Budget-aware scheduling algorithms for scientific workflows with stochastic task weights on heterogeneous IaaS Cloud platforms

Yves Caniou, Eddy Caron, Aurélie Kong Win Chang, Yves Robert

Budget-aware scheduling algorithms for scientific workflows with stochastic task weights on heterogeneous IaaS Cloud platforms

Yves Caniou* Eddy Caron*, Aurélie Kong Win Chang*, Yves Robert*†
*ENS Lyon, France
†University of Tennessee, Knoxville, TN, USA
{yves.caniou|eddy.caron|aurelie.kong-win-chang|yves.robert}@ens-lyon.fr

***Abstract*—This paper introduces several budget-aware algorithms to deploy scientific workflows on IaaS Cloud platforms, where users can request Virtual Machines (VMs) of different types, each with specific cost and speed parameters. We use a realistic application/platform model with stochastic task weights, and VMs communicating through a datacenter. We extend two well-known algorithms, MIN-MIN and HEFT, and make scheduling decisions based upon machine availability *and* available budget. During the mapping process, the budget-aware algorithms make conservative assumptions to avoid exceeding the initial budget; we further improve our results with refined versions that aim at re-scheduling some tasks onto faster VMs, thereby spending any budget fraction leftover by the first allocation. These refined variants are much more time-consuming than the former algorithms, so there is a trade-off to find in terms of scalability. We report an extensive set of simulations with workflows from the Pegasus benchmark suite. Most of the time our budget-aware algorithms succeed in achieving efficient makespans while enforcing the given budget, despite (i) the uncertainty in task weights and (ii) the heterogeneity of VMs in both cost and speed values.**

I. INTRODUCTION

IaaS (Infrastructure as a Service) Cloud platforms provide a convenient service to many users. Many vendors provide commercial offers with various characteristics and price policies. In particular, a large choice of VM (Virtual Machine) types is usually provided, that ranges from slow-but-cheap to powerful-to-expensive devices. When deploying a scientific workflow on an IaaS Cloud, the user is faced with a difficult decision: which VM type to select for which task? How many VMs to rent? The heterogeneity of VMs applies to both cost and speed values, and these values are not necessarily proportional. The mapping decisions clearly depend upon the budget allocated to execute the workflow, and are best taken when some knowledge on the task profiles in the workflow is available. The standard practice is to run a classical scheduling algorithm, whether MIN-MIN [6], [14] or HEFT [24], with a VM type selected arbitrarily, and to hope for the best, *i.e.,* that the budget will not be exceeded at the end. To remedy such an inefficient approach, this paper introduces several budget-aware algorithms to deploy scientific workflows on IaaS Clouds. The main idea is to revisit well-known algorithms such as MIN-MIN and HEFT and to make a decision for each task to be scheduled based upon both machine availability *and* remaining budget.

While several cost-aware algorithms have been introduced in the literature (see Section II for an overview), this paper makes new contributions along the following lines:

- A realistic application model, with stochastic task weights;
- A detailed yet tractable platform model, with a datacenter and multiple VM categories;
- Budget-aware algorithms that extend MIN-MIN and HEFT, two widely-used list-scheduling algorithms for heterogeneous platforms;
- Refined (but more costly) variants that squeeze the most of any leftover budget to further decrease total execution time. The refined versions aim at exploiting the opportunity to re-schedule some tasks onto faster VMs, thereby spending any budget fraction leftover by the first allocation. These refined variants are much more time-consuming than the former algorithms, so there is a trade-off to find in terms of scalability.

The rest of the paper is organized as follows. Section II surveys related work. We introduce the performance model in Section III. We describe budget-aware scheduling algorithms in Section IV: Section IV-A presents the extensions to MIN-MIN and HEFT, while Section IV-B provides the refined versions. Section V is devoted to assessing their performance through extensive simulations, including comparisons of two previous budget-aware algorithms, namely BDT [3] and CG/CG+ [25]. Finally, we provide concluding remarks and directions for future work in Section VI.

II. RELATED WORK

Many scientific applications from various disciplines are structured as workflows [4]. Informally, a workflow can be seen as the composition of a set of basic operations that have to be performed on a given input data set to produce the expected scientific result. The development of complex middleware with workflow engines [11], [12], [9] has automated workflow management. IaaS Clouds raised a lot of interest recently, thanks to an elastic resource allocation and pay-as-you-go billing model. In a Cloud environment, there exist many solutions for scheduling workflows [18], [23], some of which include data management strategies [26]. Also [20] introduced two auto-scaling mechanisms to solve the resource allocation problem for unpredicted workflow jobs in a cost-efficient way. [27] introduced a workflow scheduling in Clouds solutions with

security and cost considerations. [2] provides guidelines and analysis to understand cost optimization in scientific workflow scheduling by surveying existing approaches in Cloud computing. Although the multi-objective offline scheduling problem that consists in meeting deadlines and respecting a budget has been extensively studied for deterministic workflows ([7], [15], [1], for example), it has received much less attention in a stochastic context. In a provider-centered point of view, [16] proposed a framework with the objective of meeting deadlines minimizing the impact of the execution of the workflow's execution on the cluster. In a slightly different context, *e.g.* with faults, preemptive tasks, from the provider point of view, [5] targets a complicated multi-criteria objective, namely (i) minimizing each job's completion time, (ii) maximizing the global utilization of the platforl and (ii) dividing fairly the resources between all the users in a scalable manner.

To the best of our knowledge, the closest papers to this work are [3], [25], which both propose workflow scheduling algorithms (BDT in [3], CG/CG+ in [25]) under budget and deadline constraints, but with a simplified platform model. We extended BDT and CG/CG+ to enable a fair comparison with our algorithms, and present the corresponding results in Section V-D. Finally, [19] also proposes workflow scheduling algorithms under budget and deadline constraints. Their platform model is similar to ours, although we allow for computation/transfer overlap and account for a startup delay t_{boot} to boot a VM. However, their application framework and objective are different: they consider workflow ensembles, *i.e.,* sets of workflows with priorities, that are submitted for execution simultaneously, and they aim at maximizing the number, or the cumulated priority value, of the workflows that complete successfully under the constraints. Still, we share the approach of partitioning the initial budget into chunks to be allotted to individual candidates (workflows in [19], tasks in this paper).

Workflows	
n	number of tasks in the workflow
T_i	The i^{th} task of the workflow
$\overline{w_i}$, σ_i	weight of T_i: mean, standard deviation
$size(d_{T_i,T_j})$	amount of data from T_i to T_j
Platform	
k	number of VM categories
$s_1 \leq s_2 \cdots \leq s_k$	VMs speeds
$\overline{s}$	average speed
$c_{h,k}$, $c_{ini,k}$	per time unit cost and initial cost for category k
c_{tsf}	per time unit cost of I/O operations
$c_{h,DC}$	per time unit cost of datacenter usage
bw	bandwidth between VMs and the datacenter

Table I: Summary of main notations.

III. Model

This section details the application and platform model used to assess the performance of the scheduling algorithms. Table I summarizes the main notations used in this paper.

A. Workflows

The model of workflows presented here is directly inspired by [17], [19]. A task workflow is represented with a DAG (Directed Acyclic Graph) $G = (V, E)$, where V is the set of tasks to schedule, and E is the set of dependencies between tasks. In this model, a dependency corresponds to a data transfer between two tasks. Tasks are not preemptive and must be executed on a single processor[1]. Most workflow scheduling algorithms use as starting assumption that the exact number of instructions constituting a task is known in advance, so that its execution time is given accurately. However, this hypothesis is not always realistic. The number of instructions for a given task may strongly depend on the current input data, such as in image processing kernels. In our model, we only know an estimation of the number of instructions for each task. For lack of knowledge about the origin of time variations, we assume that all the parameters which determine the number of instructions forming a task are independent. This resulting number is the task *weight* and follows a Gaussian law with mean $\overline{w_i}$ and standard deviation σ_i which can be estimated (for example by sampling).

To each dependency $(T_i, T_j) \in E$ is associated an amount of data of size $size(d_{T_i,T_j})$. We say that a task T is ready if either it does not have any predecessor in the dependency graph, or if all its predecessors have been executed and all the output data generated.

