Budget-aware Static Scheduling of Stochastic Workflows with DIET

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

Budget-aware Static Scheduling of Stochastic Workflows with DIET

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

***Abstract*—Previous work has introduced a Cloud platform model and budget-aware static algorithms to schedule stochastic workflows on such platforms. In this paper, we compare the performance of these algorithms obtained via simulation and via execution on an actual platform, Grid'5000. We focus on DIET, a widely used workflow engine, and detail the extensions that were implemented to conduct the comparison. We also detail additional code that we made available in order to automate and to ease the reproducibility of such experiments.**

DIET; static workflow scheduling; Cloud platform.

I. Introduction

Public Cloud has emerged as an interesting tool for scientists, offering an infrastructure adaptable on demand, with a variety of available options and performances. Multiple workflow engines for Cloud platforms have been designed, with more and more functionalities [1] [2] [3]. These workflow engines aim at helping users to pick the most appropriate resources for their intended work, in terms of the number and characteristics of the Virtual Machines (VMs) selected for execution. The main objective is to enable easy-to-produce applications while guaranteeing that, given a constrained budget, rented resources are used up to the maximum of their capacity.

In [4], we described a Cloud platform model, and we detailed and compared several static budget-aware scheduling algorithms in an extensive simulation campaign. One major objective of this work is to further validate the conclusions of the simulation study by conducting real-life experiments with the same scientific workflows on the French national validation platform Grid'5000 [5] and assess the accuracy of simulation results on a real-life platform. Additional contributions are the evolution of DIET (Distributed Interactive Engineering Toolbox) to handle static scheduling, and a tool that automatically generates DIET code to execute the target workflows.

The rest of the paper is organized as follows. We first study related work in Section II. Then, in Section III, we introduce the DIET middleware and describe the new DIET functionalities. In Section IV, we overview the budget-aware algorithms that we aim at comparing. Section V details the experimental framework. Results are reported in Section VI, with a comparison analysis between real executions on Grid'5000 and the corresponding simulations.

II. Related work

A. Workflow engines

Many scientific applications from various disciplines are structured as workflows [6]. 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. For a long time, the development of complex middleware with workflow engines [7], [2], [3] has automated workflow management. For example, the Pegasus Workflow Management System [3] maps workflows on resources until a given horizon beyond which it considers pre-scheduling is inefficient, and allows its users to plug their own scheduling algorithms if needed. Steep [8] comes along its own way to schedule workflows, submitting complete process chains to its remote agents, and supports cyclic workflows graphs. Apache Airflow [9] is more oriented on accessibility to most users, hence its focus on the user interface and the use of Python to describe workflows or interact with the engine. In [10], the authors summarize the key features of four production-ready WMSs: Pegasus, Makeflow, Apache Airflow, and Pachyderm. For this work, we focus on DIET [11] because of its practicality, and the possibility to extend it with additional modules.

Infrastructure as a Service (IaaS) Clouds raised a lot of interest recently thanks to an elastic resource allocation and pay-as-you-go billing model. A Cloud user can adapt the execution environment to the needs of his application on a virtually infinite supply of resources. While the elasticity provided by IaaS Clouds gives way to more dynamic application models, it also raises new issues from a scheduling point of view. An execution now corresponds to a certain budget, which imposes some constraints on the scheduling process. In [12], the authors propose a performance-feedback autoscaler that is budget-aware: using Apache Airflow, they tackle the same scheduling problem as in this paper, but they focus on the auto-scaling problem (allocating and de-allocating resources on the fly).

B. Budget aware static algorithms

Scheduling scientific workflows in cloud is a well-studied domain [13], [14]. To the best of our knowledge, the closest papers to the budget-aware algorithms with stochastic execution times that we designed in [4] and compare in this paper are [15] and [16], which both propose workflow scheduling algorithms (BDT in [15], CG/CG+ in [16]) under budget

constraints, but with a simplified platform model. As in our previous work [17], we have extended BDT and CG/CG+ to enable a fair comparison with our algorithms. More recent scheduling algorithms exist, but address different aspects of the problem. For example, [18] aims at simultaneously minimizing the cost and makespan of workflow executions, but they use a simplified platform model without Cloud storage. Another study [19] uses a platform closer to ours, but with a completely different objective, namely the minimization of data transfers (which should eventually reduce both makespan and cost too). Finally, the workflows considered in [20] present an uncertainty in task durations, but the authors schedule multiple workflows at the same time instead of optimizing for a single workflow.

III. Framework

Scientific workflows are 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. Workflows are scheduled on an heterogeneous platform representing an IaaS Cloud, with a tarification depending on the performances of the used machines, on the amount of data transferred to and out of the Cloud, and on the amount of time during which each machine has been used (see Section IV-B for cost instances).

