Scheduling independent stochastic tasks on heterogeneous cloud platforms

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

Scheduling independent stochastic tasks on heterogeneous cloud platforms

Yiqin Gao*, Louis-Claude Canon†, Yves Robert*‡, Frédéric Vivien*
*Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France
†FEMTO-ST, Université de Bourgogne Franche-Comté, France
‡University of Tennessee Knoxville, USA

***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 lengths of tasks follow a variety of standard probability distributions (exponential, uniform, half-normal, etc.), which is known beforehand and 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 tend to infinity. Then we introduce several heuristics and compare their performance by running an extensive set of simulations.**

I. Introduction

This paper deals with the following problem: given a cloud platform and a bag of stochastic 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 follows a different probability distribution on each VM type, in order to account for their different performance. For instance, the expectation of the distribution of task durations on a given VM can be inversely proportional to the raw speed of that VM, while the standard deviation can account for the interplay between task profiles and VM parameters, such as memory usage, communication pattern, etc. In the paper, we use an extensive set of widely used distributions, namely exponential, uniform, half-normal, lognormal, gamma, inverse-gamma and Weibull distributions.

This task model assumes that some tasks may not be executed in the end. In fact, there are three cases: (i) some tasks are launched and reach completion, meaning that they are successfully executed: (ii) some tasks are launched but they are interrupted before completion, meaning that their execution has failed; and (iii) some tasks are not launched at all. The objective is to maximize the number of successful tasks, given the deadline and budget constraints. This scheduling problem naturally arises with many applications in the context of information retrieval (see Section II for a detailed discussion). Informally, the goal is to extract as much information as possible, by launching analysis tasks whose execution time strongly depends upon the nature of the data sample being processed. A typical example is a set of image files, whose processing times heavily depend upon the elements that are present (or not) within each image. Not all data samples must be processed, but the larger the number of data samples successfully processed, the more accurate the analysis.

Furthermore, this task model is closely related to *imprecise computations* [2], [13], [27], 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 she must select which ones to execute. Our work perfectly corresponds to optimizing the processing of the optional parts. Among domains where tasks may have optional parts (or where some tasks may be entirely optional), one can cite recognition and mining applications [32], robotic systems [23], speech processing [16]; and [26] also cites multimedia processing, planning, artificial intelligence, and database systems. In these applications, the processing times of the optional parts are heavily data-dependent, hence the need to estimate them via a probability distribution.

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, because: (i) the time and 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], 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 cutting threshold is shown to maximize the expected *success rate* on the VM, i.e., the average number of tasks

successfully executed per time unit. This cutting threshold depends upon the probability distribution of task execution times and is computed numerically.

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. 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 we also show 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 makes it the heuristic of choice for the target optimization problem.

The main contributions of this work are the following:

- We provide several theoretical results (NP-completeness, approximation algorithm GREEDY and performance lower bound) for the problem instance with large budget and deadline. These results show the difficulty of the optimization problem under study, and lay the foundations for its analysis;
- We compare the performance of GREEDY to that of several heuristics for the general problem with arbitrary deadline and budget values, and for all the probability distributions mentioned above. Not only GREEDY is superior to the other heuristics, but its performance is very close to the lower bound on most instances. Altogether, GREEDY provides a robust approach to the problem.

The rest of the paper is organized as follows. Section II surveys related work. We detail the framework and objective in Section III, and recall prior strategies for interrupting tasks on a given VM in Section IV. We provide complexity results (NP-completeness and 2-approximation algorithm) in Section V. We compare these heuristics in Section VI, assessing their performance for an extensive set of simulation parameters. Finally, we provide concluding remarks and directions for future work in Section VII.

II. 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 deadline and budget constraints. However, because we do not assume to know in advance the exact execution time of tasks (we are in a *non-clairvoyant* setting and know only its distribution), this work is also closely related to the scheduling of bags of tasks. We survey both topics in Sections II-A and II-B. Finally, in Section II-C, we survey task models that are closely related to our model.

A. Cloud computing

There exists a huge literature on cloud computing, and several surveys review this collection of work [4], [37], [38]. Singh and Chana published a recent survey devoted solely to cloud resource provisioning [37], 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], [14], [20], [29], [30], [41], 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 [28], [36]. Among these articles, Poola et al. [36] 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 [24], [39]. For instance, Tian and Chen [39] 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.

