Received: 31 July 2020 | Revised: 31 August 2021 | Accepted: 25 October 2021

DOI: 10.1002/cpe.6719

SPECIAL ISSUE PAPER

WILEY

# Data-aware and simulation-driven planning of scientific workflows on IaaS clouds

Tchimou N'Takpé[1] | Jean Edgard Gnimassoun[2] | Souleymane Oumtanaga[2] | Frédéric Suter[3]

[1]Mathematics and Computer Science Laboratory, Nangui Abrogoua University, Abidjan, Côte d'Ivoire

[2]Computer Science and Telecommunications Research Laboratory (LARIT), National polytechnic institute Houphouët-Boigny (INP-HB), Abidjan, Côte d'Ivoire

[3]IN2P3 Computing Center, CNRS, Lyon-Villeurbanne, France

**Correspondence**
Tchimou N'Takpé, Mathematics and Computer Science Laboratory, Nangui Abrogoua University, Abidjan, Côte d'Ivoire.
Email: tchimou.ntakpe@gmail.com

## Abstract

The promise of an easy access to a virtually unlimited number of resources makes Infrastructure as a Service Clouds a good candidate for the execution of data-intensive workflow applications composed of hundreds of computational tasks. Thanks to a careful execution planning, workflow management systems can build a tailored compute infrastructure by combining a set of virtual machine instances. However, these applications usually rely on files to handle dependencies between tasks. A storage space shared by all virtual machines may become a bottleneck and badly impact the application execution time. In this article, we propose an original data-aware planning algorithm that leverages two characteristics of a family of virtual machines instances, that is, a large number of cores and a dedicated storage space on fast SSD drives, to improve data locality, hence reducing the amount of data transfers over the network during the execution of a workflow. We also propose a simulation-driven approach to solve a cost-performance optimization problem and correctly dimension the virtual infrastructure onto which execute a given workflow. Experiments conducted with real application workflows show the benefits of the presented algorithms. The data-aware planning leads to a clear reduction of both execution time and volume of data transferred over the network while the simulation-driven approach allows us to dimension the infrastructure in a reasonable time.

**KEYWORDS**

data-intensive workflows, IaaS cloud, makespan reduction, workflow scheduling

## 1 | INTRODUCTION

Scientific workflows constitute an appealing approach to express the complex orchestration of interdependent computations and have become mainstream in many scientific domains.[1] They usually allow users to describe the different steps needed to go from a typically vast amount of data generated by a scientific experiment to the production of an original scientific result. The execution of such data-intensive applications made of hundreds of computational tasks on large scale distributed infrastructures is usually handled by a workflow management system (WMS).[2-4] Tasks such as resource selection, data management, or computation scheduling are delegated to the WMS, hence hiding the complexity of these operations to the end user.

Commodity clusters and computing grids have long been the infrastructures of choice to execute scientific workflows. The institution of the owner of the workflow generally hosts and manages the former, hence easing the access to resources, while the latter allowed scientists to run their workflows at an unprecedented scale by aggregating resources from multiple institutions. With the support of major companies such as Amazon,[5] Google,[6] or Microsoft,[7] Infrastructure as a Service (IaaS) clouds have become serious contenders to clusters and grids. Indeed, IaaS clouds combine their respective advantages by providing an easy access to a virtually unlimited amount of resources. Thanks to a careful *planning* of a workflow

execution, a WMS can thus build a compute and storage infrastructure specifically adapted to this particular workflow on demand, by combining a specific set of virtual machine instances.

The description of a scientific workflow is generally independent of the characteristics of the infrastructure onto which it will be executed. This offers a greater flexibility to users who can, through the WMS, run the same workflow on different infrastructures without any change in their application. A direct consequence of this flexibility is that dependencies between computational tasks, that is, a data produced by a task is consumed by another, are generally handled with files. The produced intermediate data is written on disk, the file is then potentially transferred over the network to another storage device where the consuming task will eventually read it.

In this article, we propose to leverage two characteristics of a family of virtual machines instances provided by Amazon Web Services,[5] that is, a large number of cores and a dedicated storage space on fast SSD drives, to improve data locality, hence reducing the amount of data transfers over the network during the execution of a workflow. This approach should have a direct and beneficial impact on the execution time of the workflow. Then, we propose to rely on realistic simulations to dimension a cloud infrastructure that leads to a good cost-performance tradeoff. To this end, we developed a simulator on top of the WRENCH framework.[8] WRENCH can be used to build simulators of WMSs that are accurate, can run fast and scalably on a single computer, and be implemented with minimal software development effort. The main contributions of this article thus are:

- An original data-aware planning algorithm that aims at minimizing the amount of data transfers over the network during the execution of a workflow given a specific set of virtual machine instances.
- A simulation-driven approach to solve a cost-performance optimization problem and correctly dimension the virtual infrastructure onto which execute a given workflow.