A. DIET workflow engine

DIET is a well established workflow engine [11] [1]. A DIET platform (Figure 1) consists of a hierarchy of agents (Master Agents MA and Local Agents LA) scheduling the requests addressed by a client and sending them to the appropriate servers, the Servers Deamon (SeD). DIET users, following the GridRPC paradigm, usually submit individual tasks, but it is possible to submit a whole workflow.

In this case, the client sends the XML file which describes its structure to a special agent, called a MA_{DAG}, which consequently manages task dependencies and handles ready tasks submissions to the related MA. The MA dynamically determines which server will execute a request. A server first looks for the files needed for the task execution, downloads them if they are not already on the server, and then executes the request.

B. Extending the MA_{DAG} for static scheduling

DIET was able to schedule workflows dynamically. We improved the MA_{DAG} to apply a static schedule given by a user. From the user point of view, the client sends to the MA_{DAG} the usual files needed for the execution of the workflow, plus files describing the desired schedule:

- The Workflow Description File (WDF): the XML file describing the workflow as in any DIET execution of workflow,
- The Desired Schedule file (DSF): a file giving the desired schedule. Each line gives the machine attributed to a task, under the formulation <task> <server>. The order of the tasks gives their priority, the first ones having the highest priority. The static schedule is also used to write the code of the target servers, hence this file has to be made available before launching execution,
- The Mapping File (MF): a file giving the equivalence between the name of the platform servers and their counterpart from the DSF. Each line gives a machine-server equivalence, under the formulation <machine> <server>.

DIET will then follow the given schedule.

On a more technical level, it is not possible to specify in the MA_{DAG} which server will receive a given task: the MA_{DAG} leaves the duty of selecting a server to its MA, only sending it the name of the services ready to be executed. To alleviate this limitation, we exploit the fact that the MA uses the name of the service to find which server can run it. In order to let the user to be able to change the task-server attribution according to the information gathered by the DIET agents, we choose to have the servers declare their services twice. The first one is a regular generic declaration, with a name identical to the one given in the XML file describing the workflow, and the same for all the servers able to run it. This allows the MA_{DAG} to get the list of all the servers available to run a given service corresponding to a task;The second one is specific to the new functionality and has a name composed of the concatenation between the name of the service as described in the XML file and the name of the server. This local name is the one which will be used to send the request once the server is chosen.

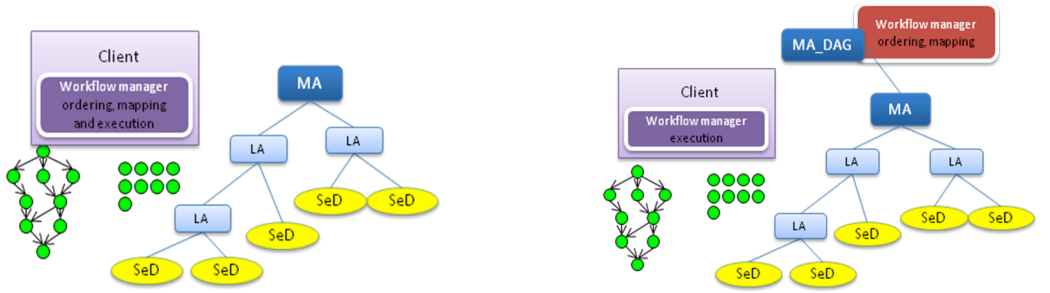
C. Experiment-oriented tools

Given the large number of runs needed by the experiments, we have developed a couple of tools to automate the creation of the workflows meant to be executed on Grid'5000. We first use a home-made simulator [21] to generate the schedules with the selected algorithms. We then generate the corresponding DIET workflows needed for our experiments with a home-made workflow generator, OGMa [21].

Our simulator is based on simDAG [22]. It generates both the static schedules given to DIET and the simulated results. Basically, it needs a description of the platform, the DAX file (https://pegasus.isi.edu/documentation/development/schemas.html) of the workflow, and the parameters of the scheduling problem (budget, algorithms, etc.). It creates a schedule, writes the mapping <task, VM> into a file intended for OGMa, in the order of their attribution, as determined by the chosen algorithm. Then, it simulates with simDAG the execution of the given workflow on the given platform, using the calculated mapping. Finally, it calculates and writes in a file the resulting makespan, cost, and various other metrics.

Once the simulations done and the schedules calculated, we used OGMa to generate the elements needed for the experiments on Grid'5000. Concretely, OGMa uses the given DAX file describing the target workflow, along with information on the focused platform and simulations, and the static schedule, to write all files: the source files for the servers and the client, the configuration files needed for every DIET entities, placeholder files for the workflow, the DSF, the MF and the WDF.