B. 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 [22], [35]. Some of them consider the classical clairvoyant model [22] (while [12] 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 [33]–[35]. They do not assume they know the distribution of execution times but try to learn it on the fly [33], [34].

This work differs from ours as these authors do not consider deadlines. For instance, in [35], 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. [40] consider a single application comprising independent tasks with deadlines but without any budget constraints. In their model, tasks are supposed to have different execution times but they only consider the average execution time of tasks rather than its probability distribution (this is left for future work); moreover, they do not report on the amount of deadline violations. Mao et al. [31] 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.

C. Task model

Our task model assumes that some tasks may not be executed. This model is very closely related to *imprecise computations* [2], [13], [27]. Furthermore, this 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* [25] 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 [17].

Altogether, the present study appears to be unique because it uses a semi non-clairvoyant framework and assumes an overall deadline in addition to a budget constraint. Our previous work [9] 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.

III. Problem definition

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

A. Platform and tasks

We aim at scheduling a set of independent stochastic tasks all available at time zero 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 unit of time 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 known beforehand and 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 II), 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 VI), 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 report results for a variety of standard probability distributions (exponential, uniform, half-normal, lognormal, gamma, inverse-gamma and Weibull). 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 $\sum_{j=1}^{k} m_j = M$.

Table I
SUMMARY OF MAIN NOTATIONS.

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

B. 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 before the deadline is

reached or the totality of the budget is spent. More precisely, the optimization problem $\text{OPT}(\mathcal{P}, b, d)$ 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.

IV. Cutting threshold

In this section, we discuss scheduling strategies that decide to interrupt a task when its execution time reaches a given threshold. Consider a given VM and a continuous distribution $\mathcal{D}$ for execution times on that VM, with expected value μ and standard deviation σ. We start with some classical strategies:

- $\text{MeanVariance}(x)$ 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).
- $\text{Quantile}(x)$ is the family of heuristics that interrupt a task when its execution time reaches the x-quantile of the distribution $\mathcal{D}_i$ with $0 \leq x \leq 1$.
- Finally, NeverInterrupt is the baseline heuristic that never interrupts tasks; more precisely, to avoid outliers, NeverInterrupt interrupts a task when its execution reaches 100 times the mean value of the distribution.

Our previous work [9] introduced the OptRatio heuristic which works as follows: Let $F(x)$ be the cumulative distribution function and $f(x)$ its probability density function of $\mathcal{D}$. OptRatio interrupts all (unsuccessful) tasks at time

$$\ell^{max} = \arg\max_l \mathcal{S}(l) \text{ where } \mathcal{S}(l) = \frac{F(l)}{\int_0^l x f(x)dx + l(1-F(l))}. \quad (1)$$

The idea behind OptRatio is that it maximizes (asymptotically) the ratio $\mathcal{S}(l)$ 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). OptRatio has been shown to perform very well for a wide range of budget and deadline values [9].

For most distributions, we cannot compute ℓ^{max} analytically, but we provide a program [10] to compute it numerically. For example take a lognormal(α, β) distribution: when $\mu = 1$ and $\sigma = 3$ we find $\ell^{max} \approx 0.1$. We prove the following characterization of ℓ^{max} for exponential and uniform distributions (see the extended version [19] for the proof):

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).*

V. Complexity results

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

A. Problem instance with $b = Kd$

Consider a given VM $\text{VM}_i \in \mathcal{P}$. Given the distribution $\mathcal{D}_i$ of task execution times on VM_i, we choose a cutting threshold ℓ_i^{cut} at which to interrupt tasks, using any of methods in Section IV (for instance we take $\ell_i^{cut} = \ell_i^{max}$, the value computed for OptRatio). We then derive the (asymptotic) success rate $\mathcal{S}_i$ (average number of successful tasks per time unit) and the yield $\mathcal{Y}_i = \frac{\mathcal{S}_i}{c_i}$ (average number of successful tasks per cost unit), where c_i is the unit cost of VM_i. The asymptotic behavior of VM_i is characterized by these two parameters. With several VMs, if there is no deadline, the best solution is to use a single VM, namely the one with highest yield $\mathcal{Y}_i$. 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 $K \geq 1$, 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, hence the potential parallelism that can be achieved.

