A workflow scheduling deadline-based heuristic for energy optimization in Cloud

Emile Cadorel, Hélène Coullon, Jean-Marc Menaud

A workflow scheduling deadline-based heuristic for energy optimization in Cloud

Emile Cadorel
IMT Atlantique, Inria, LS2N, UBL
F-44307 Nantes, France
emile.cadorel@imt-atlantique.fr

Hélène Coullon
IMT Atlantique, Inria, LS2N, UBL
F-44307 Nantes, France
helene.coullon@imt-atlantique.fr

Jean-Marc Menaud
IMT Atlantique, Inria, LS2N, UBL
F-44307 Nantes, France
jean-marc.menaud@imt-atlantique.fr

***Abstract*—This article addresses the scheduling of heterogeneous scientific workflows while minimizing the energy consumption of the cloud provider, by introducing a deadline sensitive algorithm. Scheduling in a cloud environment is a difficult optimization problem. Usually, work around the scheduling of scientific workflows focuses on public clouds where infrastructure management is an unknown black box. Thus, many works offer scheduling algorithms designed to select the best set of virtual machines over time, so that the cost to the end user is minimized. This article presents a new HEFT-based algorithm that takes into account users deadlines to minimize the number of machines used by the cloud provider. The results show the real benefits of using our algorithm for reducing the energy consumption of the cloud provider.**

***Keywords*-Cloud Computing, Scientific Workflow, Scheduling, Virtual Machines, Algorithm, HEFT**

I. Introduction

The Cloud computing is a well-known paradigm defined by the NIST as "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

Cloud providers are no longer confined to hosting *ever-running* services, such as web servers or databases for instance. Indeed, nowadays Clouds host many types of applications and computations such as, for instance, Big-Data, machine learning or very short batch jobs. Moreover, different types of computations may also be combined by scientists into complex heterogeneous scientific workflows or dataflows [1], [2]. Scientific workflows model scientific applications as a set of coarse-grained tasks linked together by data dependencies. These workflows generally involve data of different sizes and time-consuming computation tasks.

To be able to host scientific workflows on Clouds, the allocation of virtual machines (VMs) and their planning over time is a major challenge for both the provider and the user (scientist). Indeed, with an optimized VMs allocation (*i.e.*, resource sharing), a Cloud provider will be able to rent more VMs and reduce its costs. Similarly, the rental prices of VMs will be reduced, which will result in more attractive offers for the user. Most of the related work done to this day for planning scientific workflows on the Cloud considers the Cloud Provider's internal scheduler as an unknown black box. As a result, much work focuses on optimizing from the user's point of view, trying to minimize the cost of use or trying to stick to a user budget while taking into account the fixed prices of Cloud providers and their infinite resources [3]–[6].

This article presents a new scheduling algorithm for Cloud providers that aims to reduce the energy consumption of Cloud providers. To this end, the algorithm attempts to minimize the number of physical machines required to plan a set of workflows (i.e. a workload). This objective (*i.e.*, minimizing the number of machines) replaces the usual makespan minimization (*i.e.*, completion time) of HPC and Grid Computing scheduling algorithms [7]–[9]. Indeed, one of the main operating cost of a Cloud computing provider is the electrical consumption. This consumption can be reduced by limiting unused (idle consumption) or under-used (power and cooling costs) physical machines [10][1]. Such optimization objective for workflows require the combination of consolidation and scheduling algorithms.

Our v-HEFT-*deadline* algorithm introduces a deadline approach for the user. Intuitively, when trying to reduce the makespan of a workflow, it is necessary to use a large number of machines, which is costly for the cloud provider. Indeed, since energy consumption cannot be represented by a linear function [11], [12] defined by the number of machines and the running time, the use of a large number of machines for a very short time is more consuming than the use of a reduced number of machines for a slightly longer time interval. Conversely, if users can extend their deadlines instead of looking for the best possible makespan, a better energy optimization (*i.e.*, number of machines used) can be achieved, resulting in a cost reduction for the Cloud provider. This cost reduction can then be reflected in the rental prices by a business model. One can note that such business model is above the scope of this paper but it seems realistic to, at least partially, pass on the savings made by the Cloud provider to the user in the rental price.

The workflows discussed in this document are highly heterogeneous. In addition to the different library require-

[1] https://www.sallan.org/pdf-docs/McKinsey_Data_Center_Efficiency.pdf

ments, some tasks may need to be performed on different operating systems (OS) in the same workflow. For example, the genomic data stream designed by the ICO in [13]–[15] uses data produced by a vendor-specific machine[2]. To convert the output formats of the machine to standards vendor-specific software developped for Windows must be used, while the other libraries (*e.g.*, Openswath) must be run under Linux. This is a common issue in scientific workflows.

In this article, we consider virtual machines (VMs) for two reasons. On the one hand, the heterogeneity of operating systems and libraries makes it mandatory to load different operating systems on the same machine simultaneously. It can be noted that the management of several operating systems can also be performed on bare metal machines. However, in this case, a complete server will be reserved for each OS, which goes against the consolidation optimization sought. The virtualization mechanism allows several operating systems to be loaded on a single server and reduces the number of servers required. On the second hand, a multi-user workload is considered, thus, strong isolation, for security reasons, must be guaranteed which is made easier by the VMs.

This article presents the following contributions: (1) a detailed model of the scheduling problem under consideration; (2) an adaptation of the HEFT algorithm, namely V-HEFT-*deadline*, that takes into account both virtualization and deadlines and that minimizes the number of physical machines used to plan a workload; (3) a detailed evaluation compared to a VM oriented adaptation of HEFT, namely V-HEFT. The rest of this document is organized as follows. Section II presents the work related to workflow planning algorithms. Then, Section III details the problem modelization, Section IV presents our new algorithm V-HEFT-*deadline* and Section V offers a detailed evaluation of our algorithm compared to the V-HEFT. Finally, the section VI concludes this work and opens some perspectives.

II. Related Work

In this section, we present some noteworthy work on scientific dataflow and workflow scheduling, whether for HPC, Grid and Cloud environments.

On the one hand, high-performance computing (HPC) and Grid computing show an interest in scientific workflow scheduling (heterogeneous, coarse-grained, interconnected tasks). These domains do not take virtualization into account in their scheduling algorithm. Indeed, physical machines are directly considered (Bare Metal). The algorithms for scheduling workflows in this area can be classified into three different groups: *batch schedulers*, *list schedulers* and *pack schedulers*.