We point out that, as DAX files do not give any details about the real content of the tasks or the files they describe, OGMa



(a) Architecture 1: Complete schedule from the Client.

(b) Architecture 2: Schedule is handled by the MA_{DAG}

Figure 1: Two different architectures of the DIET workflow engine.

only creates placeholder files and tasks. If a user wants to use OGMa as a tool to generate the raw structure of the workflow, they will have to replace these placeholders with the actual code for the tasks.

IV. Budget-aware scheduling algorithms

Details on this section can be found in [4].

A. Workflow model

A workflow is represented with a DAG of stochastic tasks. Tasks are not preemptive and must be executed on a single processor. 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 *weight* w_i of task T_i and follows a truncated Normal law with mean $\overline{w_i}$ and standard deviation $\sigma_i \overline{w_i}$. Finally, to each dependency $(T_i, T_j) \in E$ is associated an amount of data of size $size(d_{T_i,T_j})$.

B. Cloud platform model

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.

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. We 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 that 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. Without loss of generality (even if the VM is paid for each used second), 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.

Altogether, the platform 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 all 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 duration 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; (v) a VM executes at most one task at every time-step, but this task can be parallel and enroll many computing resources (hence the execution time of the task strongly depends upon the VM type).

We chose an "on-demand" provisioning system: it is possible to deploy a new VM during the workflow execution. Hence VMs may have different startup 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 is transferred to the Cloud storage, at time $H_{end,v}$.

C. Scheduling costs 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-D. 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 others VMs have been transferred to v via the Cloud storage.

a) Costs: 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 Cloud storage $\mathcal{C}_{CS}$. The cost C_v of the use of a VM v of category k_v is calculated as follows:

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

There is a startup 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 Cloud storage is based on a cost per time-unit $c_{h,CS}$, to which we add a transfer cost. This transfer cost is

computed with the amount of data transferred from the external world to the Cloud storage (size($d_{in,CS}$)), and from the Cloud storage to the outside world (size($d_{CS,out}$)). In other words, $d_{in,CS}$ corresponds to data that are input to entry tasks in the workflow, and $d_{CS,out}$ 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 Cloud storage, we define $H_{usage} = H_{end,last} - H_{start,first}$ as the total platform usage during the whole execution. We have:

$$C_{CS} = \begin{array}{l}(\text{size}(d_{in,CS}) + \text{size}(d_{CS,out})) \times c_{tsf} \\ +H_{usage} \times c_{h,CS}\end{array} \quad (2)$$

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

b) Objective: Given a budget $\mathcal{B}$ and a platform $\mathcal{P}$, the objective is to minimize total execution time while respecting the budget.

D. The HEFTBUDG scheduling algorithm

HEFTBUDG (Algorithm 1) is a budget-aware extension of HEFT [23]. This extension accounts both for task stochasticity and budget constraints, 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 underestimating 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 $\overline{w_T} + \sigma_T \overline{w_T}$.

Algorithm 1 HEFTBUDG.

```
1: function HEFTBUDG(wf, B_calc, P)
2:     s̄ ← calcMeanSpeed(P)
3:     bw ← getBw(P)
4:     budgPTsk ← divBudget(wf, B_calc, s̄, bw)
5:     ListT ← getTasksSortedByRanks(wf, s̄, bw, lat)
6:     pot, newPot ← 0
7:     for each T of ListT do
8:         host ← getBestHost(T, budgPTsk[T], P, newPot)
9:         pot ← newPot
10:        sched[T] ← host
11:        schedule(T, host)
12:        update(Used_VM)
13:    end for
14:    return ListT, sched
15: end function
```

In the beginning HEFTBUDG calls *divBudget()* (Algorithm 2): given the workflow wf, we first get the maximum of total work (getMaxTotalWork(wf)) and the total amount of data transfers (getMaxTotalTransfData (wf)) required to execute the workflow, and we reserve a fraction of the budget to cover the cost of the Cloud storage and VM initialization; then we divide what remains, $\mathcal{B}_{calc}$, into the workflow tasks. To estimate the fraction of budget to be reserved, assuming that $\mathcal{B}_{ini}$ denotes the initial budget:

- For the cost of the Cloud storage, we need to estimate the duration $H_{usage} = H_{end,last} - H_{start,first}$ of the whole execution (see Equation (2)). To this purpose, we consider an execution on a single VM of the first (cheapest) category, compute the total duration $W_{max} = \sum_{T \in wf} (\overline{w_T} + \sigma_T)$ and let
$$H_{usage} = \frac{W_{max}}{s_1} + \frac{\text{size}(d_{in,CS}) + \text{size}(d_{CS,out})}{bw} \quad (3)$$
Altogether, we pay the cost of input/output data several times: with factor c_{tsf} for the outside world, with factor $c_{h,CS}$ for the usage of the Cloud storage (Equation (3)), 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: we estimate how much time $t_{calc,T}$ is required to execute each task T, transfer times included, and allocate the corresponding part of the budget in proportion to the whole for execution of the entire workflow $t_{calc,wf}$:

$$budgPTsk[T] = \frac{t_{calc,T}}{t_{calc,wf}} \times \mathcal{B}_{calc} \quad (4)$$

In Equation (4), we use $t_{calc,T} = \frac{\overline{w_T} + \sigma_T}{\overline{s}} + \frac{size(d_{pred,T})}{bw}$, where

$$size(d_{pred,T}) = \sum_{(T',T) \in E} size(d_{T',T}) \quad (5)$$

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 Cloud storage. 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 by calling *getTaskssortedbyranks()*: the best host for a task T will be the one providing the best Earliest Finish Time (EFT) for T, among those respecting the amount of budget $\mathcal{B}_T$ allocated to T (considering all already used VMs plus one fresh VM of each category).

HEFTBUDG reclaims any unused fraction of the budget consumed when assigning former tasks: this is the role of the

Algorithm 2 Dividing the budget onto tasks.

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

variable *pot*, which records any leftover budget in previous assignments. For some tasks, getBestHost() do not return the host with the smallest EFT, but instead the host with the smallest EFT among those that respect the allotted budget. The complexity of 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.

E. Other scheduling algorithms

[4] details several budget-aware scheduling algorithms:

- MINMINBUDG, a budget-aware extension of MINMIN [24], [25], the exact counterpart of HEFTBUDG.
- HEFTBUDG+ and HEFTBUDG+INV, aiming at exploiting the opportunity to re-schedule some tasks onto faster VMs, thereby spending any budget leftover by the first allocation, at a price of higher complexity. These refined variants differ by the order in which tasks are considered, and recompute the schedule after processing each task.
- HEFTBUDGMULT, a trade-off version that reallocates the leftover budget in a single pass: it finds makespans slightly larger than those computed with HEFTBUDG+, but with a time complexity close to HEFTBUDG.

Two competitors to the new budget-aware algorithms described above have also been simulated in [4], for the sake of comparison. These are Budget Distribution with Trickling (BDT [15]) and Critical Greedy (CG [16]). Both BDT and CG schedule deterministic workflows, and CG does not take into account communication costs. In [16], CG also comes with a refined version CG+. BDT and CG/CG+ have been extended to fit the model, so as to enforce fair comparisons.

V. EXPERIMENTAL FRAMEWORK

We use the new DIET functionalities to execute the scheduling algorithms (Section IV) on Grid'5000, a French national testbed supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations [5]. For each workflow execution, we used 31 homogeneous nodes from the `grisou` cluster in Nancy (Intel Xeon E5-2630 v3 CPU 2.4 GHz $\times 8$ cores). The three different types of VMs (see Table I) needed for the experiments have been emulated: a VM two times slower than another one executes twice as many computations. We also run the corresponding experiments with our simulator.

Table I: Parameters of the platform.

VM parameters	
Categories	$k = 3$
Setup cost	$c_{ini,=}$ \$0.00056
Category 1 (Slow)	Speed $s_1 = 3.2$ Gflops Cost $c_{h,1} =$ \$0.118 per hour
Category 2 (Medium)	Speed $s_2 = 6.4$ Gflops Cost $c_{h,2} =$ \$0.236 per hour
Category 3 (Fast)	Speed $s_3 = 9.6$ Gflops Cost $c_{h,3} =$ \$0.354 per hour
Cloud storage	
Cost per month	$c_{h,CS} =$ \$0.022 per GB
Data transfer cost	$c_{tsf} =$ \$0.055 per GB
Bandwidth	
bw	1GBps

We used three types of workflows: MONTAGE, LIGO and CYBERSHAKE, generated with [26]. Given the large makespan of these workflows (e.g., 33 hours for one CYBERSHAKE of 60 tasks scheduled with CG/CG+ [4]), we only ran small instances with 30 tasks.

A. Simulations

We used the simulator described in Section III-C to obtain both the static schedules to be evaluated on Grid'5000 and the simulated results, with characteristics of the platform (bandwidth: 1Gb/s, performances of the VMs used as slowest type: 3.2Gf) experimentally measured on `grisou`.

