{I}_1$ of SubsetSum: given n positive integers $a_1, a_2, \ldots, a_n$ and a target T, can we find a subset J of $\{1, 2, \ldots, n\}$ such that $\sum_{i \in J} a_i = T$? We build the following instance $\mathcal{I}_2$ of OptHetero: a platform $\mathcal{P}$ with $M = n+1$ VMs, budget/deadline constraint $b = Kd$ where $K = T + 1$. VM characteristics are the following:

- VM_i, for $1 \leq i \leq n$, has success rate $\mathcal{S}_i = Ka_i$ and unit cost $c_i = a_i$
- VM_{n+1} has success rate $\mathcal{S}_{n+1} = 2K$ and unit cost $c_{n+1} = 1$.

Finally, the bound on total yield is $\mathcal{Z} = K + 1$. The size of $\mathcal{I}_2$ is clearly polynomial (and even linear) in the size of $\mathcal{I}_1$. We now show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ has a solution.

Assume first that $\mathcal{I}_1$ has a solution, i.e., a subset J with $\sum_{i \in J} a_i = T$. If we enroll all VMs whose index is in J plus VM_{n+1}, we obtain the total yield

$$\mathcal{Y}^{tot} = \frac{\sum_{i \in J} Ka_i + 2K}{\max(K, \sum_{i \in J} a_i + 1)} = \frac{KT + 2K}{\max(K, T + 1)} = \frac{K(K-1) + 2K}{K} = K + 1.$$

Hence, a solution to $\mathcal{I}_2$.

Assume now that $\mathcal{I}_2$ has a solution, i.e., an index subset Q with total yield $\mathcal{Y}^{tot} \geq \mathcal{Z} = K+1$. If the last VM is not enrolled, i.e., if $n+1 \notin Q$, then $\mathcal{Y}^{tot} = \frac{\sum_{i \in Q} Ka_i}{\max(K, \sum_{i \in Q} a_i)} \leq K$, a contradiction. Hence, necessarily $n + 1 \in Q$. Let $J = Q \setminus \{n + 1\}$, we are going to show that J is a solution of $\mathcal{I}_1$. We know that

$$\mathcal{Y}^{tot} = \frac{\sum_{i \in J} Ka_i + 2K}{\max(K, \sum_{i \in J} a_i + 1)} \geq K + 1.$$

Let $U = \sum_{i \in J} a_i$. If $U \geq K$ then $\mathcal{Y}^{tot} = \frac{KU+2K}{U+1} = K + \frac{K}{U+1} < K + 1$, a contradiction. If $U \leq K - 2$ then $\mathcal{Y}^{tot} = \frac{KU+2K}{K} = U + 2 < K + 1$, a contradiction. Hence, $U = K - 1 = T$, and J is a solution to $\mathcal{I}_1$. This concludes the proof. □

4.1.2 Greedy heuristic

The OptHetero problem is similar to a knapsack problem, and a natural heuristic is to enroll VMs with highest yield first. Table 3 shows a little example with a platform $\mathcal{P}$ consisting of $M = 5$ VMs. We use $K = 5$ in the example.

VM	Success rate	Unit cost	Yield
VM_1	$\mathcal{S}_1 = 10$	$c_1 = 1$	$\mathcal{Y}_1 = 10$
VM_2	$\mathcal{S}_2 = 6.2$	$c_2 = 3$	$\mathcal{Y}_2 \approx 2.1$
VM_3	$\mathcal{S}_3 = 8$	$c_3 = 4$	$\mathcal{Y}_3 = 2$
VM_4	$\mathcal{S}_4 = 6$	$c_4 = 4$	$\mathcal{Y}_4 = 1.5$
VM_5	$\mathcal{S}_5 = 4$	$c_4 = 4$	$\mathcal{Y}_5 = 1$

Table 3: Example of platform $\mathcal{P}$ ($M = 5$).

In Table 3, VMs are ordered by non-decreasing yield, so the greedy heuristic selects VM_1 first, then VM_2, etc. The performance achieved is the following:

- Using only VM_1: $\mathcal{Y}^{tot} = \frac{10}{\max(5,1)} = 2$;
- Using VM_1 and VM_2: $\mathcal{Y}^{tot} = \frac{10+6.2}{\max(5,1+3)} = 3.24$;
- Using VM_1, VM_2 and VM_3: $\mathcal{Y}^{tot} = \frac{10+6.2+8}{\max(5,1+3+4)} = 3.025$;
- Using VM_1, VM_2, VM_3 and VM_4: $\mathcal{Y}^{tot} = \frac{10+6.2+8+6}{\max(5,1+3+4+4)} \approx 2.5167$;
- Using all five VMs: $\mathcal{Y}^{tot} = \frac{10+6.2+8+6+4}{\max(5,1+3+4+4+4)} = 2.1375$.

In the example, the best choice is to use only VM_1 and VM_2, for a total yield $\mathcal{Y}^{tot} = 3.24$. In the following, we characterize how many VMs should be chosen. Finally, note that in the example, the optimal solution is to use only VM_1 and VM_3, for a total yield $\mathcal{Y}^{tot} = \frac{10+8}{\max(5,1+4)} = 3.6$.

Proposition 2. *Consider a platform $\mathcal{P}$ with M VMs ordered by non-increasing yields and with the constraint $b = Kd$. The total yield $\mathcal{Y}^{tot}$ achieved by the greedy heuristic is maximum when enrolling either the first $i^* - 1$ VMs or the first i^* VMs, whichever has the higher total yield, where i^* is the smallest index such that $\sum_{i=1}^{i^*} c_i > K$.*

In other words, the greedy heuristic should enroll VMs until their cumulated cost exceeds K, and then the best solution is either using all theses VMs or using all of them except the last one. In the example of Table 3, we have $i^* = 3$ and the best solution is with the first two VMs. We let Greedy denote the greedy heuristic which enrolls the optimal number of VMs. Note that when two different VMs have the same yield, we rank them and use the one with lowest unit cost first, which is better for scenarios where the budget is limited.

Proof. For $1 \leq i \leq M$, we consider the first i VMs and define

- the cumulated success rate $\mathcal{S}_i^{tot} = \sum_{j=1}^{i} \mathcal{S}_j$;

- the cumulated cost $\mathcal{C}_i^{tot} = \sum_{j=1}^{i} c_j$;
- the cumulated success/cost ratio $\mathcal{R}_i = \frac{\mathcal{S}_i^{tot}}{\mathcal{C}_i^{tot}}$.

Now the total yield achieved with the first i VMs is

$$\mathcal{Y}_i^{tot} = \min\left(\mathcal{R}_i, \frac{\mathcal{S}_i^{tot}}{K}\right).$$

Note that i^* is the smallest index i such that $\mathcal{C}_i^{tot} \geq K$.