This article is organized as follows. In Section 2, we describe the platform and application models used in this article. Then, in Section 3, we detail the proposed data-aware planning algorithm while Section 4explains how simulation is used to determine the most efficient set of virtual machines to allocate. We evaluate the performance of the proposed algorithms in Section 5. Section 6 reviews the related work on scheduling of scientific workflows on IaaS clouds. Finally, we conclude this article and present future work directions in Section 7.

## 2 | PLATFORM AND APPLICATION MODELS

In this article, we base our platform model on a typical IaaS cloud setup. Multiple virtual machine (VM) instances are deployed on physical servers within a single datacenter. More precisely, we consider a set of VMs similar to the Amazon EC2 M5 instances.[9] More precisely, we consider M5d instances that come with a local storage space on NVMe SSD drives while the regular M5 instances have to rely on the Amazon Elastic Block Storage (EBS) service to store data. Table 1 details the characteristics of the available M5d instances. The indicated costs in dollars per hour correspond to on-demand Linux instances in the US-East region (Ohio) at the time of writing of this article.

The number of virtual cores (vCPUs) in this instance series ranges from 2 to 96, with a constant amount of memory per core of 4 GiB. These instances are typically deployed by Amazon on nodes featuring a Intel Xeon Platinum 8000 series processor. The specific feature of the M5d instances is to attach a fast block-level storage on SSD drives that is coupled to the lifetime of the instance. In this work, we aim at leveraging this fast storage that is shared by the vCPUs of an instance but dedicated to them to store the intermediate files produced during the execution of a workflow, hence reducing the number of data transfers over the network for tasks scheduled on the same virtual machine. Only the *entry* and *exit* files of the workflow will be stored on an external storage node.

**TABLE 1** Characteristics of the AWS M5d instances.

| Model | vCPU | Memory (GiB) | Instance storage (GiB) | Network bandwidth (Gbps) | EBS bandwidth (Mbps) | Cost ($ per hour) |
|---|---|---|---|---|---|---|
| m5d.large | 2 | 8 | 1 × 75 NVMe SSD | Up to 10 | Up to 3500 | 0.113 |
| m5d.xlarge | 4 | 16 | 1 × 150 NVMe SSD | Up to 10 | Up to 3500 | 0.226 |
| m5d.2xlarge | 8 | 32 | 1 × 300 NVMe SSD | Up to 10 | Up to 3500 | 0.452 |
| m5d.4xlarge | 16 | 64 | 2 × 300 NVMe SSD | Up to 10 | 3500 | 0.904 |
| m5d.8xlarge | 32 | 128 | 2 × 600 NVMe SSD | 10 | 5000 | 1.808 |
| m5d.12xlarge | 48 | 192 | 2 × 900 NVMe SSD | 10 | 7000 | 2.712 |
| m5d.16xlarge | 64 | 256 | 4 × 600 NVMe SSD | 20 | 10,000 | 3.616 |
| m5d.24xlarge | 96 | 384 | 4 × 900 NVMe SSD | 25 | 14,000 | 5.424 |

The network bandwidth with other instances or the EBS depends on the size of the instance. We assume that only the largest instances that can exploit a full node, that is, with 48, 64, or 96 vCPUs, have a guaranteed network bandwidth of 10, 20, and 25 Gbps respectively. For smaller instances, that is, from 2 to 32 cores, we consider the available bandwidth be proportional to the number of cores and equal to 208.33 Mbps per core. All the virtual machine instances started for the execution of a given workflow are connected through a single switch.

According to the description of the M5d instances, the connection from a VM to the EBS goes through a dedicated network link, which is taken into account in our simulated infrastructure. As for the network connections between VMs, we assume the bandwidth of the dedicated connection between a VM and the EBS be proportional to the number of cores for small VMs with up to 32 cores (i.e., 218.75 Mbps per core).

In Reference 10, the authors showed that the bandwidth depends on the file sizes, number of files, instance types, and so forth. But our simulations assume a good QoS and the respect of the performance characteristics for the resources allocated by the cloud provider.