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

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

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

$$\mathcal{Y}^{tot} = \frac{\sum_{i \in Q} \mathcal{S}_i}{\max(K, \sum_{i \in Q} c_i)}. \quad (2)$$

We are ready to define the asymptotic optimization problem (when b tends to infinity) 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* (2) *is maximized.*

B. NP-completeness

In this section, we show that the decision problem associated to the asymptotic problem 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, whose yield can be computed in linear time. For the completeness, we make a reduction from SUBSETSUM, a well-known NP-complete problem [21]. Consider an instance $\mathcal{I}_1$ of SUBSETSUM: given n positive integers a_1, a_2, ..., 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

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

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. □

C. Greedy heuristic

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

Table II
EXAMPLE OF PLATFORM $\mathcal{P}$ ($M = 5$).

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	$\mathcal{S}_5 = 4$	$c_4 = 4$	$\mathcal{Y}_5 = 1$

In Table II, 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 II, 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

$$\mathcal{Y}^{tot,frac} = \frac{\sum_{j \in \mathcal{P}} \mathcal{S}_j t_j}{b} \tag{3}$$

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* (3) *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^{opt} 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 $t_i^{opt} > t_i$. 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 $t_j^{opt} < t_j$. 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 in the optimal solution, we re-distribute the amount of budget $\beta = \min\{(t_j - t_j^{max})c_j, (t_i^{max} - t_i)c_i\}$ from VM$_i$ to VM$_j$. The first term 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 nonnegative value $\frac{\beta(\mathcal{Y}_j - \mathcal{Y}_i)}{b}$. If $\mathcal{Y}_i < \mathcal{Y}_j$, this contradicts the optimality. Otherwise, each VM$_k$, where $j \leq k \leq i$ has same yield (because of the ordering method of the greedy algorithm); then the optimal solution and the greedy algorithm have same global yield. 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

$$\mathcal{Y}^{opt\text{-}frac} = \frac{\mathcal{S}_{i^*-1}^{tot}}{K} + \left(1 - \frac{\mathcal{C}_{i^*-1}^{tot}}{K}\right) \frac{\mathcal{S}_{i^*}}{c_{i^*}}. \tag{4}$$

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

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

Similarly to the proof of Proposition 2, we can easily prove by induction that $\frac{\mathcal{S}_{i^*}^{tot}}{\mathcal{C}_{i^*}^{tot}} \geq \frac{\mathcal{S}_{i^*}}{c_{i^*}}\cdot$. As $0 \leq 1 - \frac{\mathcal{C}_{i^*-1}^{tot}}{K} \leq 1$, we have $\frac{\mathcal{S}_{i^*}^{tot}}{\mathcal{C}_{i^*}^{tot}} \geq \left(1 - \frac{\mathcal{C}_{i^*-1}^{tot}}{K}\right)\frac{\mathcal{S}_{i^*}}{c_{i^*}}\cdot$. We derive:

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

□

VI. 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 algorithms are implemented in R and the related code, data and analysis are publicly available in [18].

A. VMs selection heuristics

As we have different types and numbers of VMs, we aim at finding 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 compare several methods for choosing VMs. They differ 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. This choice has objective to show the improvement of the last step on results. Here are the three orderings:

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

In addition, we assess the absolute performance of each method by comparing with FRACTIONAL, which is the yield achieved by the solution to the fractional problem FRACOPT-HETERO (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.

B. Parameters

In the following experiments, all platforms are composed of 10 VMs from each of six VM types, for a total of $M = 60$ VMs. In other words, $k = 6$ and $m_j = 10$ for $1 \leq j \leq 10$.

Each VM type is characterized by a unit cost and 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 is proportional to its average speed: $c_j = s_j$.

Other scenarios, with different values of m_j, and with unit costs increasing faster than average speeds, are available in the extended version [19].

The second VM characteristic is the distribution of the execution times, which follow standard probability distributions. The heterogeneity of a scheduling problem instance has several meanings (for instance, both tasks and machines heterogeneity are studied in [11]). 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 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}. Of course for exponential and half-normal distributions, which have a single parameter, the standard deviation is given when choosing the mean, so this discussion only applies to the two-parameter distributions (lognormal, uniform, gamma, inverse-gamma and Weibull). Altogether, the expected value μ_j and standard deviation σ_j on VM_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 $[3 - x_{\text{CV}}, 3 + x_{\text{CV}}]$. We use $0 \leq x_{\text{CV}} \leq 3$ in the experiments.