First we observe that the $\mathcal{R}_i$ are non-increasing. This is because VMs are ordered by ratio $\frac{\mathcal{S}_i}{c_i}$. We easily check that

$$\frac{\mathcal{S}_1}{c_1} \geq \frac{\mathcal{S}_2}{c_2} \qquad \Rightarrow \qquad \frac{\mathcal{S}_1}{c_1} \geq \frac{\mathcal{S}_1 + \mathcal{S}_2}{c_1 + c_2} \geq \frac{\mathcal{S}_2}{c_2}$$

and the result follows by induction.

For $i \geq i^*$, we have $\mathcal{Y}_i^{tot} = \mathcal{R}_i \leq \mathcal{R}_{i^*} = \mathcal{Y}_{i^*}^{tot}$. For $i \leq i^* - 1$, we have $\mathcal{Y}_i^{tot} = \frac{\mathcal{S}_i^{tot}}{K} \leq \frac{\mathcal{S}_{i^*-1}^{tot}}{K} = \mathcal{Y}_{i^*-1}^{tot}$. This concludes the proof. □

In order to show that the performance of Greedy is within a factor two of the optimal, we define the FracOptHetero fractional version of the OptHetero problem. The only difference between FracOptHetero and OptHetero is that each VM enrolled at the beginning can now be stopped at any time before the deadline or the exhaustion of the budget. For FracOptHetero, the total yield is

$$\mathcal{Y}^{tot,frac} = \frac{\sum_{j \in \mathcal{P}} \mathcal{S}_j t_j}{b} \tag{5}$$

where t_j is the running time of VM_j. Formally:

Definition 2 (FracOptHetero)**.** *Given the set $\mathcal{P}$ of available VMs and the constraint $b = Kd$, determine t_j, which is the running time of the j-th VM in $\mathcal{P}$, so that the value of $\mathcal{Y}^{tot,frac}$ in Equation (5) is maximized (t_j is null if we do not use the j-th VM). Each VM_i in $\mathcal{P}$ obeys the* OptRatio *strategy and interrupts all tasks at time ℓ_i^{max}, with expected success rate $\mathcal{S}_i$.*

For this problem, the following variant of the greedy algorithm is optimal:

Proposition 3. *Consider a platform $\mathcal{P}$ with M VMs ordered by non-increasing yields and with the constraint $b = Kd$. An optimal solution for* FracOptHetero *is obtained by enrolling the first $i^* - 1$ VMs until the deadline and enrolling the i^*-th VM to exhaust the rest of the budget, where i^* is the smallest index such that $\sum_{i=1}^{i^*} c_i > K$.*

Proof. For the proof, we assume that i^* does exist, otherwise all VMs are enrolled until the deadline, which is optimal. Let t_i^{max} denote the running time of VM_i in the optimal solution, and t_i be its running time in the greedy algorithm. If an optimal solution is not making the greedy choice, there exists an index i such that $t_i^{max} > t_i$. Because the greedy algorithm uses the first $i^* - 1$ VMs until the deadline, we have $i \geq i^*$. Also, because the budget is exhausted by the greedy algorithm (from the existence of i^*), there must exist an index j such that $t_j^{max} < t_j$. Since the greedy algorithm does not use VMs of index $k \geq i^* + 1$, we have $j \leq i^*$, hence $j < i$. With the ordering method in the greedy algorithm, we can conclude that $\mathcal{Y}_i \leq \mathcal{Y}_j$. Then we re-distribute the amount of budget $\beta = \min\{(t_j - t_{j^*})\mathcal{S}_j, (t_{i^*} - t_i)\mathcal{S}_i\}$ to VM_j, instead of VM_i . The value of β guarantees that, after the re-distribution, VM_j spends not more budget than its does in the greedy algorithm. After the re-distribution, the yield of the optimal solution is increased by a positive value $\frac{\beta(\mathcal{Y}_j - \mathcal{Y}_i)}{b}$, which contradicts the optimality. This concludes the proof. □

Let $\mathcal{Y}^{max}$ be the highest yield for OptHetero, and $\mathcal{Y}^{opt\text{-}frac}$ be the highest yield for FracOptHetero problem. From Proposition 3, we know that $\mathcal{Y}^{opt\text{-}frac}$ is achieved by the greedy algorithm, which is given by

$$\mathcal{Y}^{opt\text{-}frac} = \frac{\mathcal{S}_{i^*-1}^{tot}}{K} + \left(1 - \frac{\mathcal{C}_{i^*-1}^{tot}}{K}\right)\frac{\mathcal{S}_{i^*}}{c_{i^*}}. \tag{6}$$

Proposition 4. Greedy *is a 2-approximation algorithm for* OptHetero.

Proof. We need to prove that: $\mathcal{Y}_{greedy}^{tot} \geq \frac{1}{2}\mathcal{Y}^{opt}$. We have

$$\begin{aligned}\mathcal{Y}_{greedy}^{tot} &= \max(\mathcal{Y}_{i^*-1}^{tot}, \mathcal{Y}_{i^*}^{tot}) \\ &= \max\left(\frac{\mathcal{S}_{i^*-1}^{tot}}{K}, \frac{\mathcal{S}_{i^*}^{tot}}{\mathcal{C}_{i^*}^{tot}}\right).\end{aligned}$$

Similarly to the proof of Proposition 2, we can easily prove by induction that:

$$\frac{\mathcal{S}_{i^*}^{tot}}{\mathcal{C}_{i^*}^{tot}} \geq \frac{\mathcal{S}_{i^*}}{c_{i^*}}.$$

As $0 \leq 1 - \frac{\mathcal{C}_{i^*-1}^{tot}}{K} \leq 1$, we have:

$$\frac{\mathcal{S}_{i^*}^{tot}}{\mathcal{C}_{i^*}^{tot}} \geq \left(1 - \frac{\mathcal{C}_{i^*-1}^{tot}}{K}\right)\frac{\mathcal{S}_{i^*}}{c_{i^*}}.$$

So we have:

$$\begin{aligned}
\mathcal{Y}^{tot}_{greedy} &\geq \max\left(\frac{\mathcal{S}^{tot}_{i^*-1}}{K}, \left(1-\frac{\mathcal{C}^{tot}_{i^*-1}}{K}\right)\frac{\mathcal{S}_{i^*}}{c_{i^*}}\right) \\
&\geq \frac{1}{2}\left[\frac{\mathcal{S}^{tot}_{i^*-1}}{K} + \left(1-\frac{\mathcal{C}^{tot}_{i^*-1}}{K}\right)\frac{\mathcal{S}_{i^*}}{c_{i^*}}\right] \\
&= \frac{1}{2}\mathcal{Y}^{opt\text{-}frac} \\
&\geq \frac{1}{2}\mathcal{Y}^{opt}.
\end{aligned}$$

□

5 Experiments

This section assesses the performance of several strategies to interrupt executing tasks and to choose the number and types of VMs to enroll for a given budget and deadline. The code and scripts used for the simulations and the data analysis are available online at `https://doi.org/10.6084/m9.figshare.7777046.v1`.