In [9], Min-Min and Max-Min are presented. Both are batch algorithms (i.e. independent job scheduling). The Min-Min batch algorithm first creates a list of independent tasks and schedules them, starting its decision process with the task with the lowest execution time (or the highest execution time for Max-Min). Then it deletes the tasks from the dependency graph, creating a set of new independent tasks. These two phases are repeated until all tasks are scheduled. These algorithms have been designed to plan independent task sets; therefore, they are not well suited for workflow and workflow planning, as indicated in [5].

In [7], Sun et al. claim that most HPC scheduling algorithms focus on CPU and core usage and that in practice, the user of the batch scheduler must juggle alone with other node resources, such as memory, requiring larger resources (a complete node, for example). Sun et al. consider all node resources more precisely by studying two algorithms based on list and pack scheduling. First, the list scheduling algorithm schedules the sorted task in a list and tries to minimize the time it takes to complete the submitted tasks (*makespan*). Secondly, the pack scheduling algorithm creates task packets that do not exceed the considered capacity and prohibits any further scheduling until all tasks in a packet are completed.

The HEFT algorithm [8] is a well-known heuristic based on a *list scheduling*. This heuristic is divided into two parts. First, a list is created containing all the tasks of the entire workload sorted by priority. Second, each task is scheduled, one by one, on the available resources while trying to minimize the overall completion time required to execute the workflows (*makespan*). In the scheduling phase of HEFT, a selection of physical resources (node) is made. For this purpose, the task computation time is calculated for each resource. The resource offering the most efficient execution (best execution time) is selected and the task is scheduled on it.

On the second hand, workflows scheduling strategies have also been studied for Cloud computing environments. Typically, [3] focuses on minimizing end-user costs to run scientific workflows in a public Cloud where the scheduling decision is an unknown black box. In [6], [16] a budget must be respected according to a public Cloud offer. Such work assumes that the Cloud provider is always able to meet the customer's needs (infinite resources).

In [4], a workflow planning algorithm is presented. A deadline is taken into account by the scheduling algorithm and a hybrid Cloud environment is considered (i.e. a combination of private and public Clouds). The model takes into account complex tasks, for which the number of instructions and the amount of communication between tasks are known, as well as the capacity of all Cloud resources. The scheduling algorithm is an ad hoc solution where a *list scheduling* is first run on the private infrastructure. If this scheduling cannot meet the user's deadline, the public Cloud is used to obtain more resources. This second part also assumes infinite resources of the public Cloud. The objective of using deadline, in [4] is different from our

[2] https://sciex.com/Documents/Downloads/Literature/Tech-Note-MSMSall-SWATH-Acquisition.pdf

objective. Indeed, deadlines are intended to determine when it is necessary for the user to pay for more resources in a public Cloud.

In [5] an algorithm based on HEFT is presented. This algorithm divides a client's budget by the number of workflows to schedule in a public Cloud environment. Moreover, this paper also extends Min-Min and shows that Min-Min is less effective than HEFT in minimizing makespan. The objective considered in [5] is to respect a user budget. As for [6], [16], a public Cloud environment is considered and the assumption of infinite resources is made.

Finally, in [6] presents a PSO (Particle Swarm Optimization) algorithm for workflow scheduling. In this article, the authors consider a public Cloud, and aim to minimize the cost while respecting user deadlines, and considering a given public Cloud offer (and associated prices).

One can note from the above related-work on dataflows and workflows scheduling that, as far as we know, none of the existing contributions focus on our specific scheduling problem: taking deadlines into account in the internal workflow scheduling of the Cloud provider in order to reduce its energy consumption. First of all, HPC and Grid computing do not take into account virtualization, which significantly modifies algorithms, especially resource modeling. In addition, the objective of these algorithms is to minimize the makespan. Second, in Cloud related documents, the optimization strategy is not designed for the Cloud provider's internal scheduler, which is considered as a black box, but for the end-user costs while considering renting prices for VMs.

We could have chosen any of the above heuristics to solve our scheduling problem. Because of its clear two-phase approach and simplicity, we decided to base our work on the well-known HEFT algorithm in this contribution.

III. Problem modelization

In this section, we present the model that describes our scheduling problem. The problem we aim to solve is to schedule all tasks of submitted workflows on virtual resources while satisfying tasks dependencies and their resource needs, and while respecting the capacity constraints of physical machines. Our objective is to minimize the energy consumption of the Cloud provider by reducing the number of physical machines used to plan workflows.

Execution environment. We denote $\mathcal{J}$, the set of tasks to be scheduled and executed. These tasks make up the different workflows. Each workflow can be represented as a *DAG* - **D**irected **A**cyclical **G**raph $G = (T, D)$, where $T \subset \mathcal{J}$ and D is the set of data dependencies between tasks. Each edge of the graph $d \in \mathcal{D}$ is weighted, and its weight represents the size of the data to be transferred from one task to another. In this article, we assume that the number of instructions to execute a task is given. We are aware that it is difficult to obtain this information accurately in practice. However, this problem is beyond the scope of this contribution and is left to future work. Each task has constraints that can be divided into two categories: the hardware constraints that can be quantified, such as the number of cores, the amount of memory; and the software constraints (OS, library, etc.). The environment in which we want to schedule workflows is a set of clusters of physical machines, with the ability to deploy different types of virtual machines. $\mathcal{V}$ denotes the set of VMs under usage within the infrastructure. Virtual machines are deployed from images, which have different hardware and software capacities. Let $\mathcal{N}$ be the set of compute nodes (physical machines). A node is associated with a given cluster. Bandwidth between nodes is considered heterogeneous, depending on the different clusters they belong. We denote $bw^{\mathcal{N}}_{n,m}$ the bandwith between n and m, for n and $m \in \mathcal{N}$. We denote $speed^{\mathcal{N}}_{n}$ the speed of the node $n \in \mathcal{N}$, in a number of instructions per core, and per instant (*e.g.*, seconds). In this paper, we do not consider any over-provisioning of the nodes; consequently one core is reserved for one VCPU.