Finally, we fix the budget at $b = \beta \sum_{j=1}^{k} m_j c_j = \beta \times 630$[3], where $\beta \in \{0.01, 0.05, 0.1, 1, 2, 5, 8, 10\}$. For each budget value, we vary the deadline as $d = \beta 630^{\frac{i}{5}}$ (hence $K = 630^{1-\frac{i}{5}}$) for $0 \leq i \leq 5$. This leads to 6 deadline values following a geometric progression between two extreme cases $d = b$ and $d = \beta$. The first case is when the deadline is sufficiently large to exhaust the entire budget by selecting any single VM. The second case is when the deadline is so tight that all VMs must be used to exhaust the budget. Altogether, we have 8 budget and 6 deadline values, hence 48 configurations. For each configuration, each strategy is run 10 times on 100 randomly drawn platform instances (the mean of the distribution is fixed, and we draw the value of the standard deviation as discussed above, except for exponential and half-normal distributions).

The numbers of successes are 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. Due to lack of space, we cannot present all results, which are available in the extended version [19]. Instead, we start with a summary table covering all distributions, and then focus on lognormal distributions.

Table III
PERFORMANCE RATIO OF ALL ORDERINGS OVER FRACTIONAL.

Ordering	Mean	Median	Q10%	Q90%
GREEDY	0.9977	0.9994	0.9668	1.0272
GREEDY$^{\text{ltd}}$	0.6047	0.8385	0	0.9998
COST	0.6943	0.9507	0.0522	1.0208
COST$^{\text{ltd}}$	0.5587	0.7634	0	1.0016
EXPECT	0.7766	0.9717	0.1556	1.0124
EXPECT$^{\text{ltd}}$	0.3642	0	0	0.9973

C. Result synthesis for all distributions

In Table III, OPTRATIO is chosen as cutting threshold heuristic on each VM. We use $b = 630$ (hence $\beta = 1$) and $x_{\text{CV}} = 3$. For each distribution, we have 6 values of d. For exponential and half-normal, the standard deviation is given when we select the mean, so we run only one platform configuration. For the other five probability distributions (lognomal, uniform, gamma, inverse-gamma and Weibull), we draw 100 platform configurations, as mentioned above. Each configuration is run 10 times, which leads to a total of 30,120 experiments. Now, for each heuristic, we proceed as follows: for each experiment, we record the ratio of the number of successful tasks achieved by the VMs selection heuristic over the number of successful tasks achieved by FRACTIONAL; this leads to the statistical values reported in Table III: mean, median, 10% Quantile and 90% Quantile. Table III is fairly representative of the many more experiments available in [19].

Table III shows that GREEDY performs very well overall. Its ratio is close to 1, not only for the mean value, but even for the 10% quantile. In other words, GREEDY has a performance close to that of FRACTIONAL; it also consistently outperforms all the other VMs selection heuristics.

For each heuristic, the non-limited version is always much better than the limited one. Because limited versions readily

[1] https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls

[2] https://cloud.google.com/compute/pricing

[3] 630 represents the budget required to enroll all 60 VMs for one time unit.

discard each VM for which there is not enough budget to run until the deadline, they are at risk of wasting a big fraction of the budget and then produce a bad, even null result. Indeed, there is a large difference between the mean and median values for the limited versions, showing that there are many results close to 0. Results for the 90% quantile are good for all heuristics, and even larger than 1, while FRACTIONAL represents an asymptotic upper bound. Here is the explanation: a sixth of experiments are for $d = 1$. In this case, all the heuristics enroll all the 60 VMs to execute tasks. As the execution time of a task is randomly drawn, some heuristics can have a better result than FRACTIONAL.

D. Lognormal distribution

In this section, we focus on lognormal distributions and further study the impact of several parameters. A lognormal distribution is a natural candidate to model task execution times, because it has been advocated to model file sizes [15], and 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.

For a lognormal(α, β) distribution, the density function is $f(x) = \frac{e^{-\frac{(\log(x)-\alpha)^2}{2\beta^2}}}{x\beta\sqrt{2\pi}}$ for $x \in [0, \infty)$, the mean is $\mu = e^{\alpha+\frac{\beta^2}{2}}$ and the standard deviation is $\sigma = e^{\alpha+\frac{\beta^2}{2}}\sqrt{e^{\beta^2}-1}$. To ensure a given expected value μ and standard deviation σ, we set $\alpha = \log(\mu) - \log(\sigma^2/\mu^2+1)/2$ and $\beta = \sqrt{\log(\sigma^2/\mu^2+1)}$.