B. Grid'5000 experiments

One needs to enforce that tasks have similar durations in the simulations and in the experiments. In the workflow description file, the amount of work of a task is given in seconds, when the amount of data transferred between two tasks is given in bytes. Thus for OGMa, a task consists of three phases: a phase during which the input files are read, a phase during which an amount of double floating-point additions is calculated based on the task duration given in the DAX file and the information provided about the used platform, and a phase during which the output files are written.

SimDAG uses a fixed (but arbitrary) number to calculate a number of flops based on the task duration in seconds given by the DAX file. This arbitrary number did not correspond to the VMs enrolled on Grid'5000. To preserve a similar ratio between the time used to move data $t_{data,T}$ and the time used to execute a task $t_{calc\_only,T}$ in the real setup and in the simulation, we modified the number of operations for each service according to the observed characteristics of the platform.

In the simulator, the amount of time $t_{calc\_only,T}$ used to execute the task T, composed of w_T operations, on a reference host of speed $s_{host,simu}$ operations per second, and without counting data transfers, is equal to $t_{calc\_only,T} = \frac{w_T}{s_{host,simu}}$. The amount of time $t_{data,T}$ to transfer all the data needed to execute T, for a total size of $size(d_{pred,T})$, with a bandwidth bw_{simu} is equal to $t_{data,T} = \frac{size(d_{pred,T})}{bw_{simu}}$. Hence the ratio to preserve is $r_{simu} = \frac{w_T}{size(d_{pred,T})} \times \frac{bw_{simu}}{s_{host,simu}}$.

Similarly, in the Grid'5000 execution, the ratio for a bandwidth bw_{g5000} and a reference host speed $s_{host,g5000}$, with the same amount of transferred data $size(d_{pred,T})$ and a task T composed of $w_{T,g5000}$ operations is $r_{g5000} = \frac{w_{T,g5000}}{size(d_{pred,T})} \times \frac{bw_{g5000}}{s_{host,g5000}}$. Finally, we calculated the number of operations needed to execute the task T on a reference host of Grid'5000 as $w_{T,g5000} = w_{T,simu} \times \frac{s_{host,g5000}}{s_{host,simu}} \times \frac{bw_{simu}}{bw_{g5000}}$. We used this number of operations to generate workflow tasks of appropriate duration.

In the case of LIGO and CYBERSHAKE, instead of running an equally long workflow as described in the DAX files, we chose to generate one with the same shape (same tasks, dependencies between tasks, proportion between data transfers and amount of work for tasks), but whose malespan is made shorter, with the application of a fixed coefficient to decrease the time of execution of the workflows. As seen in Section VI, this has close to no impact on the performance of the algorithms.

As per the execution itself with DIET, we used one VM to run the client task, the MA, the MA_{DAG} agent and omniNames (the naming service on which DIET relies to operate communication between its components), and one VM per server.

C. Experimental campaign

We have generated schedules: (i) for three workflow types: CYBERSHAKE, LIGO and MONTAGE; and (ii) for ten scheduling algorithms: MINMINBUDG, HEFTBUDG, HEFTBUDGMULT, HEFTBUDG+, HEFTBUDG+INV, BDT, MINMIN, CG, CG+ and HEFT. We have selected a range of budget values which have an actual impact on the makespan. For some of these budget values, a few algorithms are failing to produce valid schedules. Here we define a *valid* schedule as a schedule which successfully enforces the budget constraint.

We have executed, on Grid'5000 and on the simulator, 30 experiments per combination (budget $\times$ algorithm $\times$ workflow), and collected the makespan and cost of each execution. The costs are calculated as in [4], with prices adapted to the performances of the used VMs. The raw data, their treatment, and the whole experimentation setup, are available in [21].

VI. RESULTS

We first focus on the observation of makespan results from real life experiments compared to their simulation. Even results that spend more than the allocated budget are considered (in other words, these results are produced by non-valid schedules). Then we assess the accuracy of experimental costs conducted with the MONTAGE workflow, with a comparison between real life experiments and their simulation, and in this case we restrict to results under the given budget.

The experiments have been carried so that the generated workflows are equivalent to their simulated counterpart, with a fixed factor as only difference. For the sake of comparison, and to highlight how similar the obtained results are between the simulation and the execution on Grid'5000, we scale them in the figures: MONTAGE makespans are 4.6 times lower than their simulation, LIGO ones are 49.06 times lower, and CYBERSHAKE ones 6.9 times lower.