B. Platform

Our model of Cloud platform mainly consists of a datacenter and processing units. It is based to a great extent on the offers of three Cloud providers: Google Cloud[2], Amazon EC2[3] and OVH[4]. Given that Cloud providers propose a fault-tolerance service which ensures a very high availability of resources (in general over 99.97%[5]) as well as sufficient data redundancy, the datacenter and processing units are considered reliable and not subject to faults.

There is only one datacenter, used by all processing units. It is the common crossing point for all the data exchanges between processing units: these units do not interact directly, because of security issues for example. When a task T is to be executed on a VM v, all input data of T generated by one predecessor T' must be accessed from the datacenter, unless that this data has been produced on the same VM v (meaning that T' had been scheduled on v too). The datacenter is also where the final generated data are stored before being transferred to the user. For simplicity, we consider that the datacenter bandwidth is large enough to feed all processing units, and to accommodate all submitted requests simultaneously, without any supplementary cost.

The processing units are VMs (Virtual Machines). They can be classified in different categories characterized by a set of parameters fixed by the provider. Some providers offer parameters of their own, such as the number of forwarding rules[6]. We

[1]This assumption is only for the sake of the presentation; it is easy to extend the approach to parallel tasks.

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

[3]https://aws.amazon.com/ec2/pricing/on-demand/

[4]https://www.ovh.com/fr/public-cloud/instances/tarifs/

[5]https://cloudharmony.com/status

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

only retain parameters common to the three providers Google, Amazon and OVH: A VM of category k has n_k processors, one processor being able to process one task at a time; A VM has also a speed s_k corresponding to the number of instructions it can process per time unit, a cost per time-unit $c_{h,k}$ and an initial cost $c_{ini,k}$; All these VMs take an initial, and uncharged, amount of time t_{boot} to boot before being ready to process tasks. Already integrated in the schedule computing process, this starting time is thus not counted in the cost related to the use of the VM, which is presented in Section III-C. Without loss of generality, categories are sorted according to hourly costs, so that $c_{h,1} \leq c_{h,2} \cdots \leq c_{h,n_k}$. We expect speeds to follow the same order, but do not make such an assumption.

The platform thus consists of a set of n VMs of k possible categories. Some simplifying assumptions make the model tractable while staying realistic: (i) We assume that the bandwidth is the same for every VM, in both directions, and does not change throughout execution; (ii) A VM is able to store enough data for the tasks assigned to it: in other words, a VM will not have any memory/space overflow problem, so that every increase of the total makespan will be because of the stochastic aspect of the task weights; (iii) Initialization time is the same for every VM; (iv) Data transfers take place independently of computations, hence do not have any impact on processor speeds to execute tasks.

We chose an "on-demand" provisioning system: it is possible to deploy a new VM during the workflow execution if needed. Hence VMs may have different start-up times. A VM v is started at time $H_{start,v}$ and does not stop until all the data created by its last computed task have been transferred to the datacenter, at time $H_{end,v}$. VMs are allocated by continuous slots. If one wants discontinuous allocations, one may free the VM, then use a new one later, which at least requires sending all the data generated by the last processed task to the datacenter, and reloading all input data of the first task scheduled on that new VM before execution.

C. Workflow execution, cost and objective

Tasks are mapped to VMs and locally executed in the order given by the scheduling algorithm, such as those described in Section IV. Given a VM v, a task is launched as soon as (i) the VM is idle; (ii) all its predecessor tasks have been executed, and (iii) the output files of those predecessors mapped onto other VMs have been transferred to v via the datacenter.

a) Cost: The cost model is meant to represent generic features out of the existing offers from Cloud providers (Google, Amazon, OVH). The total cost of the whole workflow execution is the sum of the costs due to the use of the VMs and of the cost due to the use of the datacenter $\mathcal{C}_{DC}$. The cost C_v of the use of a VM v of category k_v is calculated as follows:

$$C_v = (H_{end,v} - H_{start,v}) \times c_{h,k_v} + c_{ini,k_v} \tag{1}$$

There is a start-up cost c_{ini,k_v} in Equation (1), and a term c_{h,k_v} proportional to usage duration $H_{end,v} - H_{start,v}$.

The cost for the datacenter is based on a cost per time-unit $c_{h,DC}$, to which we add a transfer cost. This transfer cost is computed with the amount of data transferred from the external world to the datacenter (size$(d_{in,DC})$), and from the datacenter to the outside world (size$(d_{DC,out})$). In other words, $d_{in,DC}$ corresponds to data that are input to entry tasks in the workflow, and $d_{DC,out}$ to data that are output from exit tasks. Letting $H_{start,first}$ be the moment when we book the first VM and $H_{end,last}$ be the moment when the data of the last processed task have entirely been sent to the datacenter, we have:

$$\begin{aligned} C_{DC} = &(\text{size}(d_{in,DC}) + \text{size}(d_{DC,out})) \times c_{tsf} \\ &+ (H_{end,last} - H_{start,first}) \times c_{h,DC} \end{aligned} \tag{2}$$

Altogether, the total cost is $C_{wf} = \sum_{v \in R_{VM}} C_v + C_{DC}$, where R_{VM} is the set of booked VMs during the execution.

b) Objective: Given a deadline $\mathcal{D}$ and a budget $\mathcal{B}$, the objective is to fulfil the deadline while respecting the budget:

$$\mathcal{D} \geq H_{end,last} - H_{start,first} \quad \text{and } \mathcal{B} \geq C_{wf} \tag{3}$$

A more complicated objective would be to find the schedule that minimizes the makespan while respecting the budget, namely $\min(H_{end,last} - H_{start,first})$ where $\mathcal{B} \geq C_{wf}$.

IV. Scheduling algorithms

This section introduces Min-MinBudg and HEFTBudg, the budget-aware extensions to Min-Min [6], [14] and HEFT [24], two reference scheduling algorithms widely used by the community. Section IV-A details the main algorithms, which assign a fraction of the remaining budget to the current task to be scheduled, while aiming at minimizing its completion time. Then Section IV-B provides refined versions of HEFTBudg that squeeze the most of any leftover budget to re-map some tasks to more efficient VMs. This leads to an improvement in the makespan, at the price of a much larger CPU time of the scheduling algorithms. We did not consider the corresponding refinement of Min-MinBudg because HEFTBudg turned out to be more efficient than Min-MinBudg in our simulations, always achieving a smaller makespan for the same budget.

A. Min-MinBudg *and* HEFTBudg

The budget-aware extensions of Min-Min and HEFT need to account both for the task stochasticity and budget constraint, while aiming at makespan minimization. Coping with task stochasticity is achieved by adding a certain quantity to the average task weight so that the risk of under-estimating its execution time is reasonably low, while retaining an accurate value for most executions. We use a conservative value for the weight of a task T, namely $\overline{w_T} + \sigma_T$.

As for the budget, given a workflow wf, we first reserve a fraction to cover the cost of the datacenter and VM initializations; we divide what remains into the workflow tasks. Let $\mathcal{B}_{ini}$ denote the initial budget. To estimate the amount to be reserved:

- For the cost of the datacenter, we need to estimate the duration $H_{end,last} - H_{start,first}$ of the whole execution (see Equation (2)). To this purpose, we consider an execution on a

Algorithm 1 Dividing the budget into tasks.

function DIVBUDGET($wf, \mathcal{B}_{calc}, \overline{s}, bw$)
  $W_{max} \leftarrow$ getMaxTotalWork(wf)
  $d_{max} \leftarrow$ getMaxTotalTransfData (wf)
  for each T **of** wf **do**
    $budgPTsk[T] \leftarrow \mathcal{B}_T \leftarrow \mathcal{B}_{calc} \times \frac{\frac{\overline{w_T}+\sigma_T}{\overline{s}} + \frac{size(d_{pred,T})}{bw}}{\frac{W_{max}}{\overline{s}} + \frac{d_{max}}{bw}}$
  end for
  return $budgPTsk$
end function

single VM of the first (cheapest) category, compute the total duration $W_{max} = \sum_{T \in wf}(\overline{w_T} + \sigma_T)$ and let

$$H_{end,last} - H_{start,first} = \frac{W_{max}}{s_1} + \frac{\text{size}(d_{in,DC}) + \text{size}(d_{DC,out})}{bw} \quad (4)$$

Altogether, we pay the cost of input/output data several times: with factor c_{tsf} for the outside world, with factor $c_{h,DC}$ for the usage of the datacenter (Equation (4)), and with factor $c_{h,1}$ during the transfer of data to and from the unique VM. However, there is no communication internal to the workflow, since we use a single VM.