5.1 Heuristics for interrupting tasks

In the experiments in [11], authors compare different heuristics to decide the threshold at which tasks must be stopped on a single VM . For a single type of VM VM_i, these consist in:

- MeanVariance(x) is the family of heuristics that interrupt a task as soon as its execution time reaches $\mu_i + x\sigma_i$, where x is some constant (positive or negative).
- Quantile(x) is the family of heuristics that interrupt a task when its execution time reaches the x-quantile of the distribution $\mathcal{D}_i$ with $0 \leq x \leq 1$.
- OptRatio, the heuristic introduced in Section 3.

In our experiments, we assess the performance of these heuristics with different parameters.

5.2 Heuristics for choosing VMs

As we have different types and numbers of VMs, we need to find the optimal subset to be enrolled. This is especially true when we do not have enough budget to let the VM with highest yield run until the deadline. In order to achieve this goal, we compared the following methods for choosing VMs. The methods differ only in their criteria to order the VMs and then greedily

select the VMs in that order. Each method comes in two versions: the *limited (ltd)* version enrolls the first $i^* - 1$ VMs, where i^* is the smallest index such that $\sum_{i=1}^{i^*} c_i > K$; the refined version selects the best total yield when either using $i^* - 1$ VMs, as in the limited version, or using i^* VMs. Here are the three orderings:

- EXPECT$^{\text{ltd}}$ and EXPECT (computation-speed based methods): VMs are sorted by non-increasing computation speeds.
- COST$^{\text{ltd}}$ and COST (cost-per-time-unit methods): VMs are sorted by non-decreasing unit cost.
- GREEDY$^{\text{ltd}}$ and GREEDY (yield methods): VMs are sorted by non-decreasing yield. GREEDY is indeed the greedy algorithm of Section 4.1.2.

In addition, we assess the absolute performance of each method by comparing with *Fractional*, which is the yield acheived by the solution to the fractional problem FRACOPTHETERO (see Proposition 3). Indeed, the value of *Fractional* is an upper bound to the performance, which is not always tight; we use it as a reference.

5.3 Parameters

In the following experiments, tasks are interrupted according to different heuristics MEANVARIANCE(x), QUANTILE(x) and OPTRATIO.

All platforms are composed of six VM types, in other words, $k = 6$. Number of machines for each types m_j varies between 1 and 10.

Each VM type is characterized by a distribution that determines the execution time of each task. Type j VMs have average speed $s_j = 2^{j-1}$ (i.e. $s_j \in \{1, 2, 4, 8, 16, 32\}$). These values correspond to normalized speeds in realistic platforms such as Amazon EC2[1] or Google Cloud[2] and are correlated to the number of cores in the VMs. The unit cost of a VM depends upon its average speed. We choose 3 cases in our experiments: $c_j = s_j$, $c_j = 1.5s_j$ (proportional values) and $c_j = s_j^{1.5}$ (non proportional values).

The distribution of the execution times for each VM is either a lognormal, uniform or exponential. The parameters for the first two distributions allows us to fix the expectation and standard deviation of the execution times. The heterogeneity of a scheduling problem instance has several meanings (for instance, both tasks and machines heterogeneity are studied in [13]). In our case, we consider the heterogeneity of the expectation and the heterogeneity of the variability. For all tested distributions, the expectation of execution times is fixed as the inverse of the VM speed, which determines the first

[1] `https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls`
[2] `https://cloud.google.com/compute/pricing`

type of heterogeneity. For the second type, we control the variation of the Coefficient of Variation (CV), which is defined as the ratio of standard deviation over expectation. Similar CVs for all VMs lead to a low variability heterogeneity: execution times varies similarly on all VMs. On the contrary, different CVs mean that execution times are closer to their expectations on some VMs than on some others. For instance, two distributions with expectations 1 and 2 and the same CV 1 have expectation heterogeneity but no variability heterogeneity. This is the opposite with distributions both with expectation 1 and with CVs 1 and 2. This second type of heterogeneity is controlled by parameter x_{CV}.

For lognormal on type j, the expected value μ_j and standard deviation σ_j are set as follows: $\mu_j = \frac{1}{s_j}$, $\sigma_j = \mu_j U$ where the parameter U is drawn randomly and uniformly in the interval $[3 - x_{\mathrm{CV}}, 3 + x_{\mathrm{CV}}]$. Here, the heterogeneity parameter $0 \leq x_{\mathrm{CV}} \leq 3$ controls how much a VM type is heterogeneous in terms of variability. For exponential on type j, the parameter is $\lambda_j = \frac{1}{s_j}$ because the expected value of $\exp(\lambda)$ is $\mu = \frac{1}{\lambda}$. For uniform on type j, execution times follow a uniform distribution with $\mu_j = \frac{1}{s_j}$ and a standard deviation σ_j uniformly drawn between 0 and $\frac{\mu_j}{\sqrt{3}}$ (leading to a CV between 0 and $\frac{1}{\sqrt{3}}$).

We fix the budget at $b = \sum_i c_i$ and vary the deadline d (and thus K) between two extreme cases. The first case is when the deadline is sufficiently large to exhaust the entire budget by selecting a single VM. This is the case when $d = b$ (and thus $K = 1$) because the minimum unit cost is one. The second case is when the deadline is so constrained that all VMs must be used to exhaust the budget. This occurs when $d = 1$ (and thus $K = b$).

Each of the strategies is run 10 times on each of 100 platform instances. The numbers of successes is reported in boxplots, each of which consists of a bold line for the median, a box for the quartiles, whiskers that extend at most to 1.5 times the interquartile range from the box and additional points for outliers.

5.4 Results

The performances of the methods for choosing VMs (Section 5.2) and choosing time of cutting tasks (Section 5.1) are impacted by the budget and deadline that we have, and also the characteristics of the platforms and tasks.

The following figures depict the effect of the execution time distribution, the heuristic of cutting tasks, the variability heterogeneity by varying x_{CV}, the deadline constraint by varying d (and thus K) with fixed b, the number of each type of machines, the relationship between unit cost of a VM and its average speed.