The scientific workflows to schedule are represented by Directed Acyclic Graphs (DAGs) $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{v_i | i = 1, \dots, V\}$ is a set of vertices representing the computational tasks of the workflow and $\mathcal{E} = \{e_{i,j} | (i,j) \in \{1, \dots, V\} \times \{1, \dots, V\}\}$ is a set of edges between vertices, representing either a data dependency, that is, a file transfer, or a flow dependency between two tasks. We focus on workflows that comprise a large number of sequential tasks, that is, running on a single core, which are representative of real scientific applications.[11] Each of the task composing the workflow has a predefined (estimated) duration, requires a set of *input files* to start its execution, and will produce a set of *output files* upon completion. We thus denote as $Input_i^k$ (resp. $Output_i^k$), the $k^{th}$ input (resp. output) file of a given task $v_i$. When an output file produced by a task $v_i$ is consumed as input by another task $v_j$, this creates a data dependency between $v_i$ and $v_j$, represented by the edge $e_{i,j}$. The input files that are not produced by any of the tasks in the workflow are called the *entry files* of the workflow. Conversely, the output files that are not consumed by any task are called the *exit files* of the workflow. Finally, we define two quantities associated to each task composing the workflow that will be used during the scheduling process. We define the *Local Input Volume* of task $v_i$ on machine $M_j$, or $LIV_{i,j}$, as the sum of the sizes of the files that $v_i$ takes as input that are locally stored on $M_j$. Respectively, we define the *Local Output Volume*, or $LOV_{i,j}$ as the sum of the sizes of the files produced by $v_i$ that are used by successors of $v_i$ also scheduled on $M_j$. Note that if a file is used by more than one successor, its size is accounted for as many times as successors. The *LIV* (resp. *LOV*) of an entry (resp. exit) task is by definition set to zero.

During the execution of the workflow, all the intermediate files, that is, those that are produced by a task and consumed by another, will be stored locally on the SSD storage of one or several machines. Only the entry and exit files of the workflow will be stored on the EBS service accessible by all the machines. The time to transfer a file from one machine to another includes the time to read the file on the disk of the source machine, the duration of the data transfer over the network and the time to write the file on disk at destination.

The execution time of a workflow, also named *makespan*, is the difference between the actual finish time of the last executed task and the actual start time of the first executed task. The actual start time of a task $v_i$ (or $AST(v_i)$) is the time when $v_i$ starts the transfer of its input files. The actual finish time of a task $v_i$ (or $AFT(v_i)$) is the time when $v_i$ finishes the transfer of its output files. So the *makespan* of a workflow is defined as:

$$makespan = \max_{v_i \in \mathcal{V}} \{AFT(v_i)\} - \min_{v_i \in \mathcal{V}} \{AST(v_i)\}.$$

We assume that each used VM is billed per second. The cost associated to the execution of the workflow is thus defined as:

$$cost = cost\_per\_core \times total\_cores \times makespan,$$

where *cost_per_core* is the unit cost and *total_cores* is the total number of cores used to execute the workflow.

## 3 | DATA-AWARE PLANNING OF WORKFLOW TASKS

In this section, we assume that the provisioning of $m$ virtual machine instances has already been done. How $\mathcal{M}$, this set of instances, is defined will be explained in Section 4. The objective of the proposed planning algorithm is to schedule the set $\mathcal{V}$ of $V$ tasks composing the workflow on $\mathcal{M}$ while leveraging two main characteristics of the target IaaS cloud platform, that is, multicore instances and a fast local storage space, to minimize the impact of data transfers on the execution of data-intensive scientific workflows.

Algorithm 1 starts by building a sorted scheduling list that contains all the tasks in the workflow (lines 1–2). The tasks are sorted by decreasing bottom level value.[12] The *bottom level* of a task $v_i$, or $bl_i$, is the length of the longest path from $v_i$ to the end of the workflow. It sums the estimated duration of all the tasks on this path, including $v_i$. As in Reference 13 we include an estimate the data transfers cost in the computation of the bottom level values of the tasks. This ordering gives the highest priorities to the most critical tasks and ensures the respect of the dependencies between tasks. Then, the algorithm determines a first mapping for each task $v_i$ in $\mathcal{V}$ (lines 3–8). The selected machine $M_j$ in $\mathcal{M}$ is the one that: (i) minimizes the start time of $v_i$; and (ii) maximizes the volume of the locally stored input files for $v_i$. The rationale is that between two virtual machines able to start $v_i$ at the same time, we favor the one that minimizes the amount of data transfers over the network.

**Algorithm 1.** Mapping workflow tasks

1: Compute $bl_i$ of each task $v_i$
2: Sort $\mathcal{V}$ by decreasing $bl_i$ values
3: **for all** $v_i \in \mathcal{V}$ **do**
4: $M \leftarrow \{M_j \in \mathcal{M} \mid st_j(v_i)$ is minimal **and** $LIV_{i,j}$ is maximal$\}$
5: Map $v_i$ on $M$
6: Update the *usage profile* of $M$
7: **end for**
8: **for** $l = L$ to 0 **do**
9: $V_l \leftarrow$ tasks in level $l$ sorted by decreasing $bl$ values
10: rearrange($V_l$) ▷ see Algorithm 2
11: **end for**

As all the considered virtual machine instances have multiple cores, scheduling a task $v_i$ on a machine $M$ implies to maintain a local schedule inside the virtual machine. In order to maximize the utilization of the cores within a virtual machine, we manage each machine as a job and resource manager will do. In particular, we leverage the available information on the (estimated) duration of each task to implement a conservative backfilling mechanism[14] when building the local schedule. Keeping such a *usage profile* of a virtual machine up to date is mandatory to determine the time when a new task can start on this particular machine (i.e., $st_j(v_i)$). Then, after selecting $M$ for the execution of $v_i$, we update the usage profile of $M$ (line 7). The usage profiles of the virtual machines are also used in the second step of Algorithm 1 in which we rearrange the tasks in this initial schedule to further reduce the amount of data transferred over the network.