• For the initialization of the VMs, we assume a different VM of the first category per task, hence we budget the amount $n \times c_{ini,1}$.

Combining these two choices is conservative: on the one hand, we consider a sequential execution, but account only for input and output data with the external world, eliminating all internal transfers during the execution; on the other hand, we reserve as many VMs as tasks, ready to pay the price for parallelism, at the risk of spending time and money due to data transfers during the execution. Altogether, we reserve the corresponding amount of budget and are left with $\mathcal{B}_{calc}$ for the tasks.

This reduced budget $\mathcal{B}_{calc}$ is shared among tasks in a proportional way (see Algorithm 1): we estimate how much time $t_{calc,T}$ is required to execute each task T, transfer times included, and allocate the corresponding part $\mathcal{B}_T$ of the budget in proportion to the whole for execution of the entire workflow $t_{calc,wf}$:

$$\mathcal{B}_T = \frac{t_{calc,T}}{t_{calc,wf}} \times \mathcal{B}_{calc} \quad (5)$$

In Equation (5), we use

$$t_{calc,T} = \frac{\overline{w_T} + \sigma_T}{\overline{s}} + \frac{size(d_{pred,T})}{bw},$$

where

$$size(d_{pred,T}) = \sum_{(T',T) \in E} size(d_{T',T}) \quad (6)$$

is the volume of input data of T from all its predecessors. Similarly, we use $t_{calc,wf} = \frac{W_{max}}{\overline{s}} + \frac{d_{max}}{bw}$, where $d_{max} = \sum_{(T_i,T_j) \in E} size(d_{T_i,T_j})$ is the total volume of data within the workflow. Computed weights ($\overline{w_T} + \sigma_T$ and W_{max}) are divided by the mean speed $\overline{s}$ of VM categories, while data sizes ($size(d_{pred,T})$ and d_{max}) are divided by the bandwidth bw between VMs and the datacenter. Again, it is conservative to assume that all data will be transferred, because some of them will be stored in-place inside VMs, so there is here another source of over-estimation of the cost. On the contrary, using the average speed $\overline{s}$ in the estimation of the computing time may lead to an under-estimation of the cost when cheaper/slower VMs are selected.

This subdivided budget is then used to choose the best VM to host each ready task (see Algorithm 2): the best host for a task T on platform $\mathcal{P}$ will be the one providing the best EFT (Earliest Finish Time) for T, among those respecting the amount of budget $\mathcal{B}_T$ allocated to T. The platform $\mathcal{P}$ is defined as the set of host candidates, which consists of already used VMs plus one fresh VM of each category. For each host candidate $host$, either already used (set Used_{VM}) or new candidate (set New_{VM}), we first evaluate the time $t_{Exec,T,host}$ needed to have T executed (*i.e.,* transfer of input data and computations) on $host$:

$$t_{Exec,T,host} = \delta_{new} \times t_{boot} + \frac{\overline{w_T} + \sigma_{task}}{s_{k_{host}}} + \frac{size(d_{in,T})}{bw} \quad (7)$$

In Equation (7), we introduce the boolean δ_{new} whose value is 1 if $host \in \text{New}_{\text{VM}}$ to account for its startup delay, and 0 otherwise. Also, some input data may already be present if $host \in \text{Used}_{\text{VM}}$, thus we use $size(d_{in,T})$ instead of $size(d_{pred,T})$ (see Equation (6)), defining $d_{in,T}$ as those input data not already present on $host$.

To compute $EFT_{T,host}$, the Earliest Finish Time of task T on host $host$, we account for its Earliest Begin Time $t_{begin,host}$ and add $t_{Exec,T,host}$. Then $t_{begin,host}$ is simply the maximum of the following quantities: (i) availability of $host$; (ii) end of transfer to the datacenter of any input data of T. The latter includes all data produced by a predecessor of T executed on another host; these data have to be sent to the datacenter before being re-emitted to $host$, since VMs do not communicate directly. There is a cost associated to these transfers, which we add to $t_{Exec,T,host} \times c_{h,host}$ to compute the total cost $c_{T,host}$ incurred to execute T on $host$. We do not write down the equation defining $t_{begin,host}$, as it is quite similar to previous ones. Since we already subtracted from the initial budget everything except the cost of the use of the VMs themselves, getBestHost() can safely use $\mathcal{B}_T$ as the upper bound for the budget reserved for task T.

The algorithm reclaims any unused fraction of the budget consumed when assigning former tasks: this is the role of the variable *pot*, which records any leftover budget in previous assignments. Finally, MIN-MINBUDG (Algorithm 3) and HEFTBUDG (Algorithm 4) are the counterpart of the original MIN-MIN and HEFT algorithms, extended with the provisioning for the budget. For some tasks, getBestHost() will not return the host with the smallest ETF, but instead the host with the smallest ETF among those that respect the allotted budget. The complexity of MIN-MINBUDG and HEFTBUDG is $O(n+e)p$, where n is the number of tasks, e is the number of dependence edges, and p the number of enrolled VMs. This complexity is the same as for the baseline versions, except that p is not fixed

a priori. In the worst case, $p = O(\max(n, k))$ because for each task we try all used VMs, whose count is possibly $O(n)$, and k new ones, one per category.

Algorithm 2 Choosing the best host for each ready task.

function GETBESTHOST($T, budgPTsk[T], \mathcal{P}, pot$)
  $\mathcal{B}_T \leftarrow budgPTsk[T] + pot$
  // initialisation: new host of cheapest category:
  $bestHost \leftarrow v$, where $v \in \text{New}_{\text{VM}}$ and $k_v = 1$
  $minEFT \leftarrow EFT_{T,bestHost}$
  for each $host$ **of** $(\text{Used}_{\text{VM}} \cup \text{New}_{\text{VM}})$ **do**
    if $((EFT_{T,host} < minEFT)$
    and $(c_{T,host} <= \mathcal{B}_T))$ **then**
      $minEFT \leftarrow EFT_{T,host}$
      $bestHost \leftarrow host$
      $pot \leftarrow \mathcal{B}_T - c_{T,host}$
    end if
  end for
  return $bestHost, pot$
end function

Algorithm 3 MIN-MINBUDG

function MIN-MINBUDG($wf, \mathcal{B}_{calc}, \mathcal{P}$)
  $\overline{s} \leftarrow calcMeanSpeed(\mathcal{P})$
  $bw \leftarrow getBw(\mathcal{P})$
  $budgPTsk \leftarrow divBudget(wf, \mathcal{B}_{calc}, \overline{s}, bw)$
  $pot, newPot \leftarrow 0$
  while ! $areEveryTasksSched(wf)$ **do**
    $selectedHost \leftarrow null$
    $selectedTask \leftarrow null$
    $minFT \leftarrow -1$
    $readyTasks \leftarrow getReadyTasks(wf)$
    for each T **of** wf **do**
      $host \leftarrow getBestHost(T, budgPTsk[T],$
        $\mathcal{P}, newPot)$
      $finishTime \leftarrow EFT_{T,host}$
      if $((minFT < 0)$
      or $(finishTime < minFT))$ **then**
        $minFT \leftarrow finishTime$
        $selectedTask \leftarrow T$
        $selectedHost \leftarrow host$
        $pot \leftarrow newPot$
      end if
    end for
    $sched[selectedTask] \leftarrow selectedHost$
    $schedule(selectedTask, selectedHost)$
    $update(\text{Used}_{\text{VM}})$
  end while
  return $sched$
end function

B. HEFTBUDG+ *and* HEFTBUDG+INV

This section details two refined versions of HEFTBUDG. Because of the many conservative decisions taken during the design of the algorithm, it is very likely that not all the initial budget $\mathcal{B}_{ini}$ will be spent by HEFTBUDG. In order to refine the solution returned from HEFTBUDG, we re-consider each decision taken and try to improve it. HEFTBUDG (just as HEFT) assigns priorities to the tasks based upon their bottom level [24]. Let LISTT be the ordered list of tasks by non-decreasing priority, and let *selSched* denote the schedule returned by HEFTBUDG. The first variant HEFTBUDG+ (see Algorithm 5) processes the tasks in the order of LISTT, hence in the same order as HEFT and HEFTBUDG, while HEFTBUDG+INV uses the reverse order. For both variants, let T be the task currently considered: we then generate new schedules obtained by assigning T on each already used VM except the one given by *selSched*, and on a new one for each category. We compute c_{tot} and $t_{calc,wf}$ for each of them, and keep the one which has the shortest makespan and respects the budget.