Figure 1 shows the effect of the distribution on the performance of VM selection methods. The studied methods perform differently only with the

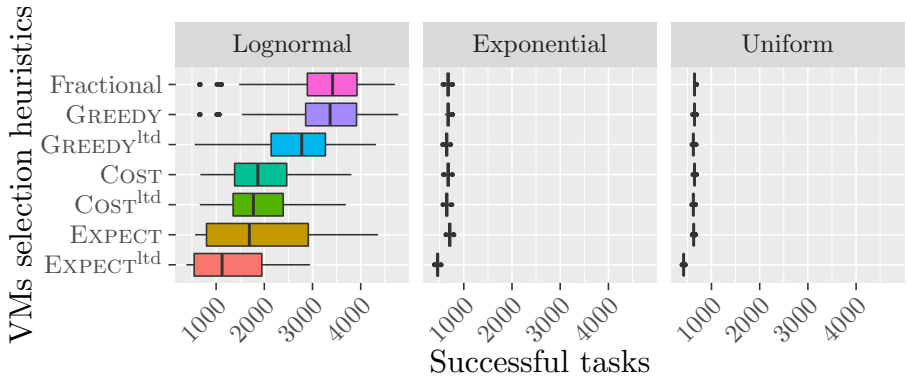


Figure 1: Number of successfully executed tasks for different methods to choose VMs when interrupting tasks with heuristic OPTRATIO. Each of the 100 platforms is generated with $m_j = 10$, $M = 60$, $c_j = s_j$ and is used 10 times with $b = 630$ and $d \approx 13.2$ ($K \approx 47.8$). Execution times follow either a lognormal distribution with $x_{\text{CV}} = 3$, an exponential one or a uniform one.

lognormal distribution. With this distribution, GREEDY is the only one to have the same performance as the fractional solution. The other methods lead to (significantly for some methods) fewer successful tasks. With the exponential and uniform distributions, all heuristics have the same optimal behavior (except EXPECT$^{\text{ltd}}$) because these distributions lead to the same yield for all VMs. Ordering them has then no impact as long as most of the budget is spent. EXPECT$^{\text{ltd}}$ ends up wasting a significant proportion of the budget because it selects first costly VMs. Even though the expected execution time is the same with all distributions, higher number of successful tasks are obtained with the lognormal distribution because tasks are interrupted early.

Figure 2 shows the number of successfully executed tasks for each heuristic of cutting threshold, when the parameter K varies. The result shows that, using the method GREEDY to choose VMs, OPTRATIO is always at least not worse than any other heuristics. We can also find that, except for OPTRATIO, QUANTILE (0.1) and QUANTILE (0.2) can also produce results better than others. This is because the threshold calculated by OPTRATIO is usually between 10^{-1} and 10^{-3} in our case and the threshold provided by QUANTILE (0.1) and QUANTILE (0.2) is closer to this value than other heuristics.

Figure 3 demonstrates the dependence between the level of variability heterogeneity controlled by x_{CV}, the deadline constraint by varying both d and K with a constant budget b, and the performance disparity between the methods. When $x_{\text{CV}} = 0$, heuristics are similar as in Figure 1 with exponential and uniform distributions for the same reason (i.e. all VMs have the same yield). As x_{CV} increases, the difference between the fractional

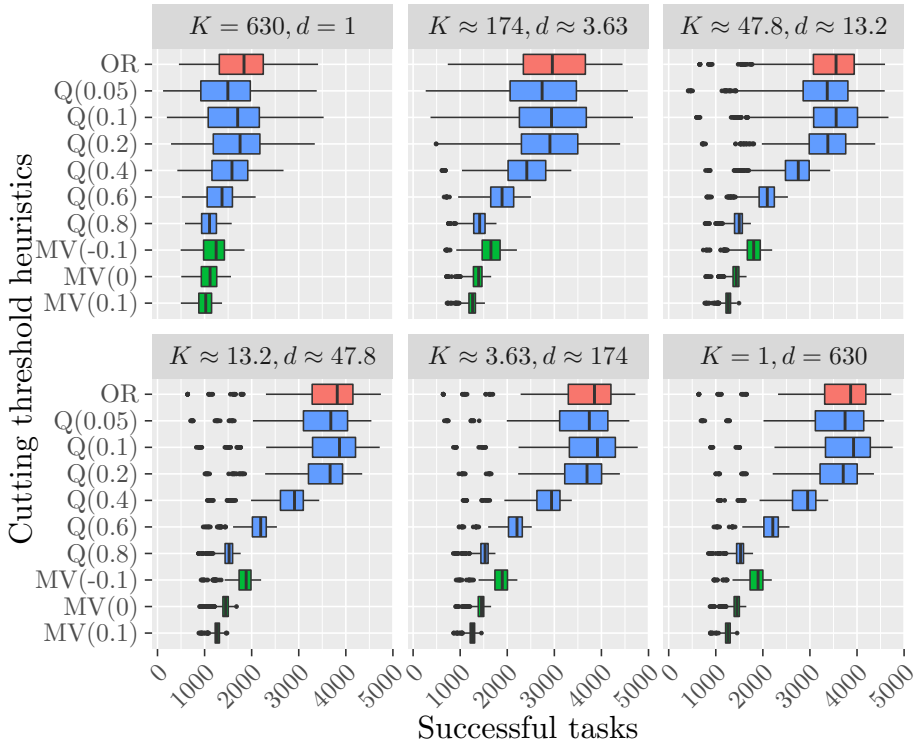


Figure 2: Number of successfully executed tasks of different cutting threshold heuristics with method Greedy. Each of the 100 platforms is generated with $m_j = 10$, $M = 60$, $c_j = s_j$ and is used 10 times with $b = 630$. The values for d and K follows a geometric progression between 1 and $b = 630$. Execution times follow a lognormal distribution with $x_{\text{CV}} = 3$.

method and all other methods except Greedy expands up to a factor three for the median performance. Note that the maximum number of successful tasks increases with x_{CV} because the methods manage to select VMs with the best yield. In particular, it is possible to perform twice as much tasks with $x_{\text{CV}} = 3$ than with $x_{\text{CV}} = 0$ because some VMs have a higher yield. Figure 4 provide more examples with different x_{CV}. We can see the trend more clearly. The effect of the deadline is similar to the effect of x_{CV} even though its cause differs. With the extreme case when K is large (i.e. $K = b = \sum_i c_i$ and $d = 1$), all methods select all VMs, which exhausts the budget when reaching the deadline. The alignment of all boxplot values in the figure for $d = 1$ confirms this effect. Moreover, all methods choose VMs in a predefined order until the sum of c_i of selected VMs reaches K, which means we must choose more VMs with large K. Since VMs are ordered by their yield in the fractional and Greedy methods , the larger the K, the smaller the average yield of chosen VMs. However, with larger deadlines,