This rearrangement step (lines 9–12) browses the workflow *level by level* from bottom to top. The rationale is that during the initial placement that proceeds from top to bottom, only the volume of data coming from the direct predecessors of a task is taken into account. It is indeed impossible to account for the locality of the data needed by a direct descendant of a task when scheduling it at its placement is not determined yet. This may lead to avoidable data movements.

We define as *level 0* the topmost level of the DAG that comprises all the entry tasks of the workflow. For each of the others tasks, we recursively compute its level as the maximum level of its predecessors plus one. Finally, we denote as $L$ the number of levels in the workflow. The principle of the rearrangement step is described in Algorithm 2.

We start by saving the current start time ($st^c(v_i)$) and mapping ($M^i$) for each task $v_i$ in $V_l$ (lines 2–3). Then, we determine the local volume $LV_{i,j}$ for task $v_i$ on machine $M_j$ (lines 4–6). We also save the local volume for the current mapping of $v_i$ (line 7) before canceling this mapping (line 8).

Canceling the mapping of all the tasks in a given level creates some idle slots in the usage profiles of different machines that can be used to improve data locality by "migrating" some tasks from one machine to another. The conditions to migrate a task $v_i$ from its former mapping to a new mapping on $M_k$ are that it would improve the data locality, that is, $LV_{i,k} \geq LV_i^c$, and reduce the starting time of the task, that is, $st_k(v_i) \leq st^c(v_i)$.

The main loop in Algorithm 2 (lines 11–33) aims at iteratively improving the mappings for tasks in $V_l$. At each step, we first try to find a better mapping (lines 15–21) for each task by considering the machine that leads to the greatest increase of the local volume first. If the task can also start earlier on this machine, it is selected for a new tentative mapping. There are three exit cases to this while loop: (i) there exists a better mapping for $v_i$ on another machine $M_j$; (ii) $v_i$ has been remapped on the same machine $M_i$ with a better or equal start time; or (iii) no better mapping was found. This last case means that a task with a higher priority has been mapped on $M_i$ and $st^c(v_i)$ can no longer be guaranteed. In both cases, $v_i$ is set back to its original mapping, which becomes definitive (lines 22–26). However, this decision may invalidate some of the migrations (e.g., the task with higher priority mapped on $M_i$). Then, we cancel all the tentative mappings determined in this step (lines 28–32) and look for another rearrangement of the remaining tasks.

Algorithm 2 ends when only migration decisions are taken during the current step. The level is then considered as fully rearranged and the decided mappings become definitive.

The most significant operation in Algorithm 1 is the computation of the respective start times of the tasks. Then, given a workflow composed of $V$ tasks and a platform made of $m$ VMs, the average complexity of our algorithm is $O(V.m)$. In the first step (lines 1–7), the loop (lines 3–7) performs $V$ iterations. At each iteration, we compute $m$ start times for the current task $v_i$. The complexity of this step is thus $O(V.m)$. In the rearrangement step (lines 8–11), at each iteration, we compute in average $V.m/L$ start times, where $L$ is the number of levels in the workflow. Then the complexity of the rearrangement step is also $O(V.m)$.

**Algorithm 2.** Rearrangement of tasks at level $l$

**for all** $v_i \in V_l$ **do**
  $st^c(v_i) \leftarrow$ current start time of $v_i$
  $M^i \leftarrow$ current mapping of $v_i$
  **for all** $M_j \in \mathcal{M}$ **do**
    $LV_{i,j} \leftarrow LIV_{i,j} + LOVi,j$
  **end for**
  $LV_i^c \leftarrow$ current local volume of $v_i$
  Cancel the current mapping of $v_i$
**end for**
level_is_rearranged ← FALSE
**while** ¬ level_is_rearranged **do**
  level_is_rearranged ← TRUE
  **for all** $v_i \in V_l$ **do**
    sort $\mathcal{M}$ by decreasing $LV_{i,j}$ values
    **while** $LV_{i,j} \geq LV_i^c$ **do**
      **if** $st_j(v_i) \leq st^c(v_i)$ **then**
        Map $v_i$ on $M_j$
        Update the *usage profile* of $M_j$
        **break**
      **end if**
    **end while**
    **if** $v_i$ is mapped on $M^i$ **or**
      $st_{M^i}(v_i) > st^c(v_i)$ **then** ▷ No better mapping
      $V_l \leftarrow V_l \setminus \{v_i\}$ ▷ Mapping is definitive
      level_is_rearranged ← FALSE
    **end if**
  **end for**
  **if** ¬ level_is_rearranged **then**
    **for all** $v_i \in V_l$ **do**
      Cancel the current mapping of $v_i$
    **end for**
  **end if**
**end while**

## 4 | SIMULATION-DRIVEN SEARCH OF A COST-PERFORMANCE EFFICIENT SET OF VMS

The data-aware planning produced by Algorithms 1 and 2 minimizes the amount of data transferred over the network during the execution of the workflow. However, the quality of that planning strongly depends on the set of multicore virtual machines given as input.