Software and Hardware constraints. For each node $n \in \mathcal{N}$, we denote for each instant $t \in \mathbb{N}$, the vector $H_{tn} =< h_{tnv}, \ldots, h_{tn|\mathcal{V}|} >$, where $h_{tnv} = 1$ if and only if the VM $v \in \mathcal{V}$ is hosted by n at instant t, $h_{tnv} = 0$ otherwise. We also denote for each $v \in \mathcal{V}$, at each instant $t \in \mathbb{N}$, the vector $E_{tv} =< e_{tvj}, \ldots, e_{tv|\mathcal{J}|} >$, where $e_{tvj} = 1$ if and only if $j \in \mathcal{J}$ is executed by v at instant t, $e_{tvj} = 0$ otherwise. $\mathcal{C}$ is the set of different hardware capacities (*e.g.*, RAM, CPU, HDD, etc.). For each capacity $k \in \mathcal{C}$, we define three vectors :

- $\mathcal{C}^{\mathcal{N}}_k$, of size $|\mathcal{N}|$, which represents the capacity k provided by each nodes $n \in \mathcal{N}$, such that $\mathcal{C}^{\mathcal{N}}_k(n)$ is the capacity k that $n \in \mathcal{N}$ can supply.
- $\mathcal{C}^{\mathcal{V}}_k$, of size $|\mathcal{V}|$ represents the required amount of resource k needed by each $v \in \mathcal{V}$, such that $\mathcal{C}^{\mathcal{V}}_k(v)$ is both the amount of resource k needed by $v \in \mathcal{V}$, and the amount of resource this VM v can provide.
- $\mathcal{C}^{\mathcal{J}}_k$, of size $|\mathcal{J}|$, defines the amount of resource k required by each $j \in \mathcal{J}$.

For each capacity $k \in \mathcal{C}$, we define the two following constraints in such a way that over-provisioning is not considered both for physical and virtual machines:

$$\mathcal{C}^{\mathcal{V}}_k \cdot H_{tn} \leq \mathcal{C}^{\mathcal{N}}_k(n) \ \forall n \in \mathcal{N}, \ \forall t \in \mathbb{N} \tag{1}$$

$$\mathcal{C}^{\mathcal{J}}_k \cdot E_{tv} \leq \mathcal{C}^{\mathcal{V}}_k(v) \ \forall v \in \mathcal{V}, \ \forall t \in \mathbb{N} \tag{2}$$

We also consider the software requirements of the tasks. Thus, we define $\mathcal{S}$ as the set of software requirements (*e.g.*, OS, library, language, etc.). For each software requirement $s \in \mathcal{S}$, we define that a task that requires the software s must be run on a virtual machine that has the software s.

Temporal dependency constraints. Let $speed^{\mathcal{V}}_{tv}$ be the speed of $v \in \mathcal{V}$ at instant t - let us remind that the speed is in number of instructions per core (here per VCPU). We do not consider over-provisioning, so the deterioration of the VM speed is assumed to be low [17], and considered to be 5% of the host node speed. Thus, $speed^{\mathcal{V}}_{tv} = speed^{\mathcal{N}}_{n} \cdot 0.95$ if and only if $h_{tnv} = 1$. The problem that we tackle is to improve the scheduling of a new set of workflows in terms of energy consumption. Thus, we do not consider the dynamic migration of virtual machines to different resources in this work. We plan to integrate and study this migration in our future work. Therefore, for simplicity in the rest of the paper, the speed of the VM v is time-independent and denoted as follows $speed^{\mathcal{V}}_{v}$. In our model, virtual machines are dynamically provisioned on demand to manage the specific constraints of each task. A VM needs a certain amount of time to start and be ready to perform tasks. Let $start^{\mathcal{V}}_{v}$ the instant when $v \in \mathcal{V}$ initiates its powering on. We define $boot^{\mathcal{V}}_{v}$ as the number of instructions to run before $v \in \mathcal{V}$ is ready for usage. Let $ready^{\mathcal{V}}_{v}$ be the instant when $v \in \mathcal{V}$ is ready to compute tasks, such that $ready^{\mathcal{V}}_{v} = start^{\mathcal{V}}_{v} + \frac{boot^{\mathcal{V}}_{v}}{speed^{\mathcal{V}}_{v}}$. We define the instant when a task can start as follows:

$$\forall i \in \mathcal{J} start_i = \max_{j \in pred_i} (end_j + \max(\frac{d_{ji}}{bw^{\mathcal{N}}_{loc_j, loc_i}}, ready^{\mathcal{V}}_{v})), \tag{3}$$

where v is the VM hosting the task i, loc_i refers to the node hosting the VM of the task $i \in \mathcal{J}$, and $bw^{\mathcal{N}}_{n,m}$ is the bandwidth between two physical nodes.

To explain more precisely why the Equation 3 uses the bandwidth between physical nodes, we must specify that in our model, VMs are only considered as computing resources. As a result, communications between different tasks are directly performed between nodes without a virtualization layer. This assumption made possible an overlap of communications and computations, which improves the quality (*e.g.*, execution time) of workflows executions. Figure 1 illustrates this claim with an example where a given virtual machine of a first user consumes almost all the CPU and RAM resources of a given node, which makes impossible to start another virtual machine of a second user simultaneously without making over-provisionning. We assume that communications require a very low CPU load, so this load is not taken into account when calculating the resource usage ($\mathcal{C}^{\mathcal{N}}_{core}$) of the node. In the first scenario (at the top of the figure), communications are performed within the VMs. In this case, the execution is fully sequential. Indeed, the first VM must be stopped before the second VM can be started. In the second scenario, on the contrary, since communications are made outside the VMs, it is possible to start the second VM while communications are performed for both users. As a result, the total execution time is reduced. Consequently, bandwidth between physical nodes are used in Equation 3

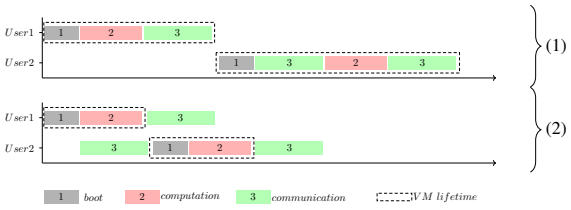

Figure 1: Representation of the gain between communication done on the VMs (1) and on the node (2)

In addition, the execution time of a task i depends on its weight W_i (number of instructions) divided by the speed of the resource that will execute the task.

$$\forall i \in \mathcal{J} exec_i = \frac{W_i}{speed^{\mathcal{V}}_{loc_i} \cdot 0.95} \tag{4}$$

Finally, each workflow $G = (T, D)$ has a *deadline* such that all tasks forming the workflow must be performed before the *deadline*.