As mentioned in Section IV-A, HEFTBUDG (just as HEFT) has a complexity $O(n + e)p$, where $p = O(\max(n, k))$ in the worst case. Both HEFTBUDG+ and HEFTBUDG+INV start with a full iteration of HEFTBUDG; then, for each task, they try a new host and generate the resulting schedule. Hence their complexity is $O(n(n + e)p)$, where $p = O(\max(n, k))$ in the worst case. This is an order of magnitude more CPU demanding than HEFTBUDG, which limits their usage to smaller-size workflows.

Algorithm 4 HEFTBUDG

function HEFTBUDG($wf, \mathcal{B}_{calc}, \mathcal{P}$)
  $\overline{s} \leftarrow calcMeanSpeed(\mathcal{P})$
  $bw \leftarrow getBw(\mathcal{P})$
  $budgPTsk \leftarrow divBudget(wf, \mathcal{B}_{calc}, \overline{s}, bw)$
  LISTT $\leftarrow getTasksSortedByRanks(wf, \overline{s}, bw, \overline{lat})$
  $pot, newPot \leftarrow 0$
  for each T **of** LISTT **do**
    $host \leftarrow getBestHost(T, budgPTsk[T], \mathcal{P}, newPot)$
    $pot \leftarrow newPot$
    $sched[T] \leftarrow host$
    $schedule(T, host)$
    $update(\text{Used}_{\text{VM}})$
  end for
  return LISTT, $sched$
end function

V. SIMULATIONS

This section provides all the simulation results. We first describe the experimental setup in Section V-A. Next in Section V-B, we assess the performance of the main algorithms MIN-MINBUDG and HEFTBUDG, using the standard MIN-MIN and HEFT heuristics as a baseline for comparison. Then in Section V-C, we proceed to the refined variants HEFTBUDG+ and HEFTBUDG+INV, and quantify their improvement in terms of makespan, as well as their additional cost in terms of CPU time. Finally, Section V-D is devoted to comparing our algorithms with (extensions of) two formerly published competitors, namely BDT [3] and CG/CG+ [25].

Algorithm 5 HEFTBUDG+

function HEFTBUDG+($wf, \mathcal{B}_{ini}, \mathcal{P}$)
  $\mathcal{B}_{calc} \leftarrow getBudgCalc(wf, \mathcal{B}_{ini}, \mathcal{P})$
  LISTT, $selSched \leftarrow$ HEFTBUDG($wf, \mathcal{B}_{calc}, \mathcal{P}$)
  $c_{tot}, t_{calc,wf} \leftarrow simulate(wf, \mathcal{P}, \text{LISTT}, selSched)$
  $minTimeCalc \leftarrow t_{calc,wf}$
  for each T **of** LISTT **do**
    for each $host$ **of** $(\text{Used}_{\text{VM}} \backslash sched[T]) \cup \text{New}_{\text{VM}}$ **do**
      $sched \leftarrow schedule(T, host)$
      $c_{tot}, t_{calc,wf} \leftarrow simulate(wf, \mathcal{P}, \text{LISTT}, sched)$
      if $((t_{calc,wf} < minTimeCalc)$
       and $(c_{tot} < \mathcal{B}))$ **then**
        $selectedHost \leftarrow host$
        $minTimeCalc \leftarrow t_{calc,wf}$
      end if
    end for
    $selSched[T] \leftarrow selectedHost$
    $update(\text{Used}_{\text{VM}})$
  end for
  return LISTT, $selSched$
end function

VM parameters	
Categories	$k = 3$
Setup delay	$t_{boot} = 10$ min
Setup cost	$c_{ini,\ell} = \$2$ for $1 \leq \ell \leq 3$
Category 1 (Slow)	Speed $s_1 = 5.2297$ Gflops
	Cost $c_{h,1} = \$0.145$ per hour
Category 2 (Medium)	Speed $s_2 = 8.8925$ Gflops
	Cost $c_{h,2} = \$0.247$ per hour
Category 3 (Fast)	Speed $s_3 = 13.357$ Gflops
	Cost $c_{h,3} = \$0.370$ per hour
Datacenter	
Cost per month	$c_{h,DC} = \$0.022$ per GB
Data transfer cost	$c_{tsf} = \$0.055$ per GB
Bandwidth	
bw	125MBps

Table II: Parameters of the IAAS Cloud platform.

A. Experimental methodology

We designed a simulator based on SimDag [22], an extension of the discrete event simulator SimGrid [10], to evaluate all algorithms. The model described in Section III is instantiated with 3 VM categories and respective costs inspired from the offers by Amazon Cloud, Google Cloud and OVH (see Table II): the cost of our VMs is based on the mean of the prices of S3 when we made our experiments, and is linear with the speed of the VM. The VM is paid for each used second.

We used three types of workflow from the Pegasus benchmark suite [21], [13]: CYBERSHAKE, LIGO and MONTAGE. Concerning LIGO, most input data have the same (large) size, only one of them is oversized compared with the others (by a ratio over 100). LIGO consists of a lot of parallel tasks sharing a link to some agglomerative tasks, one agglomerative task per little set; this scheme repeats twice since there is a second subdivision after the first agglomeration. In CYBERSHAKE, half the tasks have huge input data. The workflow itself consists of a first set of tasks generating data in parallel, data which will be used by a directly connected task (one calculating task per generating task). These parallel activities are all linked to two different agglomerative tasks. On the contrary, MONTAGE has plenty highly inter-connected tasks, rendering parallelization less easy. The number of instructions of its different tasks is balanced, as is the size of the exchanged data. For each workflow type, we used the simulator available on the Pegasus website to generate our benchmark, with five different instances per workflow type, and different numbers of tasks: 30, 60 and 90; this leads to $5 \times 3 = 15$ workflows per type.

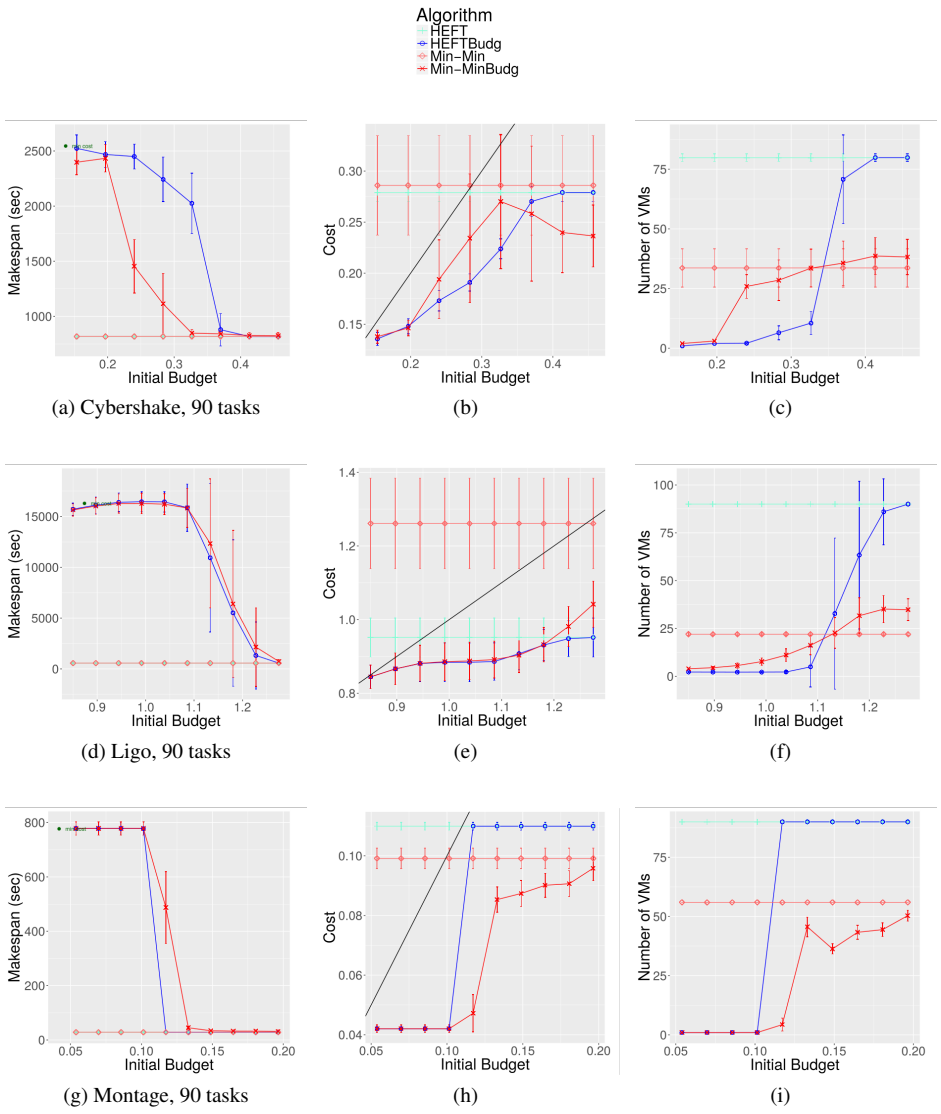
Each generated workflow is then re-used to generate workflows having the same DAG structure, but with different values for task weights: to this purpose, we keep the original weight for a task T as the mean $\overline{w_T}$ and use $25, 50, 75$ and 100% of that mean number for the standard deviation σ_T. In the figures, each simulation was repeated 25 times and we plot mean values; vertical bars represent standard deviations. Overall, 16500 experiments have been executed per workflow type and scheduling algorithm.