In this section, we present how to determine a set of virtual machines that achieves a good tradeoff between the execution time of the workflow and the number of resources to rent to an IaaS cloud provider. To this end, we wrote a simulator based on the WRENCH project,[8,15] a CyberInfrastructure simulation framework that provides high-level simulation abstractions for building accurate and scalable full-fledged simulators with minimal software development efforts. WRENCH is an open-source C++ library composed of two layers: the *core* simulation models and base abstractions (computing, communicating, storing) are provided by SimGrid[16] on top of which *services* to simulate the execution of computational workloads (compute services, storage services, network proximity services, data location services, etc.) are defined. By leveraging SimGrid's accurate models and their scalable implementations, WRENCH simulators can yield nearly identical behaviors when compared to actual systems.

Thanks to WRENCH, the time needed to build a data-aware planning and simulate its execution on a given set of virtual machine instances is very low. The experiments detailed in the next section show that the simulation time ranges from 2.28 to 50.88 s, with an average of about 13.4 s. As the size of the workflow is fixed, the simulation time directly depends on the number of cores in the virtual infrastructure. For a given total number of cores in the simulated infrastructure, we treat the selection of the set of instances as a variation of the change-making problem, and use a simple greedy algorithm for that.

Consequently, running an exhaustive set of simulations spanning all the possible numbers of cores from a minimal of two (i.e., the smallest VM size) to one thousand (i.e., the number of tasks) and extracting the set of configurations on the Pareto front could be done in an affordable time (less than an hour in our experiments), but that may also be greater than the actual makespan of the workflow. Then, we propose a strategy to reduce the number of simulations needed to return a narrow set of Pareto solutions to help the user to dimension the infrastructure.

We start by simulating the execution of the workflow on a configuration with as much cores as there are tasks composing it. This run on an obviously oversubscribed configuration allows us to determine $c_{max}$, the maximal number of cores that can be concurrently exploited. Then, we determine two important lower bounds. First, we simulate the execution of the workflow on $c_{max}$ cores in order to get an approximation of the minimum achievable execution time, $E_{c_{max}}$. Second, we simulate the execution of the workflow on only two cores to get an approximation of the minimum cost, $C_2$ (i.e., the price to pay for a 2-core instance for the duration of the execution of the workflow). We use these two values to determine what could be good candidate configurations for the execution of the workflow. Indeed, we arbitrarily assume that a good configuration should not degrade $E_{c_{max}}$ *and* $C_2$ by more than a factor two or, more formally:

$$E_i \leq 2 \times E_{c_{max}} \text{ and } C_i \leq 2 \times C_2,$$

where $E_i$ and $C_i$ respectively are the execution time and cost of an execution on a configuration with $i$ cores.

To identify the candidate configurations, we perform a binary search that stops either when there are no more configurations to test or when we found two consecutive solutions that respect the condition above. During this binary search we also save the number of cores that correspond to the first candidate solution found. It forms a second search space with the number of cores for which the binary search stopped. Then we exhaustively simulate the execution of the workflow for all the number of cores in this second search space.

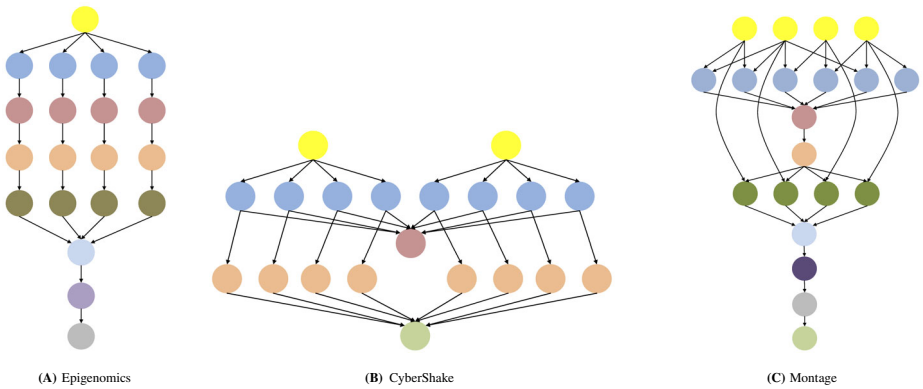
Finally, we select the set of configurations to return to the user by determining the nondominated solutions that compose the Pareto front among the results of the exhaustive search. The two extreme configurations with two and $c_{max}$ cores are also returned to the user.