Cost modelization. Our contribution aims at minimizing the operational costs of the Cloud computing provider. We assume that this cost is directly correlated to the data center's power consumption. It has already been shown [11], [12] that the CPU's energy consumption is not a linear function defined by the load and the running time. Therefore, reducing the number of nodes for a longer period of time is not equivalent to using more nodes for a shorter period. To properly model the cost and therefore the gain of our solution, the consumption of a node is defined accordingly to [11] and as follows:

$$\sum_{t \in \mathbb{N}} Pmax_n + (\frac{Pidle_n - Pmax_n}{\ln 0.01}) \cdot \ln cpu\_load_{tn}, \tag{5}$$

$$cpu\_load_{tn} = \frac{\sum_{v \in \mathcal{V}} (\mathcal{C}^{\mathcal{J}}_{core}(v) \cdot E_{tv}) \times h_{tnv}}{\mathcal{C}^{\mathcal{N}}_{core}(n)}, \tag{6}$$

where $Pidle_n$ is the idle consumption of the node, $Pmax_n$ is the power consumption of the node when fully used. The cpu_load calculated by Equation 6 is the percentage of cores used on a CPU at a given time. The value 0.01 is an arbitrarily small value to keep the logarithm calculable, and represents the minimal cpu_load.

One can note that we have conducted experiments on the Ecotype cluster (Seduce[3] platform) that perfectly match this model.

Objective. Our model aims at minimizing the number of nodes required to schedule a set of workflows in order to reduce the cost of the Cloud provider.

[3] https://seduce.fr/

IV. V-HEFT-*deadline* ALGORITHM

In this section, we present our deadline aware scheduling heuristic V-HEFT-*deadline* based on V-HEFT, which is similar to HEFT [8] algorithm, but with VM oriented ressource selection phase.

As part of our algorithm, a set of workflows - each associated with a deadline - is submitted at a given time to the Cloud provider. The algorithm is launched following a regular clock and takes into account the VMs already hosted on the Cloud infrastructure.

Priorities and deadlines. In HEFT and V-HEFT, a priority is calculated for each task i of each workflow such that tasks with higher priorities are scheduled first. This priority is established according to the average completion time (on all possible nodes) of a task, denoted $\overline{time_i^{\mathcal{J}}}$, as well as its average communication time (between all possible nodes), denoted $\overline{com_{ij}^{\mathcal{J}}}$. The rank of a task is shown in Equation (7).

$$rank_i = \overline{time_i^{\mathcal{J}}} + \max_{j \in succ_i^{\mathcal{J}}} \left(\overline{com_{ij}^{\mathcal{J}}} + rank_j \right) \quad (7)$$

As illustrated in the main algorithm of V-HEFT in Algorithm 1, once all the tasks are sorted by rank (COMPUTERANKLIST function), the scheduling algorithm is launched. This algorithm will be detailed later in this section.

Algorithm 1 V-HEFT algorithm

```
function HEFT (tasks, nodes)
    task_list ← COMPUTERANKLIST (tasks, nodes)        ▷ Eq. (7)
    for all t ∈ task_list do
        SCHEDULEEARLIEST(t, nodes)
```

The first operation performed by V-HEFT-*deadline*, as indicated in Algorithm 2, is to order the workflows by difficulty (function SORTWORKFLOWS). This difficulty is defined by the difference between the deadline and the average execution time (on all possible nodes) of the critical path of the workflow as follows $deadline - \max_{j \in \mathcal{J}} \left(rank_j \right)$. Then, V-HEFT-*deadline* processes each workflow by decreasing difficulty unlike V-HEFT that directly rank all tasks of all submitted workflows.

Algorithm 2 V-HEFT-*deadline* algorithm

```
function V-HEFT-deadline (workflows, deadlines)
    workflow_list ← SORTWORKFLOWS (workflows, deadlines)
    for all d ∈ workflow_list do
        V-HEFT-deadline-WORKFLOW(d.tasks, d.nodes, deadlines[d])
```

For each workflow to schedule, the function V-HEFT-*deadline*-WORKFLOW shown in Algorithm 3 is called. The first step of this function is to calculate the priority of each task (COMPUTERANKLIST function) of the input workflow The second step is to calculate the deadlines for each task of the current workflow. As already explained, in V-HEFT-*deadline*, the user submits a workflow with a global deadline, including all the tasks that make up this workflow. We also consider (as already explained in Section III) that the number of instructions necessary to perform each task is given when the workflow is submitted to the scheduler. It is therefore possible to calculate a time limit per task that must not be exceeded in order to meet the overall initial deadline of the workflow. Each task deadline will be used to avoid a full exploration of a scheduling that will necessarily lead to an overdue global deadline.

Algorithm 3 V-HEFT-*deadline* single workflow scheduling

```
function V-HEFT-deadline-WORKFLOW (tasks, nodes, deadline)
    task_list ← COMPUTERANKLIST (tasks, nodes)              ▷ Eq. (7)
    dead_list ← COMPUTEDEADLINES (tasks, nodes, deadline)  ▷ Eq. (8)
    ons ← FILTERUSEDNODES(nodes)
    offs ← FILTERUNUSEDNODES(nodes)
    return SCHEDULEBACKTRACK(0, 0, task_list, dead_list, ons, offs)
```

Each task deadline is calculated in the COMPUTEDEADLINES function of Algorithm 3. To calculate this time, it is necessary to start with the final tasks of the workflow that do not have successor. Indeed, for the final tasks, the time not to be exceeded is the overall deadline of the workflow. Then for any other task $i \in \mathcal{J}$ that has successors, the deadline is represented by Equation 8, where $succ_i^{\mathcal{J}} \subset T \subset \mathcal{J}$ is the set of successor of i, and such that $\forall\ j \in succ_i^{\mathcal{J}},\ \exists\ d_{ij} \in D$.

$$\forall i \in \mathcal{J} \text{ where } |succ_i^{\mathcal{J}}| > 0,$$
$$dead_i^{\mathcal{J}} = \min_{j \in succ_i^{\mathcal{J}}} \left(dead_j^{\mathcal{J}} - \min_{n,m \in \mathcal{N}} \left(\frac{W_j}{speed_n^{\mathcal{N}} \cdot 0.95} + \frac{d_{ij}}{bw_{n,m}^{\mathcal{N}}} \right) \right) \quad (8)$$

Finally, the V-HEFT-*deadline* algorithm maintains two lists of physical machines (ons and $offs$ lists) so that the scheduling algorithm knows which physical machines are under usage for tasks or not. The rest of this section details the scheduling algorithm of V-HEFT-*deadline* (and V-HEFT).