In Figure 2, we report the makespans obtained for each type of workflow as a function of the available budget. The first column represents the results obtained with the simulator. The second column represents the results obtained from the runs with DIET on Grid'5000. In most cases, the hierarchy of algorithms in the DIET executions is the same as the one in simulation. CG and CG+ obtain the highest makespans, and HEFT, BDT and MINMIN the lowest ones. Among the budget-aware algorithms, MINMINBUDG gets most of the time the highest makespans, but is inferior to the ones obtained with CG+. HEFTBUDG obtains the second highest ones. HEFTBUDG+ and HEFTBUDG+INV find the schedules with the lowest makespans. HEFTBUDGMULT schedules have makespans between HEFTBUDG and HEFTBUDG+ ones.

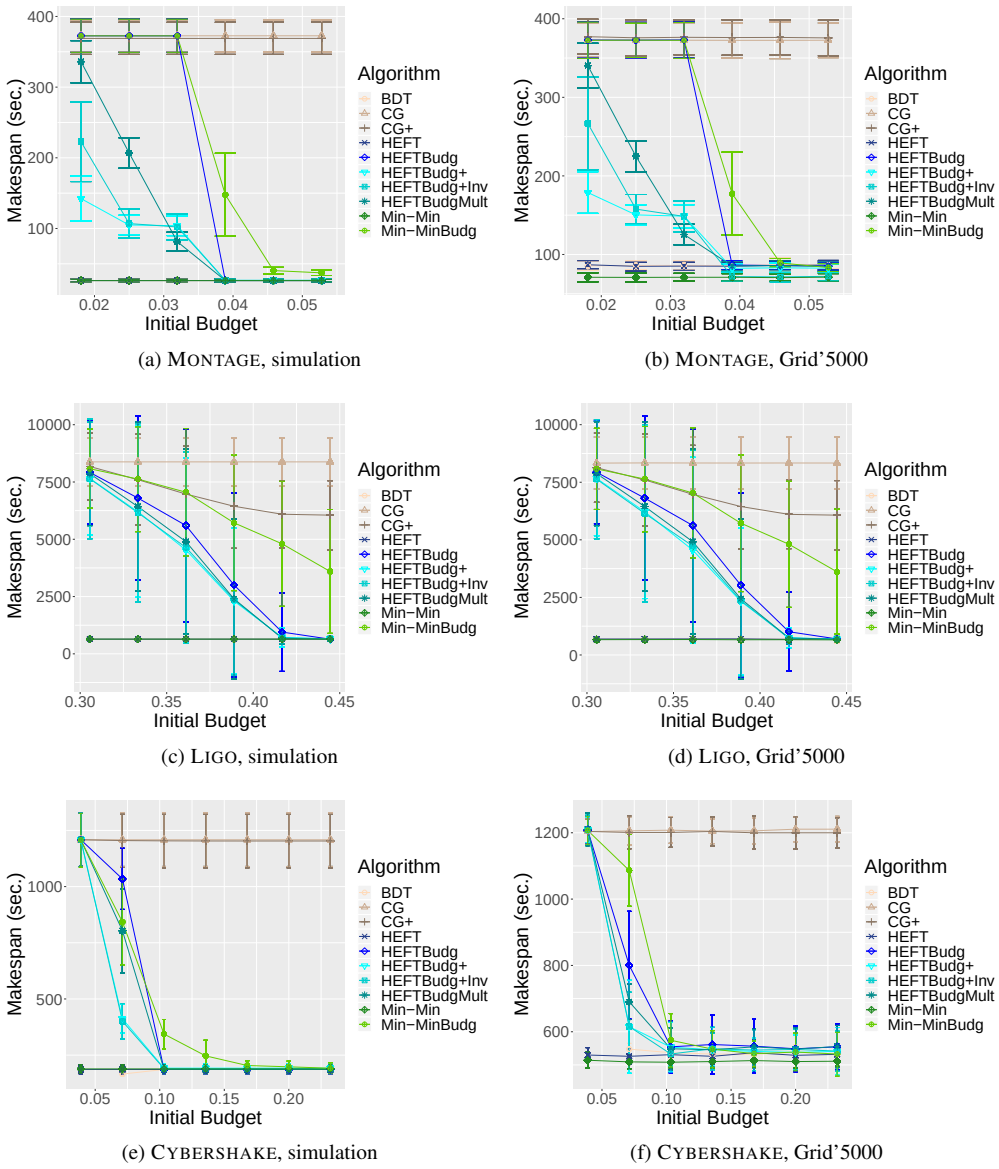
While the results of simulation and real execution are very similar for MONTAGE and LIGO, there are some differences for CYBERSHAKE. We guessed that these differences are due to file transfers, and we reran the simulations with an infinite bandwidth, but this had no impact. Still, overall, there is a pretty good correspondence between simulations and actual runs.