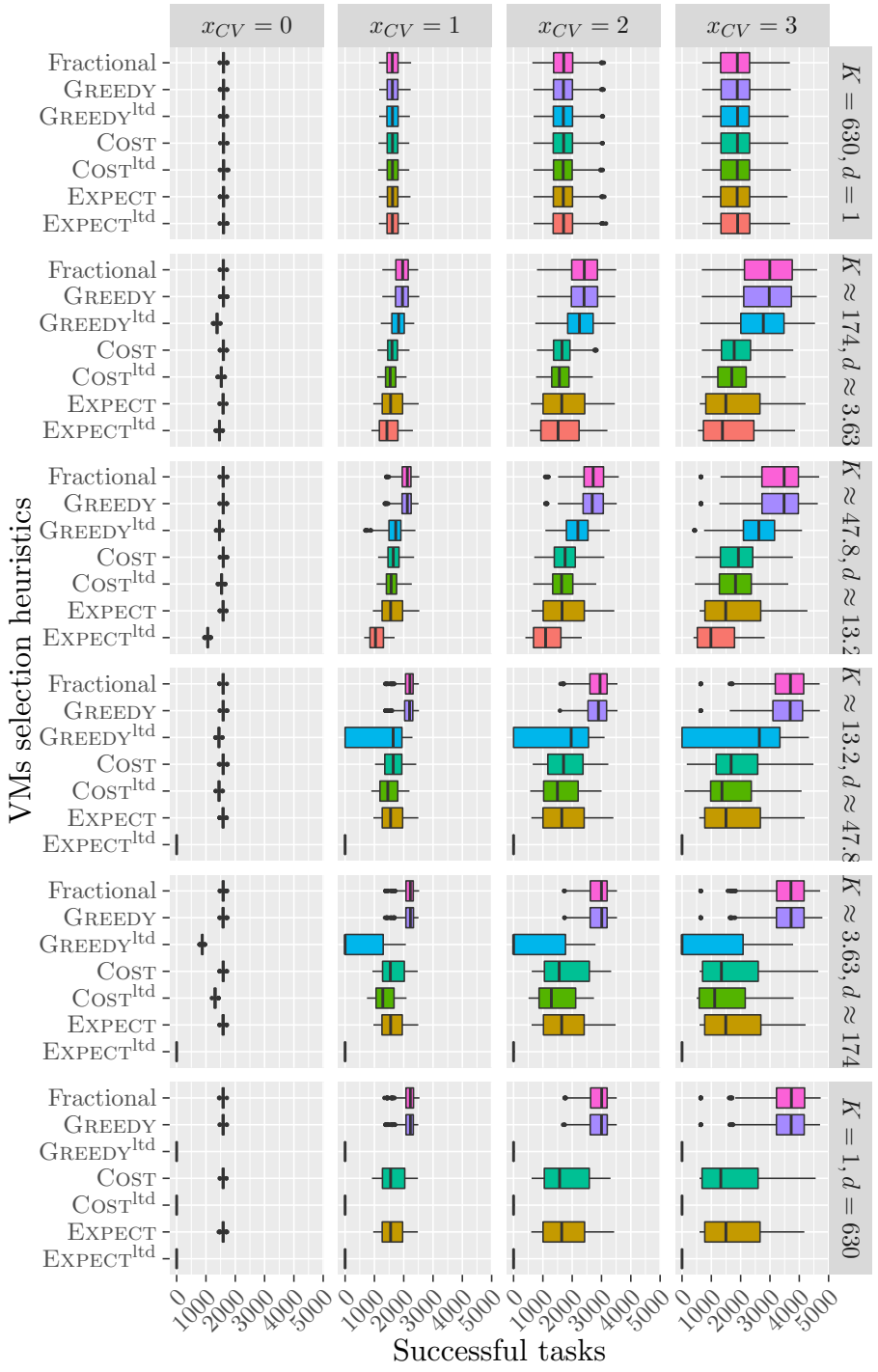


Figure 3: Number of successfully executed tasks of different methods to choose VMs when interrupting tasks with heuristic OptRatio. Platforms are generated as in Figure 2 ($m_j = 10$, $M = 60$, $c_j = s_j$, $b = 630$, and same d and K). Execution times follow a lognormal distribution.

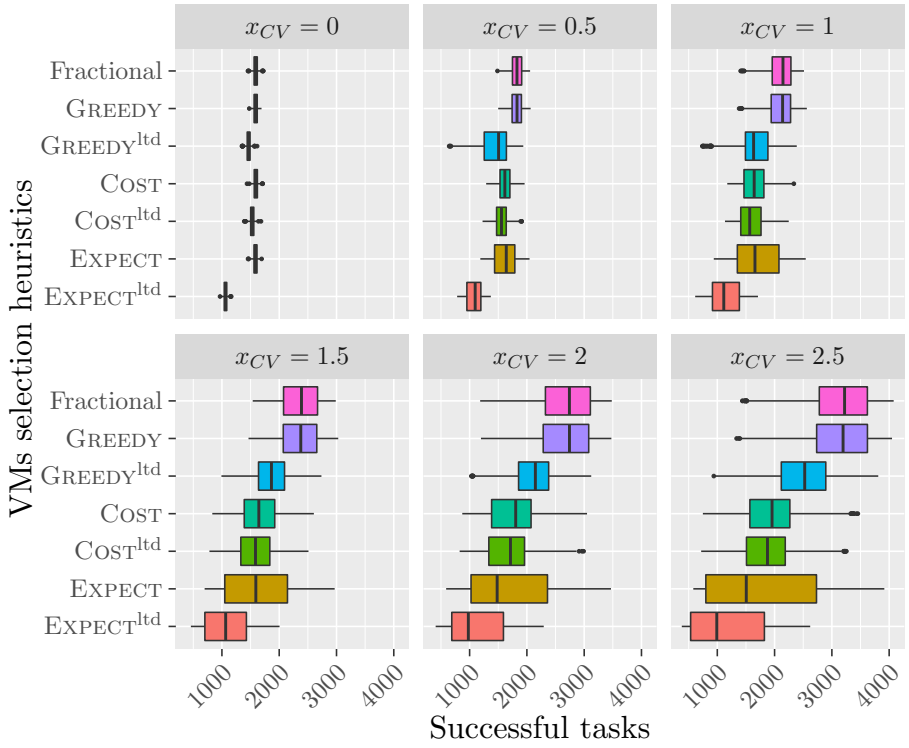


Figure 4: Number of successfully executed tasks of different methods to choose VMs when interrupting tasks with heuristic OptRatio. Each of the 100 platforms is generated with $m_j = 10$, $M = 60$, $c_j = s_j$ and is used 10 times with $b = 630$ and $d \approx 13.2$ ($K \approx 47.8$). Execution times follow lognormal distributions.

the VM choice becomes critical and only Greedy has a gain similar to the fractional method. In other words, the difference between these two methods and others increases as the parallelism K decreases. As the deadline is less constrained, Greedy can select only the VMs with best yield, which improves its performance. However, with $d \geq 47.8$, the improvement gain is less remarkable because the best VMs are all already enrolled. Thus, having a larger deadline that does not impose the selection of inefficient VMs to exhaust the budget before the deadline provides no benefit.

Figure 5 shows the results from platforms with different numbers of machines per type. We can see that there is more difference between Fractional, Greedy and the other methods when the m_j values increase. We have always Fractional and Greedy better than others. Methods Greedy, Cost and Expect are also respectively better than their limited version. We can conclude that the number of machines per type has little influence on the results, thereby confirming the superiority of Greedy.