Then, to further assess the efficiency of the previous algorithms, we made some comparisons with two competitors: Budget Distribution with Trickling (BDT [3]) and Critical Greedy (CG [25]). Both BDT and CG schedule deterministic workflows, and CG does not take into account communication costs. In [25], CG also comes with a refined version CG+. We extended BDT and CG/CG+ to fit our model, so as to enforce fair comparisons. We propose two scenarios, first with unrefined versions (MIN-MINBUDG and HEFTBUDG against BDT and CG), and then with refined ones (HEFTBUDG+ and HEFTBUDG+INV against CG+).

B. MIN-MINBUDG *and* HEFTBUDG

In this section, we report and compare results for MIN-MINBUDG and HEFTBUDG against those for the baseline algorithms MIN-MIN and HEFT. Due to lack of space, we report only results for 90 tasks. Results for 30 and 60 tasks are available in the extended version [8]. Figure 1 has three rows, one per workflow type, and reports the results as a function of the initial budget: makespan in first column, total cost in second column, and number of VMs in third column. We record the number of used VMs as an indicator of some choices made by the budget-aware algorithms that trade-off increased usage of former VMs vs. enrolment of new ones. In the first column, the green dot labeled min_cost represents the mean of the cheapest solutions for the corresponding schedule, and is obtained by allocating all tasks to the same cheapest host.

The budget constraint is respected in almost all cases (see Figures 1b, 1e and 1h; the black line draws the affine function of the initial budget). Exceptions are some instances of LIGO with a budget near the minimum needed to schedule all the tasks. The explanation is the following: we assumed that the bandwidth of the datacenter would be sufficient for all simultaneous transfers, but we observed that it became a bottleneck in that case. LIGO has a lot of parallel tasks running concurrently, that



(a) Cybershake, 90 tasks (b) (c)

(d) Ligo, 90 tasks (e) (f)

(g) Montage, 90 tasks (h) (i)

Figure 1: MIN-MIN, HEFT, budget-aware extensions MIN-MINBUDG and HEFTBUDG for the three workflow types with 90 tasks.

may well send huge data at the same time. In those very few cases, we underestimated the time needed to send these data.

Clearly, when given an infinite initial budget, MIN-MIN and HEFT give the same schedule as MIN-MINBUDG and HEFTBUDG respectively. We see that MIN-MIN and HEFT obtain similar makespans, but HEFT uses more VMs than MIN-MIN: for instance there is an average of 79 VMs for HEFT vs. 33 for MIN-MIN for CYBERSHAKE, 90 vs. 22 for LIGO and 90 vs. 56 for MONTAGE. The cost is thus smaller for more parallel workflows (LIGO), and larger for the other ones (CYBERSHAKE, MONTAGE).

We now discus the initial budget needed by the budget-aware algorithms to achieve the minimal makespan returned by the baseline version. HEFTBUDG needs a smaller initial budget than MIN-MINBUDG for MONTAGE (see Figure 1g), and a similar one for CYBERSHAKE and LIGO (see Figures 1a and 1d). We refine this analysis in the extended version [8]: the difference in minimal budgets decreases sharply with the number of tasks for CYBERSHAKE and LIGO. This is due to the graph structure of these workflows: for CYBERSHAKE, increasing the number of tasks leads to workflows with a majority of parallel tasks; for LIGO, it leads to an increasing number of independent short workflows. In both cases, increasing the number of tasks renders the workflow closer to a Bag of Tasks, and the priority mechanism of HEFTBUDG becomes less useful. On the contrary, larger MONTAGE workflows keep numerous imbricated dependencies between tasks, and HEFTBUDG remains more efficient in terms of budget.

Regarding the behavior of our algorithms, increasing initial budget will lead to enrolling more VMs with an exception: in Figure 1i, we see that the number of VMs can rise for intermediate values of budgets until exceeding that of the baseline version, then decrease again. This corresponds to the moment when several tasks have enough budget to leave their mid-

(a)

	MIN-MIN	HEFT	MIN-MINBUDG	HEFTBUDG	BDT	CG
Low	2.89 ± 0.39 3.13	2.78 ± 0.33 2.99	2.06 ± 0.23 2.19	2.60 ± 0.31 2.79	1.88 ± 0.26 1.74	3.35 ± 0.71 2.88
Medium	2.90 ± 0.37 3.13	2.76 ± 0.33 2.98	2.06 ± 0.21 2.19	2.59 ± 0.30 2.78	2.48 ± 0.43 2.21	3.34 ± 0.71 2.86
High	2.90 ± 0.39 3.14	2.77 ± 0.33 2.98	2.07 ± 0.22 2.20	3.32 ± 0.39 3.60	2.47 ± 0.44 2.22	2.45 ± 0.42 2.19

(b)

	MIN-MIN	HEFT	MIN-MINBUDG	HEFTBUDG	BDT	CG
30	0.13 ± 0.002 0.14	0.16 ± 0.02 0.17	0.10 ± 0.01 0.11	0.20 ± 0.003 0.22	0.13 ± 0.00 0.13	0.13 ± 0.00 0.13
60	0.88 ± 0.01 0.88	0.90 ± 0.01 0.90	0.63 ± 0.00 0.63	1.11 ± 0.01 1.11	0.91 ± 0.00 0.91	0.90 ± 0.01 0.90
90	2.90 ± 0.39 3.14	2.77 ± 0.33 2.98	2.07 ± 0.22 2.20	3.32 ± 0.46 3.60	2.47 ± 0.44 2.22	2.45 ± 0.42 2.19
400	395.80 ± 15.83 395.06	294.96 ± 15.83 297.23	268.15 ± 12.41 269.54	341.00 ± 14.64 340.15	363.95 ± 65.21 361.50	452.73 ± 110.38 380.26

Table III: Time to calculate a schedule, in seconds, in the form mean $\pm$ standard value, median: (a) MONTAGE workflow of 90 tasks and different budgets; (b): MONTAGE workflow with 30, 60, 90 and 400 tasks, and a high budget.

efficient VM and migrate to a fastest category VM.

We have also assessed the impact of the amount of uncertainty in task weights, but because of space limitations, we refer to the extended version [8]. Both HEFTBUDG and MIN-MINBUDG require a larger initial budget to achieve a given makespan, when σ increases; yet the budget constraint is respected, even in scenarios where task weights can be twice their mean value.