1) Cutting threshold heuristics: First, we compare the performance of the different cutting threshold heuristics in Figure 1. We report results for $b = 630$ and the 6 corresponding deadline values. We can find that, for lognormal distribution, OPTRATIO, QUANTILE (0.1) and QUANTILE (0.2) have usually better results 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. The results confirm the observations made with homogeneous machines [9]: OPTRATIO achieves the best success rate, and is significantly better than the baseline NEVERINTERRUPT. This leads to choose OPTRATIO as the cutting threshold heuristic in all the following experiments.

2) Varying budget and deadline values: Figure 2 reports results for the 48 (b, d) pairs. We make several observations.

First, when K is fixed, multiplying b and d by a value $\beta > 1$ only changes the absolute value of the result (there is a proportional relationship between β and the number of successful tasks), but the global outcome remains the same: the same machines are chosen, and the ratio of the results of each heuristic over FRACTIONAL is not modified. This shows that, for a lognormal distribution with μ and σ chosen as in our experiment, $\beta = 1$ is enough to simulate a problem instance with large b and d values. In the following experiment, we keep $b = 630$ and vary x_{CV}[4].

[4]We need larger values of b to reach steady-state for exponential and half-normal distributions, see [19].

Second, in Figure 2, we see the impact of the deadline constraint by varying both d and K with a constant b (in each column of the figure). With the extreme case when K is large (i.e., $K = \sum_i c_i$), all methods select all VMs, which exhausts the budget when reaching the deadline. The alignment of all boxplot values in the figure for $d = \beta$ 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 values of K. As VMs are ordered by their yield in the FRACTIONAL method, the larger the K, the smaller the average yield of chosen VMs. However, with larger deadlines, the VM choice becomes critical and only GREEDY has a gain similar to FRACTIONAL. In other words, the difference between these two latter methods and the other ones increases as the parallelism K decreases. As the deadline is less constrained, GREEDY can select only the VMs with best yield. With $K \leq 13.2$, the gain is less remarkable because the best VMs are all already enrolled. Only small deadlines impose the selection of inefficient VMs to exhaust the budget before the deadline. Thus, having larger deadlines provides little benefit.

Third, in all instances, GREEDY, COST and EXPECT are respectively better than or similar to GREEDY$^{\text{ltd}}$, COST$^{\text{ltd}}$ and EXPECT$^{\text{ltd}}$. These last three methods even have some zero values. As explained in Section VI-C, this is because these methods enroll VMs (in different orders) until the last VM that does not exceed the budget when executed up to the deadline d. Thus, the first VM is abandoned if the budget to execute this VM exceeds b. In this scenario, no VM is chosen by the method, and the number of successful tasks is zero.

Fourth, we observe that GREEDY remains the most efficient resource selection heuristic even for small values of the budget (when $\beta < 1$). This is good news, because we had proven the asymptotic efficiency of GREEDY but needed to check that it maintained its superiority for the whole range of budget and deadline values (even though the number of successes is no longer proportional to the budget for smaller values).

3) Impact of variability heterogeneity: Figure 3 demonstrates the dependence between the level of variability heterogeneity controlled by x_{CV} and the performance disparity between the resource selection heuristics. When $x_{\text{CV}} = 0$, all VMs have the same yield as they have the same ratio CV, thus all heuristics are similar. As x_{CV} increases, the difference between FRACTIONAL 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}, especially for FRACTIONAL and GREEDY heuristics, because the methods manage to select VMs with the best yield. In particular, it is possible to perform twice as much tasks with $x_{\text{CV}} = 2.5$ than with $x_{\text{CV}} = 0$ because some VMs in a platform configuration can have a higher yield.

4) Summary: All the above results confirm that GREEDY reaches better performance than the other resource selection heuristics, up to a factor three on average. As previously mentioned, many additional results are available in the extended version [19].