## 5 | EXPERIMENTAL EVALUATION

To evaluate our different contributions, we consider three workflow applications that are part of the Pegasus workflow gallery.[17] More precisely we use synthetic workflows, resembling those used by real world scientific applications but with a larger task count, that are generated by the workflow generator toolkit.[18] The main characteristics of these applications are given in Table 2 and their structure depicted in Figure 1.

**TABLE 2** Some characteristics of used workflows.

| Workflow | #tasks | Input files size (GB) | Total files size (GB) |
|---|---|---|---|
| CyberShake | 1000 | 150.76 | 400.39 |
| Epigenomics | 997 | 1217.72 | 1230.93 |
| Montage | 1000 | 0.65 | 17.32 |



(A) Epigenomics (B) CyberShake (C) Montage

**FIGURE 1** Structure of the used workflows (http://pegasus.isi.edu)

- **Epigenomics:** is a data processing pipeline to automate the execution of various genome sequencing operations;
- **Cybershake:** is an application of the Southern California Earthquake Center to characterize earthquake hazards;
- **Montage:** is an astronomy application that creates custom mosaics of the sky from multiple images.

The developed simulator relies on WRENCH 1.5-83d60eb which in turn depends on SimGrid 3.23.3-f2ae928. In our simulations, we did not introduce stochastic parameters and we did not consider evolutionary algorithms which require multiple runs. Thus, we did not repeat any simulation with the same input.

## 5.1 | Impact of virtual machine instance sizes

We start this evaluation by assessing the impact of the size of virtual machine instances on execution time. For each workflow, we consider infrastructures made of different numbers of physical hosts, each having 96 cores. For each number of physical hosts, we then enforce the use of a given size of instance among those described in Table 1. We also consider an additional setting in which only one core is used per VM and all the files are stored on the default Elastic Block Storage. This configuration serves as a baseline and corresponds to simply scheduling the workflows with a classical list scheduling heuristic.

Figure 2 shows the evolution of the execution time of the studied workflows when we increase the number of physical hosts in the computing infrastructure and make the size of the deployed instances vary. Here, we do not apply the rearrangement step of the offline planning made with Algorithm 2 to measure the performance of the initial offline planning only.

We observe different behaviors for each of the three considered workflows. First, the number of physical hosts used has almost no influence on the execution time for the CyberShake application while for Epigenomics, we observe a plateau from three physical hosts. For Montage, the execution time decreases up to seven physical hosts. This evolution of the execution time is directly related to the level of parallelism a workflow can exploit, that is, how many tasks can be executed concurrently.

Second, we see that the execution time decreases when the size of the virtual machine instances grows, but that the improvement becomes very limited for sizes above 32. More interestingly, we observe that for the CyberShake application, which produces much more intermediate data than

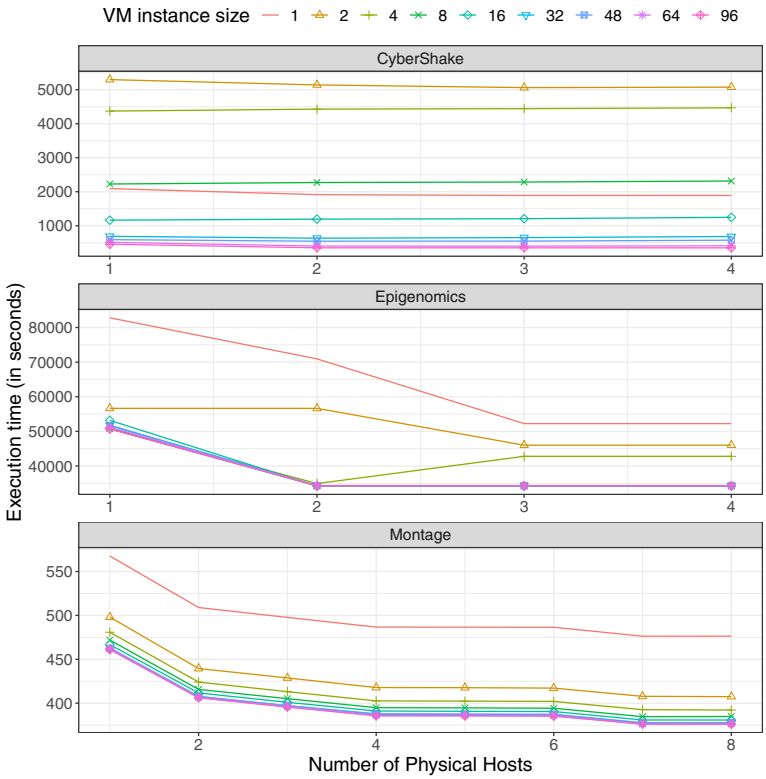



**FIGURE 2** Evolution of workflow execution time with the size of VM instances and number of physical hosts. The offline planning has not been rearranged