Finally we discuss the execution time of each algorithm. The experiments have been conducted on a computer with a Intel® Core™ i5-6200U CPU @ 2.30GHz $\times$ 4 processors. We recorded the time needed for each algorithm while calculating 5 continuous schedules, and executed 30 instances for each combination of parameters. We used three types of workflows (CYBERSHAKE, LIGO and MONTAGE) instantiated with 30, 60 and 90 tasks. As for the impact of the budget on the time needed to calculate a schedule, we used three characteristic values for each workflow, which we designate as "low", "high" and "medium". A "low" budget $\mathcal{B}_{min}$ corresponds to the minimum budget needed to find a schedule, a "high" one to a budget large enough to enroll an unlimited number of VMs. The "medium" budget is chosen as follows: for each workflow, we empirically find the minimum budget $\mathcal{B}_{\min_{best}}$ needed to obtain a makespan as good as the one found for baseline version of the algorithm, and take the average: $\mathcal{B}_{med} = \frac{\mathcal{B}_{\min_{best}} + \mathcal{B}_{min}}{2}$ Table III shows CPU times needed to calculate a schedule for workflows of varying type and size. For example, for a MONTAGE workflow of 90 tasks, HEFTBUDG needs 2.87 ± 0.52 seconds to find a schedule when it only needs 0.60 ± 0.39 seconds for a CYBERSHAKE workflow or 0.72 ± 0.40 seconds for a LIGO workflow; such differences can be seen for the other algorithms as well.

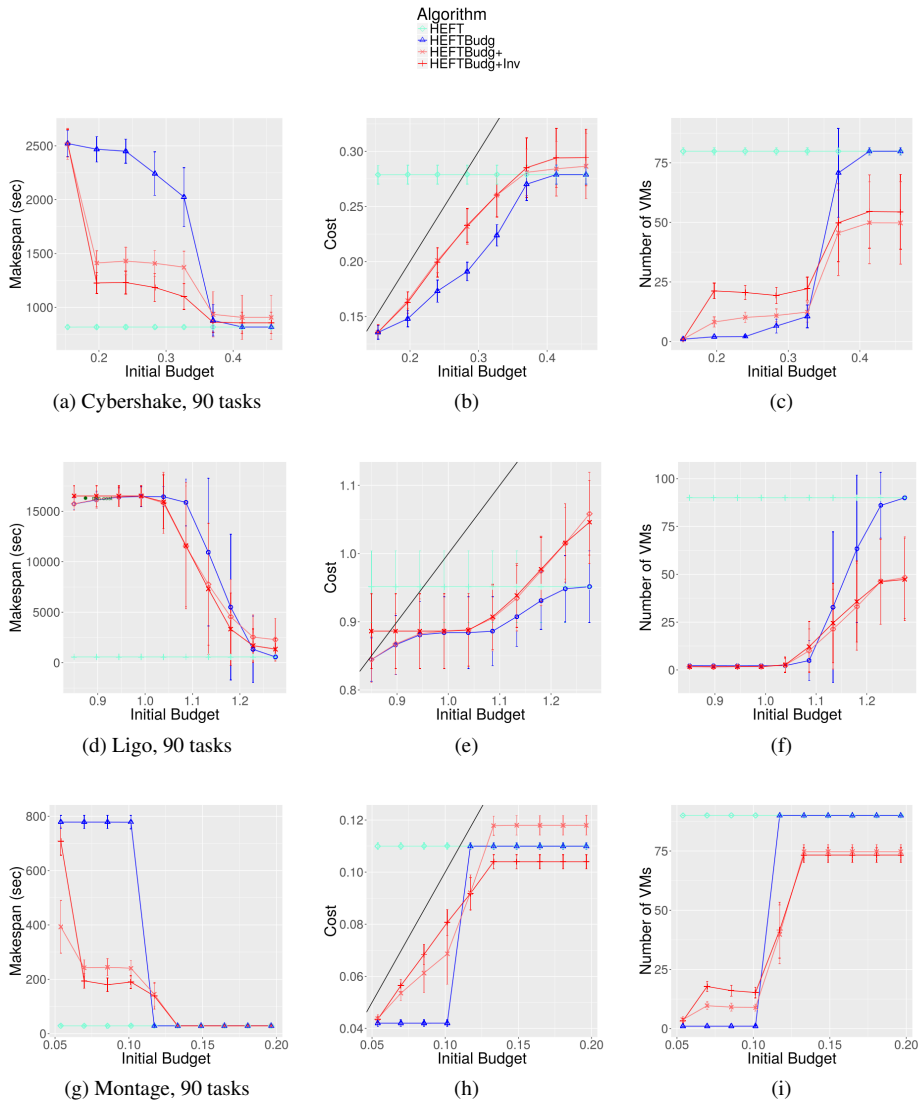
Here is a concluding remark about MIN-MINBUDG and HEFTBUDG: To cope with uncertainty in task weights, we made a pessimistic estimation of the cost of data transfers, assuming they were always produced by another VM. This was safe but led to overestimating both the time and the budget for these transfers. Also the budget assignment is somewhat unfair to the first scheduled tasks, which have no access to any leftover resource (the pot). In the next section, we aim at improving the schedule by re-examining the assignment and budget allotted to each task. We do this for HEFTBUDG only, because it typically achieves a smaller makespan than MIN-MINBUDG for a prescribed budget. Of course, similar improvements could be designed for MIN-MINBUDG.

C. HEFTBUDG+ *and* HEFTBUDG+INV

Figure 2 is the counterpart of Figure 1 to compare HEFTBUDG+ and HEFTBUDG+INV against HEFT and HEFTBUDG. The schedules obtained for both refined algorithms HEFTBUDG+ and HEFTBUDG+INV have a shorter makespan than HEFTBUDG (see Figure 2a, 2d and 2g). Their makespan can be up to one third shorter than for HEFTBUDG, *e.g.,* for MONTAGE.

Surprisingly the refined algorithms manage to achieve a smaller makespan using *fewer* VMs than HEFTBUDG. This is mostly because they succeed in assigning interdependent tasks onto the same VM. The initial budget needed to obtain the same makespan as HEFT is the same for HEFTBUDG, HEFTBUDG+ and HEFTBUDG+INV. Moreover, the budget is respected overall, as was already the case with HEFTBUDG.

To compare HEFTBUDG+ and HEFTBUDG+INV, we observe that their makespans are very similar, apart from the case where a budget near the minimum is needed to complete a schedule: in that case, HEFTBUDG+ obtains an average makespan twice shorter than HEFTBUDG+INV. This difference seen in this exact configuration may be due to the particular structure of MONTAGE and CYBERSHAKE workflows: they have a lot of initial tasks, and the amount of work for every kind of tasks is of the same magnitude. In the case of LIGO, there were few to none improvement, probably because of the structure near from a Bag of Tasks of these workflows. As for the overspent of the budget that we can sometimes see for very small initial budgets, it is likely due to the saturation of the bandwidth of the datacenter which is not taken into account. However, smaller makespans for HEFTBUDG+ and HEFTBUDG+INV are obtained at a much higher computa-



(a) Cybershake, 90 tasks (b) (c)

(d) Ligo, 90 tasks (e) (f)

(g) Montage, 90 tasks (h) (i)

Figure 2: HEFTBUDG+, HEFTBUDG+INV compared to HEFT and HEFTBUDG for the three workflow types with 90 tasks.

tional time. For instance, for MONTAGE with 90 tasks and a high budget, HEFTBUDG finds a solution in 2.60 ± 0.28 seconds while HEFTBUDG+ needs 379.45 ± 44.20 seconds and HEFTBUDG+INV needs 382.29 ± 43.25 seconds.

D. *Comparison with* BDT *and* CG/CG+

In this section, we compare our algorithms with two competitors, BDT [3] and CG/CG+ [25]. Beforehand, we briefly describe them and explain how we have extended them to match our application/platform model. A word of caution: some of our choices may be seen as arbitrary, but we have tried to be as fair as possible to allow for a meaningful comparison.

1) BDT *(Budget Distribution with Trickling):* BDT is divided into three major steps: (i) traversing the graph and grouping tasks into levels, *i.e.,* subgroups of independent tasks; (ii) sharing the budget across the different levels, according to one chosen strategy. We implemented the strategy leading to the best results in [3], *All in*, which tentatively grants all the budget to the first task of the current level. That task is not expected to consume all the budget, and the leftover is given to the next task in the level; (iii) scheduling tasks level by level. Inside a level, tasks are sorted on their increasing Earliest Start Time. Then for each task, the best host $host$ is selected to maximize the ratio $TCTF_t^{host} = \frac{Time_t^{host}}{Cost_t^{host}}$ where $Cost_t^{host} = \frac{subBudg_t - c_{t,host}}{subBudg_t - c_{min}}$ and $Time_t^{host} = \frac{ECT_{max} - ECT_{t,host}}{ECT_{max} - ECT_{min}}$. Here $subBudg_t$ is the budget fraction allocated to the task t, $ECT_{t,host}$ and $c_{t,host}$ are the Earliest Completion Time (ECT) and total cost of task t on host $host$ respectively, and c_{min} the minimal cost possible for the execution of t (cheapest VM). Finally, ECT_{min} and ECT_{max} are respectively the smallest and largest ECT possible for task t, when trying all possible VM choices for t.