Backtrack scheduling algorithm. V-HEFT-*deadline* tries to minimize the number of nodes used to schedule the workflows of a workload. To this end, our solution extends the V-HEFT algorithm with a partial backtracking heuristic. This heuristic consists in trying to perform scheduling on nodes that are under usage by other tasks. As soon as this attempt fails, because the tasks deadlines are no longer met, a backtrack is performed and a new node is considered. This new algorithm is defined in the recursive function SCHEDULEBACKTRACK of Algorithm 4 (initially called in Algorithm 3). This function takes as input the id of the first task (*backTo*), the id of the current task, the task list of the current workflow, the list of tasks deadlines, the list of nodes under usage and the list of unused nodes.

Algorithm 4 is the main part of the V-HEFT-*deadline* algorithm. If the deadline constraint cannot be met by considering nodes under usage only, the algorithm backtracks to the first task that was not scheduled on an unused node

Algorithm 4 V-HEFT-*deadline* backtrack scheduling

```
function ScheduleBackTrack (backTo, id, tasks, deads, ons, offs)
    task ← tasks [id]
    deadline ← deads [id]
    if ScheduleEarliest(task, ons, deadline) then
        if ScheduleBackTrack(backTo, id+1, tasks,
                                    deads, ons, offs) then
            return True
    if id = backTo then
        if ScheduleEarliest(task, ons + offs, deadline) then
            new_ons ← FilterUsedNodes(ons + offs)
            new_offs ← FilterUnusedNodes(offs)
            if ScheduleBackTrack(backTo+1, id+1, tasks, deads,
                                    new_ons, new_offs) then
                return True
    return False
```

given as input, *backTo*. It can be noted that in our scheduling algorithm, the maximum number of backtracks is equal to the number of unused nodes at entry point. As already explained, V-HEFT-*deadline* manages each workflow one after the other by decreasing difficulty, unlike V-HEFT which handles all the tasks of all workflows in a single sorted list. In V-HEFT-*deadline* this cannot be done, because the backtrack part would go back to tasks that have little impact on the task that has not been scheduled. For this reason, if all tasks are processed together, all nodes will be used and a result close to V-HEFT will be observed but with worse complexity. For this reason, V-HEFT-*deadline* schedules the workflows one by one, sorted by decreasing difficulty.

Resource selection. Another essential part of the V-HEFT and V-HEFT-*deadline* algorithms is the resource selection function ScheduleEarliest. This function is called each time a task is to be scheduled, and is detailed in Algorithm 5. Obviously, the V-HEFT-*deadline* ScheduleEarliest function takes deadlines into account, which is not the case for V-HEFT.

Algorithm 5 V-HEFT-*deadline* resource selection

```
function ScheduleEarliest (task, nodes, deadline)
    start ← 0
    bestPlace ← (None, 0, 0, deadline + 1)
                    ▷ A place is a tuple (node, start, duration, end) for a task
                    ▷ deadline + 1 means that the deadline is not respected
    start ← GetMaximumEnd(pred^J_task)
    for all n ∈ nodes do
        exec ← ComputeExecTime(task, n)                    ▷ Eq. (4)
        place ← getPlaceOnNode(n, task, start, exec, deadline)
                                    ▷ Eq. (1, 2)
        if place.end < bestPlace.end then bestPlace ← place
    if bestPlace.node ≠ None then
        resourceProvisioning(bestPlace.node, task,
                                    bestPlace.start, bestPlace.duration)
```

This resource selection phase is specific to the problem modeled in Section III. Due to virtualization and software constraints, virtual machines must be powered up. A VM is powered up for only two reasons: (1) the VMs already used by the owner of the current workflow cannot meet the requirements of the current task to be scheduled (or he does not have one), and for isolation and safety reasons, VMs of other users cannot be used; (2) by starting a new VM the quality of the schedule (makespan) is enhanced.

The function getPlaceOnNode called in Algorithm 5 is used to find a suitable place to perform a task inside an existing or a new VM onto a specific node (physical machine). First, this function looks for a placement on the existing VMs of the user that minimizes the ending time of the task. Then, the function selects a new VM (that takes into account the constraints of the task) and calculates the ending time of the task on this new VM (taking into account its boot time). The place inside a VM that offers the earliest ending time is selected. The placement found inside a VM may extend the lifetime of that VM, thus the function must also know the capacities of the node hosting the VM.

Finally, in Algorithm 5, the function resourceProvisionning reserves the place on the node, updates all capacities information, and eventually stores the new selected VM. One can note that this reservation may be released when the V-HEFT-*deadline* algorithm backtrack on previous task scheduling.

Complexity. The complexity of the function getPlaceOnNode is common to both V-HEFT and V-HEFT-*deadline* algorithms, thus this function is not taken into account for the complexity. The complexity of V-HEFT is a function of the number of tasks in the workflow and the number of nodes on which the scheduling solution will be done. Therefore, its worst-case complexity V-HEFT is $\mathcal{O}(|\mathcal{J}| \times |\mathcal{N}|)$, and is exactly equals to its average-case complexity.

The complexity of our new algorithm V-HEFT-*deadline*, is different due to the partial backtracking. The worst case, is when all the nodes are needed to schedule the workflow and when the backtrack is performed when reaching the last task of the workflow. The worst-case complexity is then $\mathcal{O}(\sum_{i=1}^{|\mathcal{J}|}(|\mathcal{N}|+\sum_{k=1}^{i} k))$. However, this complexity is on average far lower than the worst case, as only under usage nodes will be explored for most of the tasks. Section V validates this claim.

V. Evaluation

In this section, we present a detailed evaluation of our V-HEFT-*deadline* algorithm. All evaluations are performed on realistic workflows generated by the pegasus workflow generator[4] [6].