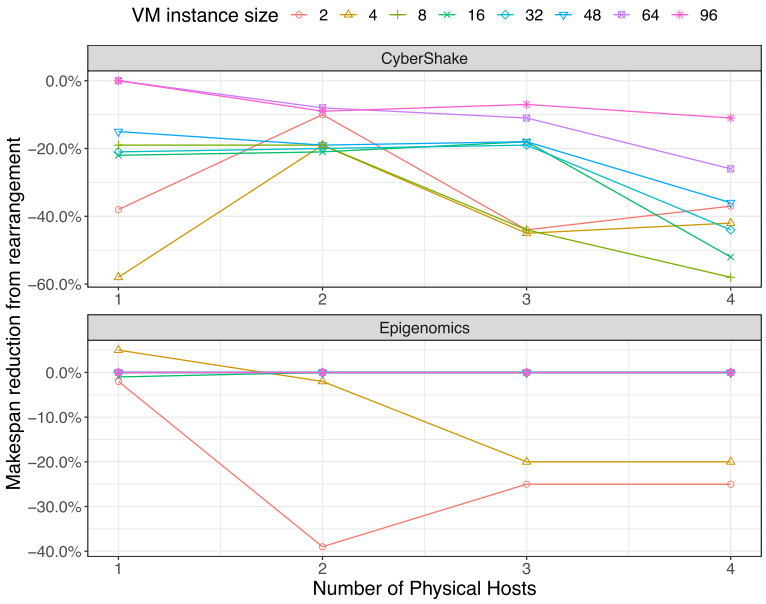



**FIGURE 3** Impact of the rearrangement step on the execution time for different VM instance sizes

the two other workflows, relying on the local storage of small instances (i.e., with up to eight cores) leads to execution times worse than the solution with one core per VM where all the intermediate data are stored on the EBS service. This is because using too many small VMs on a single host (i.e., up to 48 instances with two cores) increases the number of data transfers between instances and cause contention on the network. Conversely, in the baseline configuration, each VM benefits of a dedicated network connection to the shared storage service.

## 5.2 | Impact of the rearrangement step and comparison with HEFT

Figure 3 shows the impact of the rearrangement step on the execution time. For the Montage application, the structure of the workflow is such that rearrangement has no influence on the offline planning hence neither on the execution time. For the two other workflows, we can see that rearranging the offline planning to further reduce the amount of transfers over the network can only improve the execution time. For the Epigenomics workflow , when the size of all the VM instances is 2, we can see a sudden makespan reduction of 38%. Looking in detail at the results (i.e., to the total amount of file transfers), this important gain can be explained by the fact that the rearrangement step avoid more files transfers for this platform configuration.

The observed improvement is more significant for small instances sizes. Changing the mapping of some tasks to favor local storage effectively reduces the number of transfers over the network and thus the contention effects seen in Figure 2. For larger instance sizes the impact of rearrangement is less important but still significant (above 5%) for CyberShake.

Figure 4 shows a comparison of our algorithm including the rearrangement step (Algorithm 1) with the popular algorithm HEFT[13] which also aims to minimize the execution times of workflows. The results are similar to those of the comparison of our algorithm with the version without the rearrangement step. It should be noted that HEFT and the first step of our algorithm are quite similar. The difference is that in our algorithm, we do not take into account the communication costs when we compute start times. Thus these results lead to the same conclusions as the previous ones. HEFT is outperformed when communication times are not negligible because in Reference 13, the authors assumed that the target platform consists of a set $q$ heterogeneous processors connected in a fully connected topology in which all communications are performed without contention. HEFT is therefore not suitable for all topologies of platforms.

## 5.3 | Evaluation of the Pareto front solutions

The main objective of our algorithm is to help IaaS cloud users to find a good compromise between the execution time and the execution cost of their workflow by selecting sets of VM instances on a Pareto front. In this section, we evaluate the quality of the Pareto front solutions produced by our algorithm. In the previous sections, we showed that to minimize the execution time for a fixed number of cores, the priority should be given to large