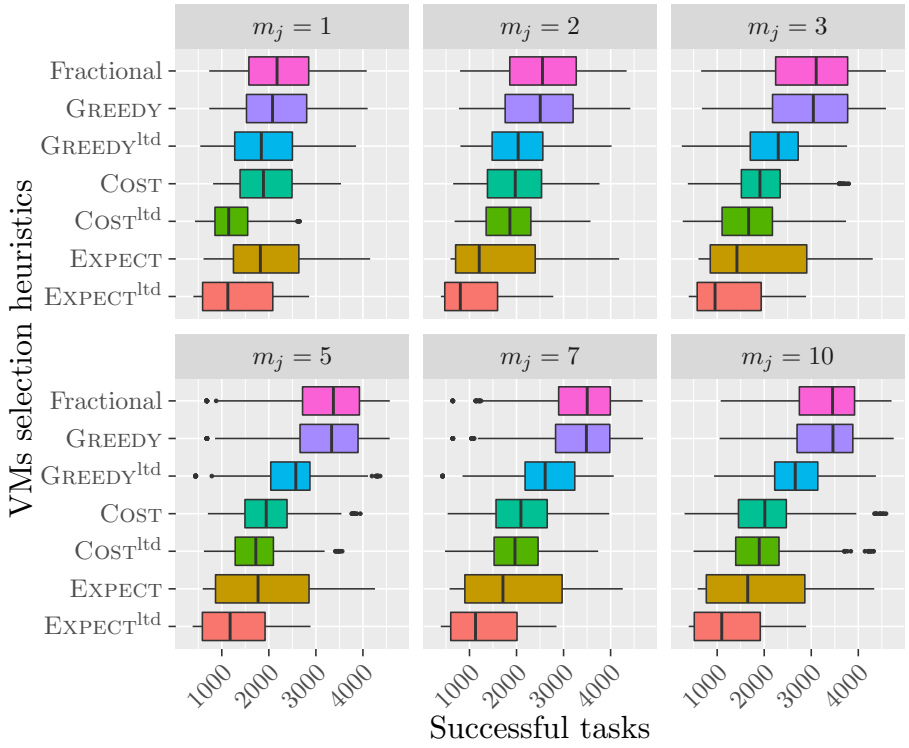


Figure 5: Number of successfully executed tasks of different methods to choose VMs when interrupting tasks with heuristic OPTRATIO. Each of the 100 platforms is generated with $m_j = 1, 2, 3, 5, 7, 10$, $M = 60$, $c_j = s_j$ and is used 10 times with $b = 630$ and $d \approx 13.2$ ($K \approx 47.8$). Execution times follow lognormal distributions with $x_{\text{CV}} = 3$.

Figures 6 and 7 correspond to the same parameters as in Figure 3 except for a different relationship between unit cost and VM speed: we let $c_j = 1.5s_j$ and $c_j = s_j^{1.5}$ respectively. We made this choice because the price paid for a VM may increases faster than its speed, due to the extra cost of memory[3]. In this case, the budget b (which is fixed at $\sum_i c_i$) used in the experiment varies, which impacts d and K too. For the first case (Figure 6), we can see that the result is similar to Figure 3, including the absolute number of successful tasks of each method, rthe elative gain between different methods, and the variation of graphs with x_{CV} and K. On the contrary, in Figure 7, we can see that, for any value of K, GREEDY and COST can have similar gain as the fractional method. But when K decreases (d increases), the relative gain of EXPECT and EXPECT$^{\text{ltd}}$ decrease faster than with other methods. Even the absolute number of successful tasks does not increase. Thus we can

[3] `https://azure.microsoft.com/en-gb/pricing/calculator/#virtual-machines`

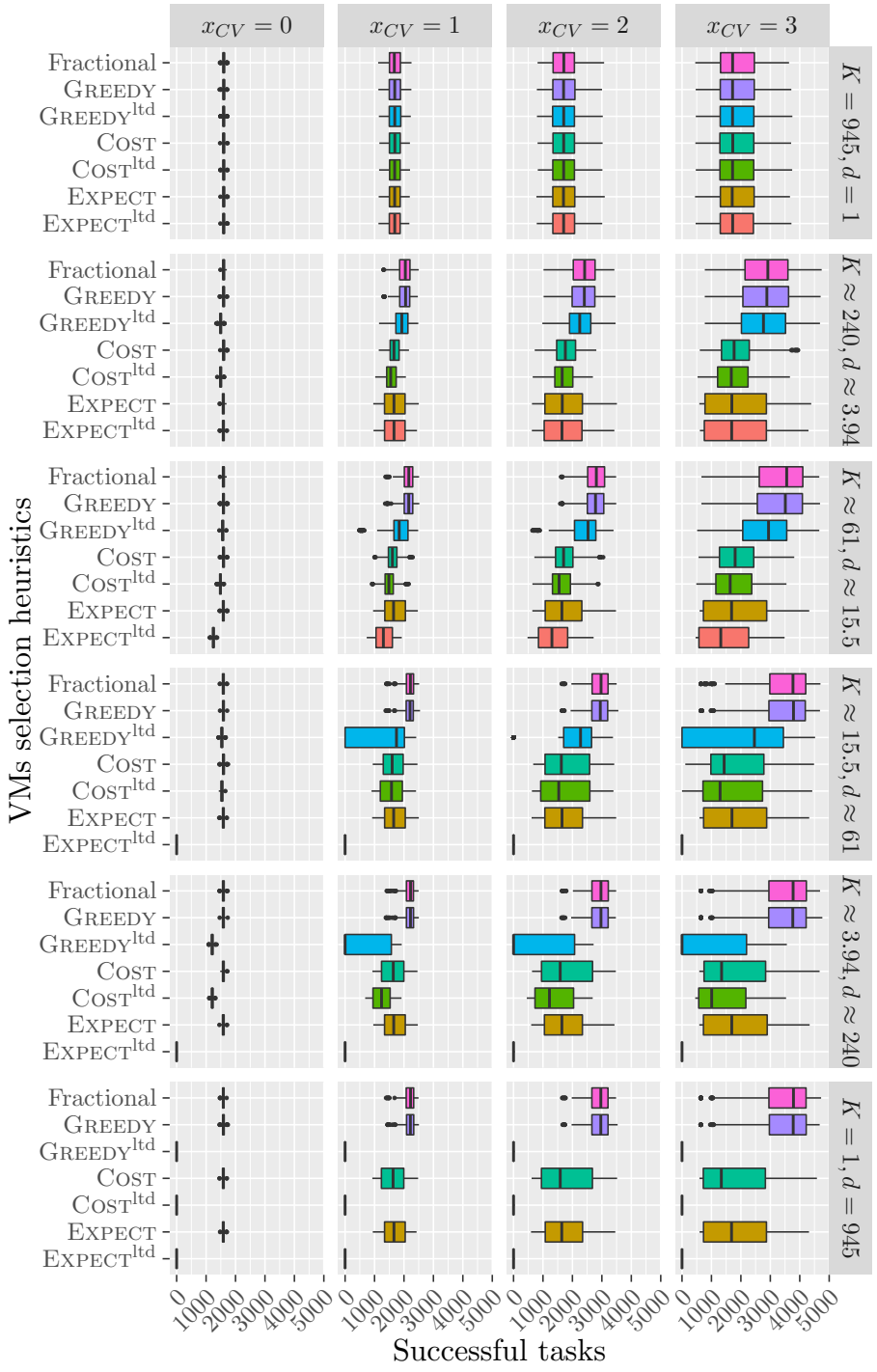


Figure 6: Number of successfully executed tasks of different methods to choose VMs when interrupting tasks with heuristic OPTRATIO. Platforms are generated as in Figure 3 ($m_j = 10$, $M = 60$, and same d and K) except $c_j = 1.5s_j$ and $b = 945$. Execution times follow a lognormal distribution.