A. Initial workload scheduling

This section presents the experimental results when considering an homogeneous group of physical machines (*i.e.*, a cluster), and when no existing tasks are already running on the cluster (*i.e.*, initial workload). This simplified use case offers the possibility to precisely analyze the behavior

[4]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

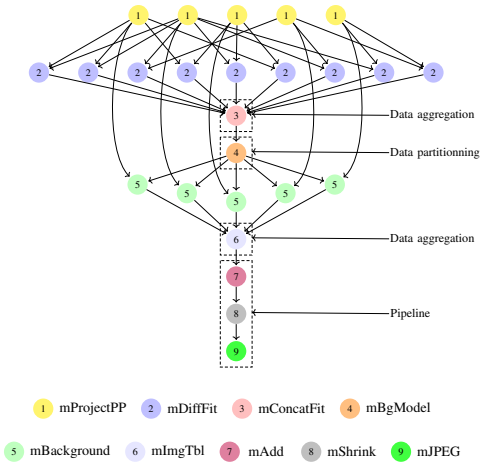

Figure 2: *Montage* workflow

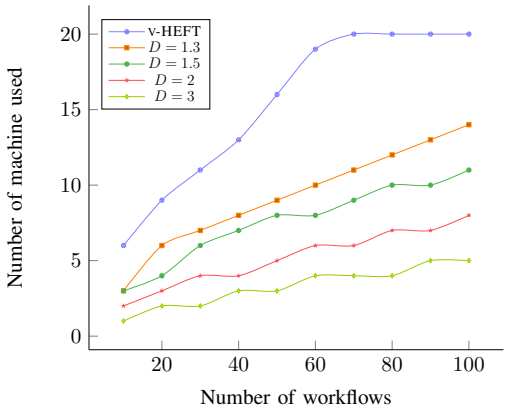

Figure 3: Comparison of the number of nodes used between v-HEFT and v-HEFT-*deadline* (with four different deadlines D).

of v-HEFT-*deadline* without too many parameters to take into account. This section shows that introducing a deadline within v-HEFT is a way to optimize the number of machines used to schedule a set of workflows, thereby reducing the energy consumption of the Cloud provider. This section also shows that the execution time of v-HEFT-*deadline* is very competitive and scalable compared to v-HEFT.

Experimental setup. The *Montage* workflow depicted in Figure 2 is a typical synthetic case-study used to evaluate scheduling algorithms [4], [16]. It is a complex workflow that integrates most of the workflow classes characterized by Bharati et al. in [18]. The simulated workload is composed of a variable number of *Montage* workflows and the simulated infrastructure is composed of 20 homogeneous nodes with the hardware configuration of the Grid'5000 Econome cluster (see the first row of Table I). To correctly select a set of deadlines for our v-HEFT-*deadline* algorithm, we ran v-HEFT to get its approximate minimum makespan.

Using the Pegasus workflow generator, only one information is provided about a task: its execution time (in seconds). We assume in our experiments that this time was produced by a single CPU core with a computing capacity of 2GFlops (floating operations per second). This assumption is used to transform information on the execution time into a number of instructions (as required by our model). We also assume that each *Montage* task has been calculated with a sufficient amount of memory, and we consider in our experiments that the memory requirements are always fulfilled. Furthermore, we consider a single VM template that uses 4 cores. Finally, the boot time of the VM template has been configured to 10 seconds on a 2 GFlops CPU core.

Quality evaluation. Figure 3 represents the number of nodes used, respectively by v-HEFT and v-HEFT-*deadline*, to schedule workloads with variable amounts of *Montage* workflows. v-HEFT-*deadline* has been run with four different deadlines proportional to the makespan computed by v-HEFT: (1) D=1.3, (2) D=1.5, (3) D=2, and (4) D=3. This result illustrates that relaxing the deadline offers more possibilities for v-HEFT-*deadline* to reduce the number of nodes needed by the schedule. As a result, when users can extend their deadlines, The Cloud provider can benefits from an energy consumption reduction. Obviously this cost reduction may be passed on the rental price for the user who then has interest in relaxing her/his deadline. One can note that even by extending the deadline from 1 (v-HEFT) to 1.5 the number of nodes can be divided by 2. When the deadline is extended from 1 to 3, only five of the twenty available nodes are used to schedule the workloads while v-HEFT would use 20 machines.

Energy consumption evaluation. Figure 4 shows estimated immediate power consumption (in Watt) of the nodes for the execution of 100 *Montage* workflows, scheduled by both v-HEFT and v-HEFT-*deadline* algorithm with different deadlines. This consumption has been estimated according to real power consumptions measured on a cluster of Grid'5000 (Ecotype). One can note that, as expected, power consumption is reduced when the deadlines are delayed.

The cost for the Cloud provider is mainly correlated to the energy consumption (Joules). The energy consumption is the sum of power consumptions (per second) during the execution of the workload. As already explained, it has been shown that energy consumption cannot be modeled by a linear function [11], [12] defined by the number of machines and the running time (see Equation 5). As a result, reducing the number of nodes may induce a reduction of the global energy consumption (*i.e.*, the cost) even if the time required to finish the workload is longer. Table II illustrates this claim by showing the results obtained for the four different deadlines in terms of the number of nodes used to schedule the workload, the total makespan needed to run the workload, as well as the energy consumption. In this experiment, unused machines are considered powered

Table I: Description of the simulated nodes

Location	Name	Number of nodes	CPU	Network
Nantes	econome	22	Intel Xeon E5-2660 (Sandy Bridge, 2.20GHz, 2 CPUs/node, 8 cores/CPU)	10 Gbps
Rennes	parapide	21	Intel Xeon X5570 (Nehalem, 2.93GHz, 2 CPUs/node, 4 cores/CPU)	20 Gbps
Grenoble	yeti	4	Intel Xeon Gold 6130 (Skylake, 2.10GHz, 4 CPUs/node, 16 cores/CPU)	10 Gbps

Table II: - Results of the scheduling of complete workload composed of *Montage* workflows.

	V-HEFT	V-HEFT-*deadline*			
Deadline	-	$\times 1.3$	$\times 1.5$	$\times 2$	$\times 3$
Nb nodes	20	14 (70%)	11 (55%)	8 (40%)	5 (25%)
Makespan	157	200 ($\times 1.27$)	233 ($\times 1.48$)	311 ($\times 1.98$)	462 ($\times 2.94$)
Energy (Joules)	333,777	298,352 (89.4%)	282,602 (84.7%)	282,887 (84.8%)	278,266 (83.4%)

off, and nodes are turned off when they have finish their computations. As expected, the makespan increases with the deadline by using V-HEFT-*deadline*, thus the overall time spent to execute the complete workload is longer than V-HEFT. However, the total energy consumption is almost reduced of 17% when using V-HEFT-*deadline* compared to V-HEFT. Therefore, the number of nodes has a direct impact on the energy consumption of the Cloud provider. One can note, though, that the execution with D=2, has a cost a bit higher than the execution with the D=1.5, meaning that the diminution of the number of nodes does not always counterbalance the makespan extension.