Next, we focus on MONTAGE workflows, and in Figure 3, we report the costs and the percentage of valid solutions found for MONTAGE executions. In Figures 3a and 3b, we see the cost of the execution of MONTAGE workflows, both for simulations and real executions, and they match almost perfectly. With low budgets, two algorithms achieve very expensive schedules: BDT and HEFT. They are followed by MINMIN. All the other budget-aware algorithms, spend twice less budget to make a schedule. In addition, the higher the budget, the more optimization opportunities for budget-aware algorithms. For the highest budgets used for our experiments, we make the following observations: (i) HEFTBUDG+ achieves the most expensive schedules, even higher than the ones from HEFT and BDT (but recall from Figure 2 that HEFTBUDG+ needed a lower initial budget than HEFT and BDT to find valid makespans); (ii) HEFTBUDG, HEFTBUDGMULT and HEFTBUDG+INV have a cost similar to HEFT (but achieve lower makespans); (iii) Similarly, MINMINBUDG and MINMIN schedules have similar costs (and lower makespans for MINMINBUDG); and (iv) The cheapest schedules come from CG and CG/CG+ (but this is at the cost of a far larger makespan).

Figures 3c and 3d show the percentage of valid schedules found by each algorithm, from simulations (left) and real-life experiments (right). Only HEFTBUDG+INV differs on the two lowest budgets without a 100% valid schedules in real-life. We know from [4] that this algorithm refines its schedule, leaving only a small leftover budget, which explains the difference. On each graph, we see that for the lowest budget, BDT, HEFT and MINMIN give no valid schedule, and for the second lowest budget, BDT and HEFT still do not. All the other algorithms give 100% valid solutions.

VII. CONCLUSION

In this paper, we have introduced a new scheduling functionality for DIET, and provided the user with a set of tools to implement, and experiment with, static scheduling algorithms



(a) MONTAGE, simulation

(b) MONTAGE, Grid'5000

(c) LIGO, simulation

(d) LIGO, Grid'5000

(e) CYBERSHAKE, simulation

(f) CYBERSHAKE, Grid'5000

Figure 2: Makespans for MONTAGE, CYBERSHAKE and LIGO workflows of 30 tasks, execution on simulation vs. Grid'5000.

for workflows. We then used this new functionality to compare the executions of ten static algorithms for scientific applications from the Pegasus benchmark, using both a simulator and the testbed Grid'5000. Both types of experiments gave similar results, validating the results obtained during the simulations executed in [5] and DIET improvements.

Further work will be devoted to better understand the behavior of budget-aware algorithms on larger and more diverse workflows, using the insights gained from both the similarities and differences found in simulations and actual executions.

REFERENCES

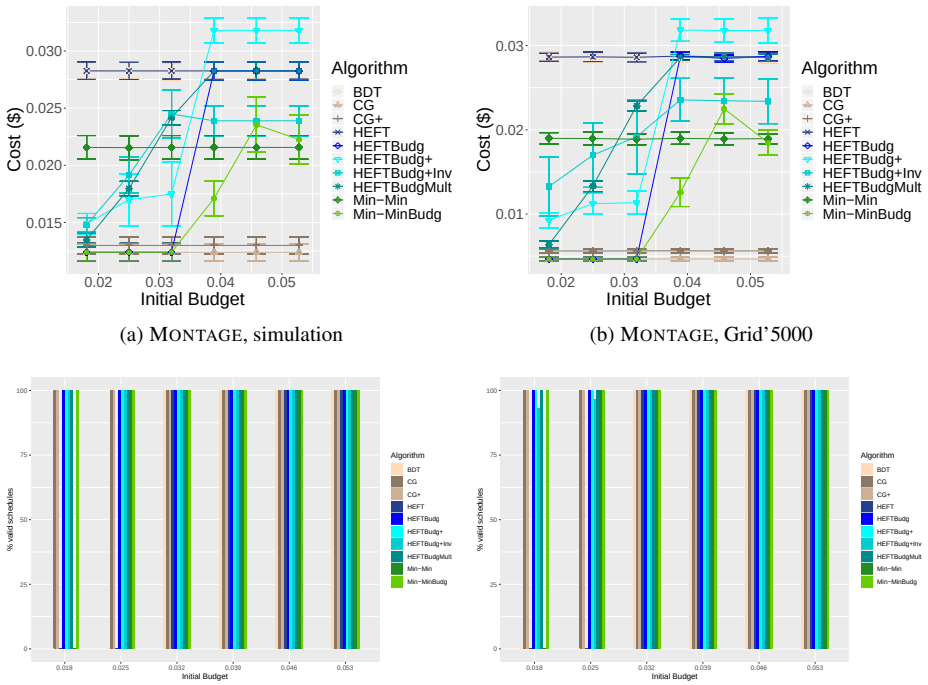
[1] E. Caron and F. Desprez, "DIET: A scalable toolbox to build network enabled servers on the grid," *International Journal of High Performance Computing Applications*, vol. 20, no. 3, pp. 335–352, 2006.