We have extended this algorithm to match our model, using the same task weights as in our own propositions. BDT uses an eager scheduling strategy, aiming at a very low makespan but at the risk of overspending the budget. Also, it is better suited to

DAGs that can be decomposed into independent levels of tasks with similar costs.

2) CG/CG+ *(Critical Greedy):* This algorithm is divided in two parts: generation of an initial schedule CG, then refinement into another schedule CG+. CG first defines a global value $gbl = \frac{\mathcal{B}-c_{min}}{c_{max}-c_{min}}$ to be used later to partition the budget $\mathcal{B}$ across the tasks. Here c_{min} is the minimal budget needed to execute the workflow (assigning all tasks to a single VM of the cheapest type), and c_{max} is the maximal one (assigning all tasks to a VM of the most expensive type). Then for each task t of the workflow (the ordering is not specified in [25] so we used HEFT), the algorithm computes the quantity $c_{t,min} + (c_{t,max} - c_{t,min}) \times gbl$ which represents the budget fraction predetermined for task t, with $c_{t,min}$ being the minimal cost needed to compute the task t and $c_{t,max}$ the maximal possible cost to compute the task t. It then selects the VM category whose cost for task t has the smallest difference in absolute value with that quantity.

Once the first schedule has been obtained with CG, it is refined to spend any leftover budget. The tasks belonging to the critical path of the schedule are re-assigned to more efficient VMs. Among these tasks, CG+ selects the task and VM pair so that re-assigning that task to that VM provides the largest ratio $\frac{\delta T}{\delta c}$, where δT is the time decrease and δc the cost increase when making the re-assignment. The refinement continues until all the budget is spent.

There are no data transfers in [25], so we had to extend CG/CG+ to include all transfer times and costs.

3) Results: Figure 3 reports several key results for MIN-MINBUDG, HEFTBUDG, BDT and CG, on CYBERSHAKE, LIGO and MONTAGE workflows. The first row represents the makespan for different initial budgets; the second row shows the percentage of executions for which each algorithm found a makespan enforcing the initial budget, and the third row shows the actual budget spent *w.r.t.* the given initial budget. We see that BDT often fails to find a valid schedule, *i.e.,* a schedule enforcing the initial budget, especially for small budgets; however when a schedule is found, its makespan is smaller than those found by MIN-MINBUDG and HEFTBUDG. For example on a CYBERSHAKE workflow of 90 tasks, BDT needs three times the lowest required budget to find a schedule. As for CG, it returns schedules that are close to the cheapest possible schedule possible. This is due to the way the budget is shared between tasks: the sub-budget given to a task is, in general, enough to choose between different instances of the cheapest VM type, but does not allow to afford a better VM type. The schedule generated by HEFTBUDG or MIN-MINBUDG (which have benefited from the transmission of leftover budgets to the next tasks) have then a smaller makespan.

Furthermore, we see in Figure 4 that CG+ keeps finding schedules with high makespans. The main reason is probably the chosen heuristic: CG+ maximizes the ratio between the time decrease and the cost increase, one pair (task, VM) after the other on the critical path. If a newly found assignment has both a lower cost and a lower makespan, the $\frac{\delta T}{\delta c}$ ratio will be negative and thus the new assignment will not be selected. This happens for example in the case when a re-assignment removes a data transfer.

The CPU execution times of BDT and CG (see Table III) have the same order of magnitude as HEFT or MIN-MIN. The execution time of CG+ is huge, an order of magnitude higher than HEFTBUDG+, but we do not report precise values for fairness: we believe that part of the high execution time is due to a technical requirement within SimDag when running the simulations.

VI. CONCLUSION

In this paper, we have presented a model and several budget-aware algorithms to schedule scientific workflows with stochastic task weights onto IaaS Cloud platforms. These platforms allow to dynamically enroll VMs of several types, with different cost and speed parameters. The first two algorithms, MIN-MINBUDG and HEFTBUDG, are extensions of the well-known MIN-MIN and HEFT heuristics. We show that they manage to find a solution whose makespan remains similar to the one of the baseline versions while enforcing the prescribed budget. We observe that for a given budget HEFTBUDG obtains a better makespan than MIN-MINBUDG, in particular for workflows with a non-trivial inter-dependency graph. We then propose two refined versions, HEFTBUDG+ and HEFTBUDG+INV, that achieve better makespans, but at a higher CPU time cost. We also extend and implement two previously published budget-aware algorithms, BDT and CG/CG+. While BDT returns the smallest makespans when it succeeds, it fails to enforce the initial budget most of the time. Globally our algorithms find better schedules than CG/CG+. Again, these comparisons must be made with caution, because the application/platform cost models of BDT and CG/CG+ were cruder than the detailed framework used in this paper.

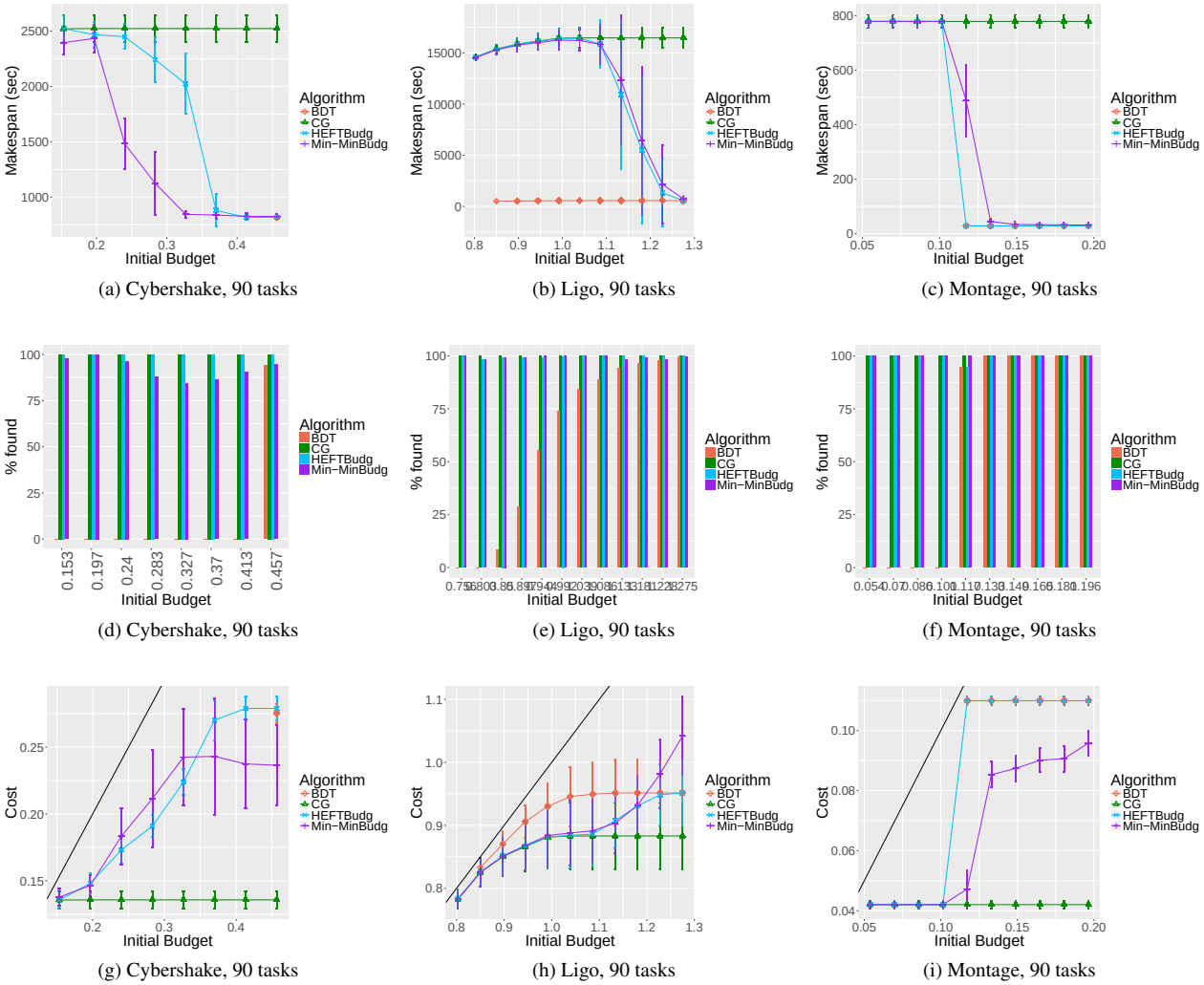
Further work will be devoted to extending the approach to on-line schedules, whenever the target Cloud infrastructure would allow to interrupt and re-schedule tasks on the fly. Indeed, if we monitor the execution of the tasks, we can detect unlikely events such as very long durations, and in such cases, it could be beneficial to interrupt some tasks and re-schedule them onto faster VMs. Such dynamic decisions encompass risks in terms of both final makespan and budget. For instance, deriving execution timeouts is a challenging problem, but we hope to design on-line heuristics that, with high probability, will decrease the final makespan while respecting the initial budget constraint.