Performance evaluation. Figure 5 shows the execution time taken by both V-HEFT and V-HEFT-*deadline* to compute the scheduling solution of the previous experiment. First, one can note that adding a backtrack system in V-HEFT-*deadline* does not deteriorate the efficiency of the algorithm because V-HEFT-*deadline* iterates on already used nodes instead of all nodes for V-HEFT. We can compute the number of scheduling operations performed by each algorithm by using the complexity we introduce in Part IV. In the case of the scheduling of 100 workflows, for V-HEFT algorithm, the number of operations is $j \times n$, with $j = 100$ and $n = 20$, that is to say 2000 operations - by a factor of 25 which is the number of tasks forming the *Montage* workflow. For V-HEFT-*deadline* algorithm, the number of operations depends on the number of nodes under usage when a workflow is scheduled. The number of operations can be estimated by using the values returned by our experiments when scheduling a variable number of workflows (Figure 3). As a result, the number of scheduling operations needed with a deadline $D = 1.3$ is approximatly 930, which is 46.5%, of the number of operations of V-HEFT algorithm. This number is validated by experiments where the time taken by V-HEFT-*deadline* algorithm (1.211 seconds) is half the time taken by V-HEFT algorithm (2.423 seconds).

B. Realistic case study

In this section, we evaluate V-HEFT-*deadline* on a set of more realistic scenarios from both workload and infrastructure perspectives.

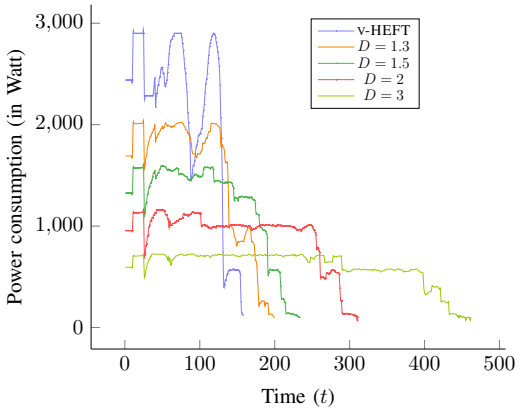

Figure 4: Comparison of immediate power consumption between V-HEFT and V-HEFT-*deadline* (with four different deadlines D).

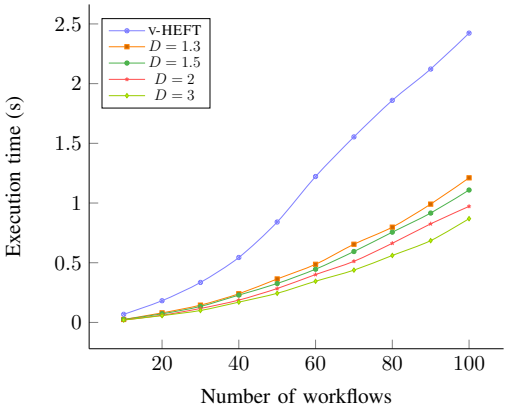

Figure 5: Comparison of execution time between V-HEFT and V-HEFT-*deadline* (with four different deadlines D).

Experimental setup. First, we consider a multi-site infrastructure composed of three different clusters. This infrastructure is a subset of the Grid'5000 testbed that uses the Renater[5] network for communications, which allows inter-cluster communications with a bandwidth of 10 Gbps. Table I gives a description of the three clusters and their

[5]https://pasillo.renater.fr/weathermap/weathermap_metropole.html

associated nodes.

Second, the considered type of workload is more heterogeneous and realistic in this experiment. Table III details the considered workloads composed of five different kind of workflows (all available within Pegasus). We assume that, in a real-case, the percentage of complex workflows is less important than simple ones. Finally, benchmarks also use different deadlines for each kind of workflow. In the scenario A the deadlines are approximately the makespan returned by V-HEFT execution.

In the following experiments, each set of workflows are owned by a specific user, *i.e.*, all the pipeline workflows belongs to the first user, *Montage* workflows belongs to the second one, etc. The four workloads presented in the Table III are submitted twice, meaning that the scheduling algorithm is called two times. Thus, tasks under execution are taken into account in this experiments. The first submission is done at $t = 500$, and the second one at $t = 1500$. Finally, at $t = 0$ half of the nodes are powered on and are half used. This simulated workload ends at $t = 1000$.

Evaluation. Figure 6 shows the estimated immediate power consumption (in Watt) of the nodes for the execution of the different scenarios (A to D). As for the first experiment V-A, the unused nodes are considered powered off and nodes are powered off as soon as they have finished there tasks. Table IV shows the results obtained for the four different scenarios in terms of the number of nodes, the total makespan and the energy consumption needed to run each workload twice. Moreover, the execution time of the scheduling algorithm is given.

As expected, the global makespan of the solution returned by our algorithm is less good than the makespan of the V-HEFT solution. However, the makespan minimization is not the optimization objectives of V-HEFT-*deadline*. One can observe that even in the scenario A, where the deadlines are almost the makespan returned by V-HEFT, the number of nodes used to schedule the workload is reduced. Actually, as V-HEFT minimizes the makespan for each task it favours empty nodes that offers the best makespan even when already used nodes offers almost the same makespan. As a result, the nodes are not used at their full capacity (under used). In contrast, our algorithm will favour nodes that are already used, and will smooth the power consumption of the nodes over the time as we can observe in Figure 6.