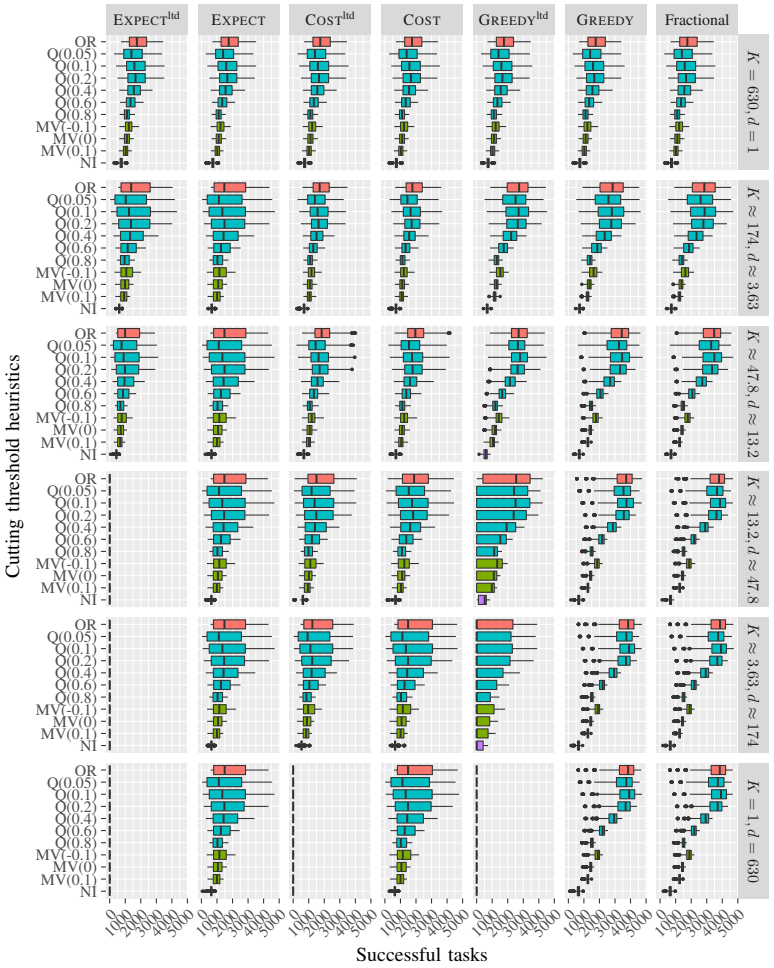

Figure 1. Number of successfully executed tasks for different resource selection and cutting threshold heuristics, with $m_j = 10$, $M = 60$, $c_j = s_j$, $b = 630$. Execution times follow a lognormal distribution with $x_{\text{CV}} = 3$.

VII. CONCLUSION

In this paper, we have dealt with the problem of scheduling stochastic tasks on a cloud platform under both deadline and budget constraints. On each enrolled VM, we use several cutting threshold heuristics to decide when to interrupt tasks. The main difficulty is to select the best subset of VMs so as to maximize the expected number of tasks that are successfully executed. We have assessed the complexity of this resource selection 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, we have conducted an extensive set of experiments, with several standard probability distributions for task execution times. These experiments show 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 main directions, to extend the pricing model with more complex scenarios, and to better account for 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 upon the time and day, or uoon the load of the cloud platform. For the second direction, multimodal distributions have been advocated to model job, file and object sizes [15]. Similarly to the lognormal distribution, such distributions represent ideal candidates to study the corresponding yield. Finally, in a fully non-clairvoyant version of the problem, the distribution of job execution times must be learned on the fly, by sampling observed runtimes of the tasks in order to update a statistical model.

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).