REFERENCES

[1] S. Abrishami, M. Naghibzadeh, and D. H. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Gener. Comput. Syst.*, 29(1):158–169, Jan. 2013.

[2] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi. Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. *Journal of Systems and Software*, 113:1–26, 2016.

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



(a) Cybershake, 90 tasks (b) Ligo, 90 tasks (c) Montage, 90 tasks

(d) Cybershake, 90 tasks (e) Ligo, 90 tasks (f) Montage, 90 tasks

(g) Cybershake, 90 tasks (h) Ligo, 90 tasks (i) Montage, 90 tasks

Figure 3: Makespan, percentage of valid schedules and cost for MIN-MINBUDG, HEFTBUDG, BDT and CG for CYBERSHAKE, LIGO and MONTAGE with 90 tasks.

[4] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Characterization of scientific workflows. In *SC'08 Workshop: The 3rd Workshop on Workflows in Support of Large-scale Science (WORKS08) web site*, Austin, TX, Nov. 2008. ACM/IEEE.

[5] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 285–300, Berkeley, CA, USA, 2014. USENIX Association.

[6] T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. P. Robertson, M. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[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. K. W. Chang, and Y. Robert. Budget-aware scheduling algorithms for scientific workflows with stochastic task weights on IaaS cloud platform. Research Report 9128, INRIA, Nov. 2017.

[9] E. Caron, F. Desprez, T. Glatard, M. Ketan, J. Montagnat, and D. Reimert. Workflow-based comparison of two distributed computing infrastructures. In *Workflows in Support of Large-Scale Science (WORKS10)*, New Orleans, November 14 2010. In Conjunction with Supercomputing 10 (SC'10), IEEE. hal-00677820.

[10] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *J. Parallel Distributed Computing*, 74(10):2899–2917, 2014.

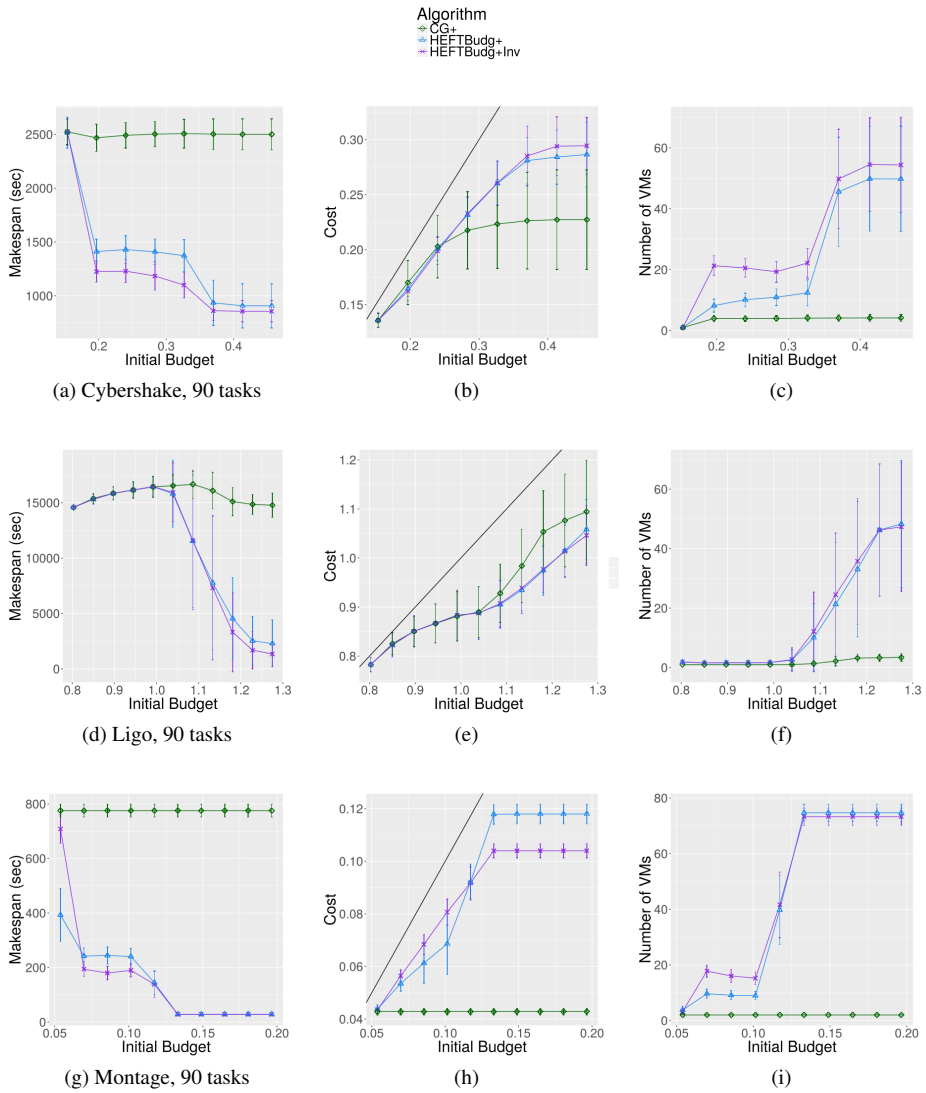
[11] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger. Workflow Management in Condor. In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, *Workflows for e-Science*, pages 357–375. Springer, 2007.

[12] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. Jacob, and D. Katz. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming Journal*, 13(3):219–237, 2005.

[13] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46(0):17–35, 2015.

[14] P. Ezzatti, M. Pedemonte, and A. Martín. An efficient implementation of the min-min heuristic. *Comput. Oper. Res.*, 40(11), 2013.

[15] 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.



(a) Cybershake, 90 tasks (b) (c)

(d) Ligo, 90 tasks (e) (f)

(g) Montage, 90 tasks (h) (i)

Figure 4: HEFTBUDG+ and HEFTBUDG+INV compared to CG+ for the three workflow types with 90 tasks.

[16] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: Guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 99–112, New York, NY, USA, 2012. ACM.

[17] G. Juve, A. L. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi. Characterizing and profiling scientific workflows. *Future Generation Comp. Syst.*, 29(3):682–692, 2013.

[18] C. Lin and S. Lu. Scheduling scientific workflows elastically for cloud computing. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 746–747. IEEE, 2011.

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

[20] M. Mao and M. Humphrey. Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In *Parallel and Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 67–78. IEEE, 2013.

[21] Pegasus. Pegasus workflow generator. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator, 2014.

[22] SimDag. Programming environment for DAG applications. http://simgrid.gforge.inria.fr/simgrid/3.13/doc/group__SD__API.html, 2017.

[23] S. Smanchat and K. Viriyapant. Taxonomies of workflow scheduling problem and techniques in the cloud. *Future Generation Computer Systems*, 52:1–12, 2015.

[24] H. Topcuoglu, S. Hariri, and M. Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distributed Systems*, 13(3):260–274, 2002.

[25] 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.

[26] D. Yuan, Y. Yang, X. Liu, and J. Chen. A data placement strategy in scientific cloud workflows. *Future Generation Comp. Syst*, 26(8):1200–1214, 2010.

[27] L. Zeng, B. Veeravalli, and X. Li. SABA: A security-aware and budget-aware workflow scheduling strategy in clouds. *J. Parallel Distrib. Comput*, 75:141–151, 2015.