In the scenarios B and C, one can note that the number of nodes used to schedule the workload is greater than in the scenario A, despite the deadline relaxing of *Inspiral* workflows. This effect is due to the scheduling at two different times. Indeed, as the first schedule is not aware of the second one, the resources usage of the Inspiral workflows will be smoothed and the local makespan of their execution will exceed the arrival of the second scheduling. This involves a lower number of available resources for the

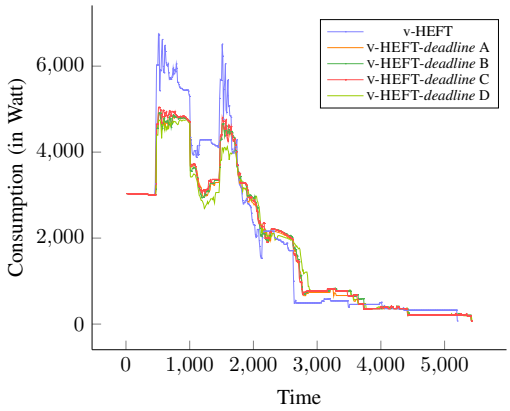

Figure 6: Comparison of the power consumption through time between the different scenari of Table III.

scheduling of the second set of workflows at $t = 1500$, and causes an increase on the number of needed nodes. The last scenario D validates this observation. In this scenario the deadlines of the *Pipeline* workflows are relaxed to give them more time to be executed even if the resources are used by *Inspiral* workflows. As expected, the number of used resources is reduced, and the global makespan delayed.

As in the simple evaluation, the total energy consumption observed is directly correlated to the number of nodes used to schedule the workload. In the scenarios B and C, the energy consumption is a bit higher than in A, for the same reasons than explained earlier. In all cases, the energy consumption is reduced compared to the V-HEFT (Table IV).

VI. Conclusion

This paper tackles the scheduling of heterogeneous scientific workflows while minimizing the energy consumption of Cloud providers. The V-HEFT-*deadline* algorithm has been presented as a solution to this scheduling problem. V-HEFT-*deadline* adds deadlines to workflows so that the number of servers needed to run the workload is reduced, as well as the energy consumption. This algorithm is based on V-HEFT, a variant of HEFT that takes virtualization into account. The V-HEFT-*deadline* algorithm has been compared to V-HEFT to assess the impact of the deadlines on the energy consumption of the Cloud provider. Experiments have been conducted on different case studies. Experiments on realistic workloads have shown a reduction in the number of nodes by up to 28% for an estimated 10% energy consumption reduction for the Cloud provider. In future work, we plan to improve our model to take into account the dynamic migrations of virtual machines as well as a more complex and realistic network model with limited bandwidth. We also plan to study the benefits of our algorithm in the context of power cap contracts between the Cloud computing provider and the energy provider.

Table III: - Percentages and deadlines of the workflows composing the workloads

Name	%	deadlines (A)	deadlines (B)	deadlines (C)	deadlines (D)
Montage	30%	150	150	150	150
CyberShake	10%	1000	1000	1000	1000
Inspiral	6%	1500	2000 ($\times$1.3)	2200 ($\times$1.45)	2200 ($\times$1.45)
Sipht	4%	4000	4000	4000	4000
Pipeline	50%	250	250	250	400 ($\times$1.6)

Table IV: - Results of the scheduling of the different scenari

	v-HEFT	deadlines (A)	deadlines (B)	deadlines (C)	deadlines (D)
Makespan	5202	5422	5422	5422	5431
Nb nodes	47	35 (74%)	35 (74%)	36 (77%)	34 (72%)
Time (s)	27.258	9.601 (35.2%)	9.705 (35.6%)	10.185 (37.4%)	9.338 (34.3%)
Energy (Joules)	9,183,968	8,395,508 (91.4%)	8,487,680 (92.4%)	8,555,411 (93.2%)	8,276,282 (90.1%)

REFERENCES

[1] L. Bertram, A. Ilkay, B. Chad, H. Dan, J. Efrat, J. Matthew, L. E. A., T. Jing, and Z. Yang, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[2] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528 – 540, 2009.

[3] 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*, vol. 113, pp. 1 – 26, 2016.

[4] L. F. Bittencourt and E. R. M. Madeira, "Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207–227, Dec 2011.

[5] Y. Caniou, E. Caron, A. K. W. Chang, and Y. Robert, "Budget-aware scheduling algorithms for scientific workflows with stochastic task weights on heterogeneous iaas cloud platforms," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2018, pp. 15–26.

[6] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, April 2014.

[7] H. Sun, R. Elghazi, A. Gainaru, G. Aupy, and P. Raghavan, "Scheduling Parallel Tasks under Multiple Resources: List Scheduling vs. Pack Scheduling," Inria Bordeaux Sud-Ouest, Research Report RR-9140, Jan. 2018.

[8] H. Topcuouglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[9] J. Yu, R. Buyya, and K. Ramamohanarao, *Workflow Scheduling Algorithms for Grid Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 173–214.

[10] R. Buyya, A. Beloglazov, and J. H. Abawajy, "Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges," *CoRR*, vol. abs/1006.0308, 2010.

[11] W. Wu, W. Lin, and Z. Peng, "An intelligent power consumption model for virtual machines under cpu-intensive workload in cloud environment," *Soft Computing*, vol. 21, no. 19, pp. 5755–5764, Oct 2017.

[12] C. Hsu and S. W. Poole, "Power signature analysis of the specpower_ssj2008 benchmark," in *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*, April 2011, pp. 227–236.

[13] H. L. Röst, G. Rosenberger, P. Navarro, L. Gillet, S. M. Miladinović, O. T. Schubert, W. Wolski, B. C. Collins, J. Malmström, L. Malmström, and R. Aebersold, "Openswath enables automated, targeted analysis of data-independent acquisition ms data," *Nature Biotechnology*, vol. 32, pp. 219 EP –, 03 2014.

[14] H. L. Röst, Y. Liu, G. D'Agostino, M. Zanella, P. Navarro, G. Rosenberger, B. C. Collins, L. Gillet, G. Testa, L. Malmström, and R. Aebersold, "Tric: an automated alignment strategy for reproducible protein quantification in targeted proteomics," *Nature Methods*, vol. 13, pp. 777 EP –, 08 2016.

[15] L. Reiter, O. Rinner, P. Picotti, R. Hüttenhain, M. Beck, M.-Y. Brusniak, M. O. Hengartner, and R. Aebersold, "mprophet: automated data processing and statistical validation for large-scale srm experiments," *Nature Methods*, vol. 8, pp. 430 EP –, 03 2011.

[16] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158 – 169, 2013, including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[17] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin, "Performance evaluation of virtualization technologies for server consolidation," 2007.

[18] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. H. Su, and K. Vahi, "Characterization of scientific workflows," in *2008 Third Workshop on Workflows in Support of Large-Scale Science*, Nov 2008, pp. 1–10.