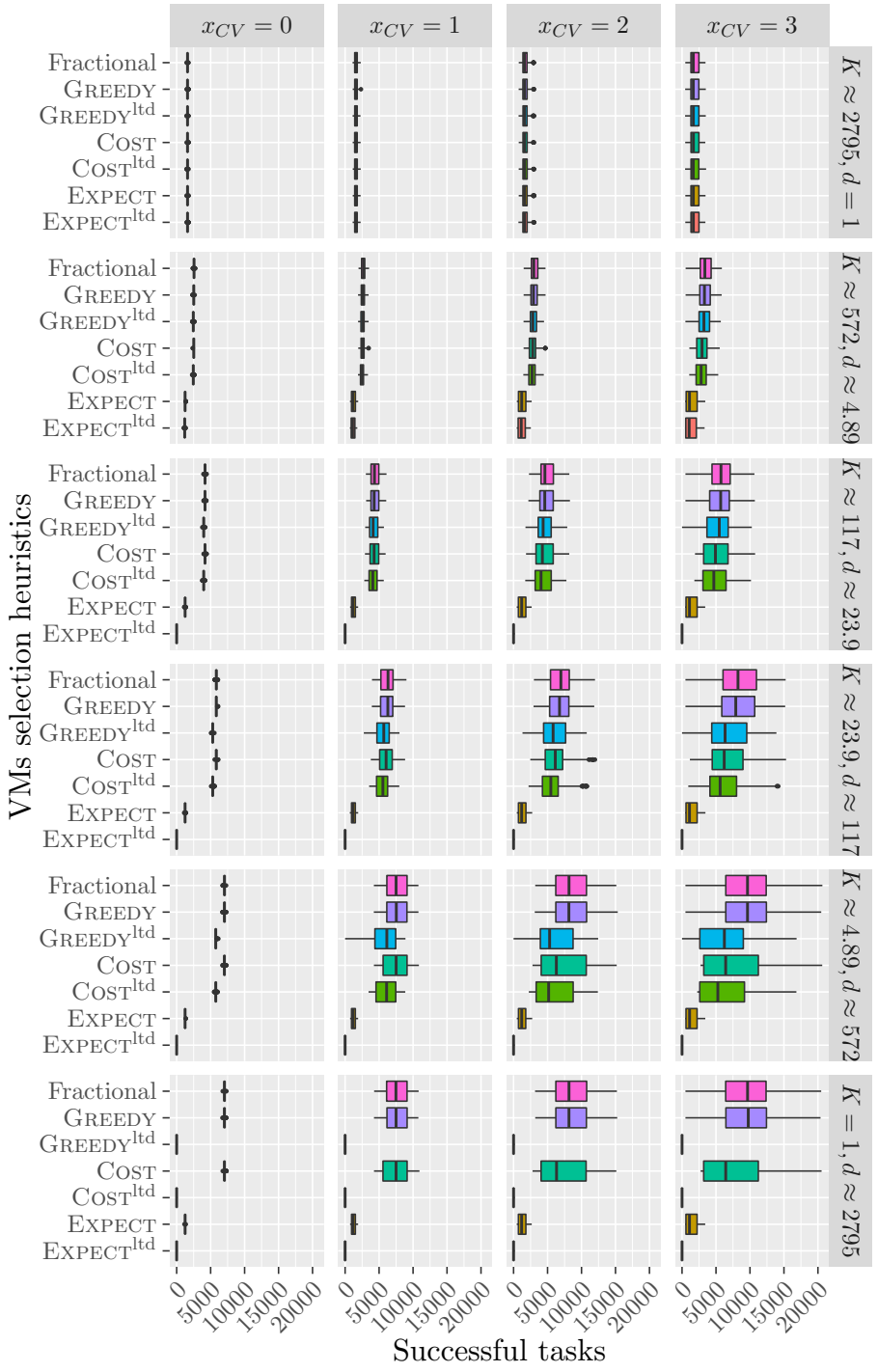


Figure 7: Number of successfully executed tasks of different methods to choose VMs when interrupting tasks with heuristic OPTRATIO. Platforms are generated as in Figure 3 ($m_j = 10$, $M = 60$, and same d and K) except $c_j = s_j^{1.5}$ and $b \approx 2795$. Execution times follow a lognormal distribution.

conclude that, when unit cost and VM speed remain proportional, the results are similar. But when we have a non-proportional relationship between these two parameters ($c_j = s_j^o$ and $o > 1$), EXPECT and EXPECT$^{\text{ltd}}$ have worse results. Thus, it is preferable to use slower (and cheaper) VMs with large deadlines.

Summary. In all figures GREEDY, COST and EXPECT are respectively better than or similar to GREEDY$^{\text{ltd}}$, COST$^{\text{ltd}}$ and EXPECT$^{\text{ltd}}$. The latter three methods even have some zero values. This is because they enroll VMs (in different orders) until the last VM that does not exceed the budget when executed up to the d. Thus, the first VM is abandoned if the budget to execute this VM exceeds b. In this situation, no VM is chosen by the method, and the number of successful tasks is zero. This observation stresses the importance of using the non-limited approaches.

Furthermore, all the above results confirm that GREEDY reaches better performance than other methods, up to a factor three on average.

6 Related work

This work falls under the scope of cloud computing since it targets the execution of sets of independent tasks on a cloud platform under a deadline and a budget constraints. However, because we do not assume to know in advance the execution time of tasks (we are in a *non-clairvoyant* setting), this work is also closely related to the scheduling of bags of tasks. We survey both topics in Sections 6.1 and 6.2. Finally, in Section 6.3, we survey task models that are closely related to our model.

6.1 Cloud computing

There exists a huge literature on cloud computing, and several surveys review this collection of work [4, 38, 39]. Singh and Chana published a recent survey devoted solely to cloud resource provisioning [38], that is, the decision of which resources should be enrolled to perform the computations. Resource provisioning is often a separate phase from resource scheduling. Resource scheduling decides which computations should be processed by each of the enrolled resources and in which order they should be performed.