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

[3] 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*, vol. 46, pp. 17–35, 2015, funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575.

[4] Y. Caniou, E. Caron, A. Kong Win Chang, and Y. Robert, "Budget-aware scheduling algorithms for scientific workflows with stochastic task weights on infrastructure as a service cloud platforms," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 17, p. e6065, 2021, [retrieved: August, 2021]. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6065

[5] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jegou, S. Lanteri, N. Melab, R. Namyst, P. Primet, O. Richard, E. Caron, J. Leduc, and G. Mornet, "Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform," in *Proc. 6th IEEE/ACM Int. Workshop on Grid Computing (Grid'2005)*. IEEE Computer Society Press, 2005, see



(a) MONTAGE, simulation

(b) MONTAGE, Grid'5000

(c) MONTAGE, percentage of valid solutions, simulation

(d) MONTAGE, percentage of valid solutions, Grid'5000

Figure 3: Costs for MONTAGE workflows of 30 tasks, execution on Grid'5000 vs. simulation, and percentage of valid solutions.

https://www.grid5000.fr/w/Grid5000:Home.

[6] 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: ACM/IEEE, Nov. 2008.

[7] 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)*, In Conjunction with Supercomputing 10 (SC'10). New Orleans: IEEE, November 14 2010, hal-00677820, [retrieved: August, 2021]. [Online]. Available: https://hal.inria.fr/hal-00677820

[8] M. Krämer, "Capability-based scheduling of scientific workflows in the cloud," in *DATA*, S. Hammoudi, C. Quix, and J. Bernardino, Eds. SciTePress, 2020, pp. 43–54.

[9] M. Beauchemin. (2014) Apache airflow project. [retrieved: August, 2021]. [Online]. Available: https://airflow.apache.org/

[10] R. Mitchell, L. Pottier, S. Jacobs, R. F. d. Silva, M. Rynge, K. Vahi, and E. Deelman, "Exploration of workflow management systems emerging features from users perspectives," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 4537–4544.

[11] E. Caron, "Contribution to the management of large scale platforms: the DIET experience," HDR (Habilitation 'a Diriger les Recherches), École Normale Supérieure de Lyon, Oct.6 2010, hal number tel-00629060, [retrieved: August, 2021]. [Online]. Available: https://hal.inria.fr/tel-00629060

[12] A. Ilyushkin, A. Bauer, A. V. Papadopoulos, E. Deelman, and A. Iosup, "Performance-feedback autoscaling with budget constraints for cloud-based workloads of workflows," *CoRR*, vol. abs/1905.10270, 2019, [retrieved: August, 2021]. [Online]. Available: http://arxiv.org/abs/1905.10270

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

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

[15] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget distribution strategies for scientific workflow scheduling in commercial clouds," in *IEEE 12th Int. Conf. on e-Science (e-Science)*, Oct 2016, pp. 137–146.

[16] 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*, vol. 3, no. 2, pp. 169–181, April 2015.

[17] T. Risset and Y. Robert, "Synthesis of processor arrays for the algebraic path problem," *Parallel Processing Letters*, vol. 1, no. 1, pp. 19–28, Sep. 1991.

[18] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, and S. Hu, "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft," *Future Generation Computer Systems*, vol. 93, pp. 278 – 289, 2019.

[19] S. Makhlouf and B. Yagoubi, "Data-aware scheduling strategy for scientific workflow applications in iaas cloud computing," *Int. J. Interactive Multimedia and Artificial Intelligence*, vol. InPress, p. 1, 01 2018.

[20] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, and N. Najjari, "Online multi-workflow scheduling under uncertain task execution time in iaas clouds," *IEEE Trans. Cloud Computing*, pp. 1–1, 2019.

[21] A. Kong Win Chang. https://graal.ens-lyon.fr/~achang/Research/. [retrieved: August, 2021].

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

[23] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[24] 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*, vol. 61, no. 6, pp. 810–837, 2001.

[25] P. Ezzatti, M. Pedemonte, and A. Martín, "An efficient implementation of the min-min heuristic," *Comput. Oper. Res.*, vol. 40, no. 11, 2013.

[26] Pegasus, "Code for the Pegasus generator," https://github.com/pegasus-isi/WorkflowGenerator, 2020, [retrieved: August, 2021].