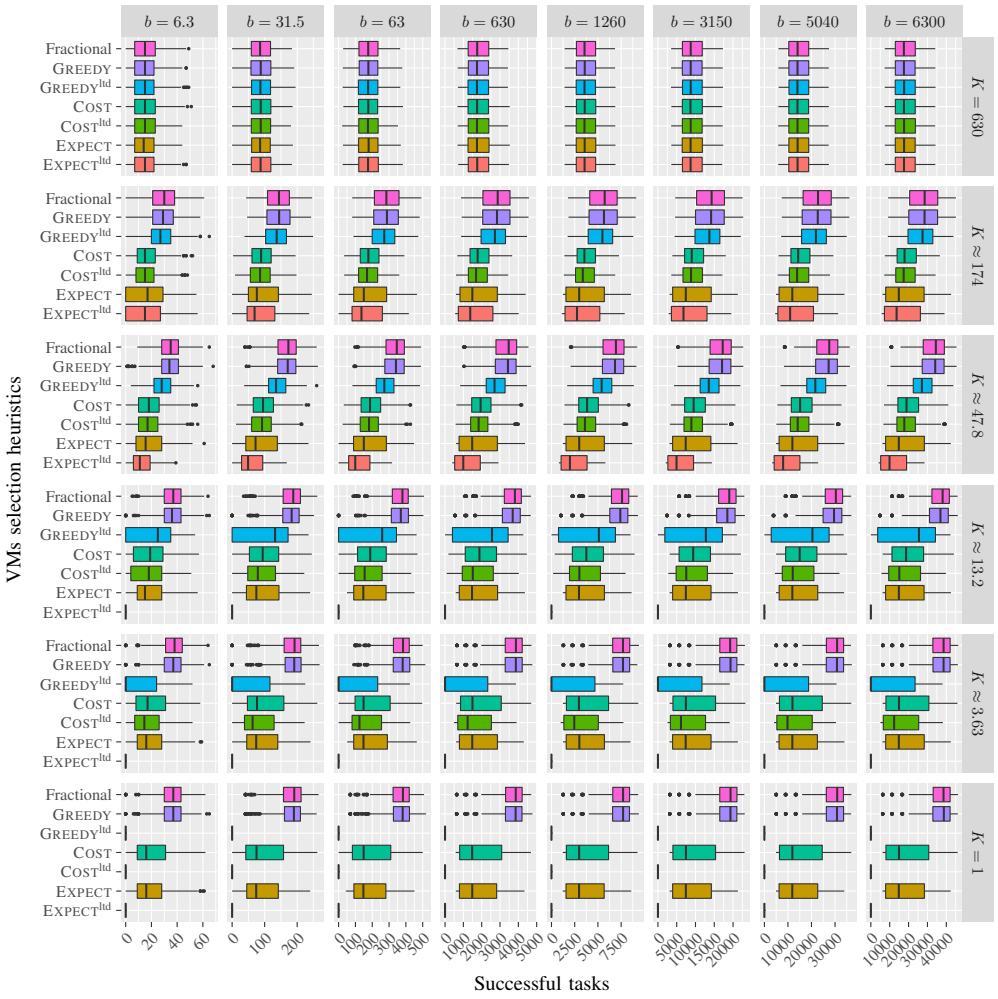

Figure 2. Number of successfully executed tasks for resource selection heuristics with OPTRATIO, $m_j = 10$, $M = 60$, $c_j = s_j$. Execution times follow a lognormal distribution with $x_{\text{CV}} = 3$.

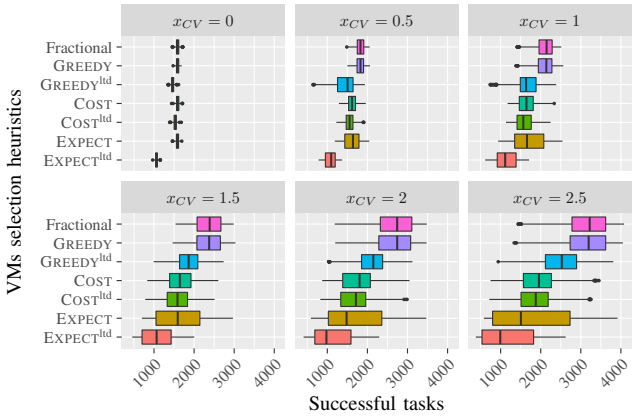

Figure 3. Number of successfully executed tasks for resource selection heuristics with OPTRATIO, $m_j = 10$, $M = 60$, $c_j = s_j$, $b = 630$ and $d \approx 13.2$. Execution times follow a lognormal distribution with x_{CV} varying.

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. Kong Win Chang, Y. Robert, and F. Vivien. Scheduling independent stochastic tasks under deadline and budget constraints. In *SBAC-PAD*. IEEE, 2018.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

[18] Y. Gao, L.-C. Canon, Y. Robert, and F. Vivien. Code to schedule stochastic tasks on heterogeneous platforms, Feb. 2019. https://figshare.com/articles/Code_to_schedule_stochastic_tasks_on_heterogeneous_platforms/7777046/3.

[19] Y. Gao, L.-C. Canon, Y. Robert, and F. Vivien. Scheduling independent stochastic tasks on heterogeneous cloud platforms. Research Report 9275, INRIA, May 2019.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

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

[39] 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.

[40] 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.

[41] 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.