Resource provisioning and scheduling are key steps to the efficient execution of workflows on cloud platforms. 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 [1,3,6,7,16,21,30,31,42], 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 [29, 37]. Among these articles, Poola et al. [37] 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 [8] 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 [25, 40]. For instance, Tian and Chen [40] consider MapReduce programs and can either minimize the financial cost while matching a deadline or minimize the execution time while enforcing a given budget.

6.2 Bags of tasks

A bag of tasks is an application comprising a set of independent tasks sharing some common characteristics: either all tasks have the same execution time or they are instances coming from a same distribution. Several works devoted to bag-of-tasks processing explicitly target cloud computing [23,36]. Some of them consider the classical clairvoyant model [23] (while [14] targets a non-clairvoyant setting). A group of authors including Oprescu and Kielmann have published several studies focusing on budget-constrained makespan minimization in a non clairvoyant settings [34–36]. They do not assume they know the distribution of execution times but try to learn it on the fly [34,35]. This work differs from ours as these authors do not consider deadlines. For instance, in [36], 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.

Vecchiola et al. [41] 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. [32] 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.

6.3 Task model

Our task model assumes that some tasks may not be executed. This model is very closely related to *imprecise computations* [2, 15, 28], 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: our work then 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 [33], robotic systems [24], speech processing [18], and [27] also cites multimedia processing, planning and artificial intelligence, and database systems. Our task model also corresponds to the overload case of [5] where jobs can be *skipped* or *aborted*. Another, related model, is that of *anytime tasks* [26] 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 [19].

Altogether, the present study appears to be unique because it uses a non-clairvoyant framework and assumes an overall deadline in addition to a budget constraint. Our previous work [9, 11] had the same setting but was limited to identical VMs, while the key problem studied in the paper is the selection of an efficient set of VMs among those available in the target cloud platform.

7 Conclusion

In this paper, we have dealt with the problem of scheduling stochastic tasks on a cloud platform under both deadline and budget constraints. The main difficulty is to select the best subset of VMs so as to maximize the expected number of tasks that are successfully executed. Then on each enrolled VM, we rely on our previous work to compute the time threshold at which to interrupt tasks.

We have assessed the complexity of this optimization problem, showing that it is NP-hard, and also designing Greedy, a greedy algorithm whose performance is proved to be a 2-approximation. Greedy sorts the VMs by non-decreasing yield and then determines the optimal number of VMs to enroll when considering them in this order. On the practical side, an extensive set of experiments shows that Greedy significantly outperforms other approaches based on simple heuristics, and reaches an absolute performance close to the upper bound established in the paper.

This work can be continued along three directions: extending the pricing model to take more complex scenarios into account, and focusing on the modeling of execution times. The first direction consists in considering start-up costs, which would limit the number of enrolled VMs, or non-constant costs that depend on the time and day, or on the load of the cloud platform. For the second direction, multimodal distributions have been advocated to model job, file and object sizes [17]. Similarly to the lognormal distribution, such distributions represent ideal candidates to study the corresponding yield. Finally, the experimental section could be extended to consider instances in which expected execution times can be correlated to the CVs on the VMs as it has been done for cost matrices [10].

Acknowledgements

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] S. Abrishami, M. Naghibzadeh, and D. H. 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.

[2] 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.

[3] 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.

[4] M. U. Bokhari, Q. Makki, and Y. K. Tamandani. A survey on cloud computing. In D. M. V. Aggarwal, V. Bhatnagar, editor, *Big Data Analytics*, volume 654 of *Advances in Intelligent Systems and Computing*. Springer, 2018.

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

[6] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng. Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8):1011 – 1026, 2011.

[7] 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.

[8] Y. Caniou, E. Caron, A. Kong Win Chang, and Y. 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.

[9] L.-C. Canon, A. K. W. Chang, Y. Robert, and F. Vivien. Scheduling independent stochastic tasks under deadline and budget constraints. Extended version. Research Report 9257, INRIA, Feb. 2019.

[10] L.-C. Canon, P.-C. Héam, and L. Philippe. Controlling the correlation of cost matrices to assess scheduling algorithm performance on heterogeneous platforms. *Concurrency and Computation: Practice and Experience*, 29(15):e4185, 2017.

[11] L.-C. Canon, A. Kong Win Chang, Y. Robert, and F. Vivien. Scheduling independent stochastic tasks under deadline and budget constraints. In *SBAC-PAD*. IEEE, 2018.

[12] L.-C. Canon, A. Kong Win Chang, F. Vivien, and Y. Robert. Code for scheduling independent stochastic tasks under deadline and budget constraints, June 2018. `https://doi.org/10.6084/m9.figshare.6463223.v2`.

[13] L.-C. Canon and L. Philippe. On the heterogeneity bias of cost matrices for assessing scheduling algorithms. *IEEE Trans. Par. Dist. Syst.*, 28(6):1675–1688, 2017.

[14] H. Casanova, M. Gallet, and F. 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] 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.

[16] 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.

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

[18] 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.

[19] T. S. Ferguson. *Optimal stopping and applications*. UCLA Press, 2008.

[20] Y. Gao, L.-C. Canon, Y. Robert, and F. Vivien. Code to schedule stochastic tasks on heterogeneous platforms, Feb. 2019.

[21] Y. Gao, Y. 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)*, Sept. 2013.

[22] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

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

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

[25] E. Hwang and K. H. 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.

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

[27] 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.

[28] 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 A. M. van Tilborg and G. M. Koob, editors, *Foundations of Real-Time Computing: Scheduling and Resource Management*, pages 203–249. Springer, 1991.

[29] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, and Y. 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.

[30] 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.

[31] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. *Future Gen. Comp. Syst.*, 48:1–18, 2015.

[32] 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, Oct. 2010.

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

[34] 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, Nov. 2010.

[35] A.-M. Oprescu, T. Kielmann, and H. Leahu. Budget estimation and control for bag-of-tasks scheduling in clouds. *Parallel Processing Letters*, 21(02):219–243, 2011.

[36] 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, Nov. 2012.

[37] 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.

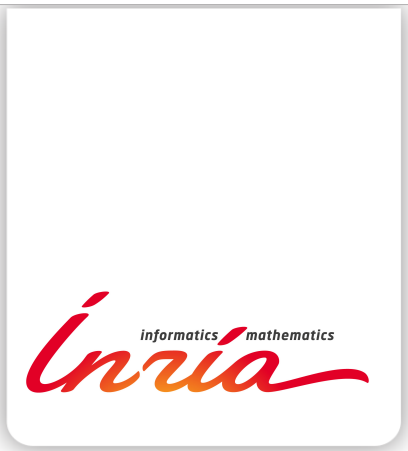
[38] S. Singh and I. Chana. Cloud resource provisioning: survey, status and future research directions. *Knowledge and Information Systems*, 49(3):1005–1069, Dec. 2016.

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

[40] 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.

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

[42] 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.



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