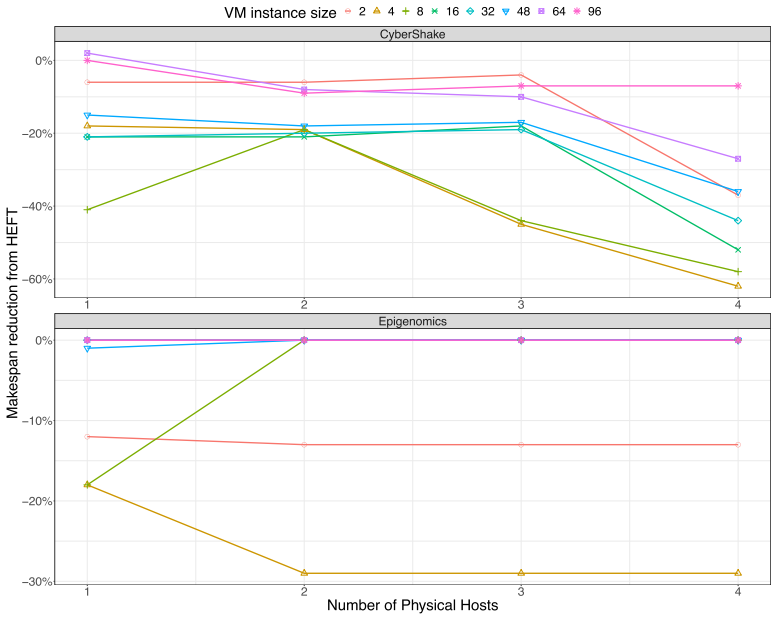



**FIGURE 4** Comparison with HEFT for different VM instance sizes

VM instances. To get the Pareto front given by our algorithm, we thus only have to run one simulation for each total number of cores and discard all the dominated solutions.

We compare this Pareto fronts to that obtained by a modified version of the MOHEFT algorithm proposed in Reference 19. The authors of MOHEFT showed that their algorithm gave better results compared to competing algorithms. However, MOHEFT is originally designed such that each task in the workflow uses all the cores of the VM on which it is scheduled. We modified MOHEFT so that a VM instance can be shared by multiple sequential tasks.

MOHEFT starts by sorting all the tasks in the workflow by decreasing bottom level values. At each iteration $i$, MOHEFT builds $P_i$ solutions as extensions of the $P_{i-1}$ nondominated solutions obtained at the previous iteration, by taking the $i$th task into account. Among these $P_i$ combinations, the algorithm keeps at most $K$ solutions ($K \leq P_i$). This prevents a combinatorial explosion of the number of intermediate solutions found at the end of each iteration. If more than $K$ solutions were produced, MOHEFT sorts the remaining solutions by decreasing crowding distances[20] and selects only the $K$ first solutions.

In our modification of MOHEFT where VMs are shared by multiple tasks, we may produce equivalent solutions. Two solutions $S_i$ and $S_j$ are equivalent if the same tasks are mapped together in differents but equivalent VMs with the same start times. At each iteration, only one represent of such equivalent solutions is kept. Moreover, in all but last iterations, we only eliminate the strictly dominated solutions. A solution $S_i$ is strictly dominated by a solution $S_j$ if both the execution time and cost of $S_i$ are strictly higher than those of $S_j$. This prevents us to eliminating intermediate solutions which can lead to good schedules. In the last iteration we eliminate all the simply dominated solutions (i.e., only one metric needs to be higher for the dominated solution).

Figures 5–7 show the Pareto fronts produced by our algorithm compared to those produced by MOHEFT, after simulations. Despite a high $K$ value (We take $K = 2000$) and a offline planning computation time of about ten hours for each workflow, MOHEFT's Pareto fronts are often dominated by those of our algorithm. Moreover, MOHEFT leads to less diversified solutions despite the use of crowding distances to select the best candidates. Note that the offline planning of MOHEFT takes as input, in addition to the workflow, a set of diverse VMs (i.e., a big platform) from which we can extract various platforms, and at most (and very often) $K$ solutions are produced in output to be simulated. In Reference 19, the authors showed that the complexity of MOHEFT is $O(K.V.m)$, where $m$ is the number of VMs in the big platforms and $V$ is the number of tasks in the workflow. While MOHEFT leads to about $K$ simulations, our algorithm leads to $c_{max}/2$ simulations.

## 5.4 | Determination of a cost-performance efficient set of virtual machine instances

In what follows, we illustrate the functioning of the proposed simulation-driven determination of a cost-performance efficient set of virtual machine instances. Table 3 shows the outcomes, for each of the considered workflows, of the first three simulations that aim at determining the maximum

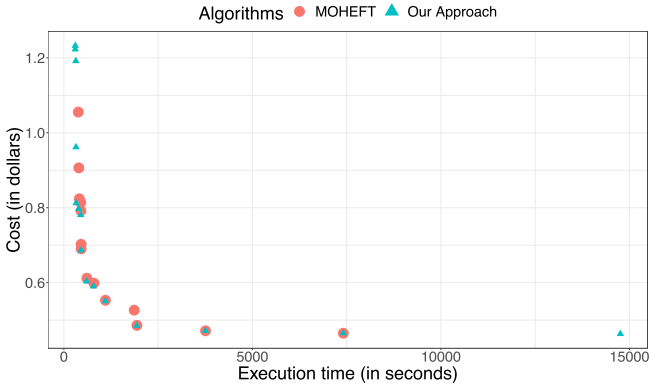



**FIGURE 5** Pareto fronts for CyberShake

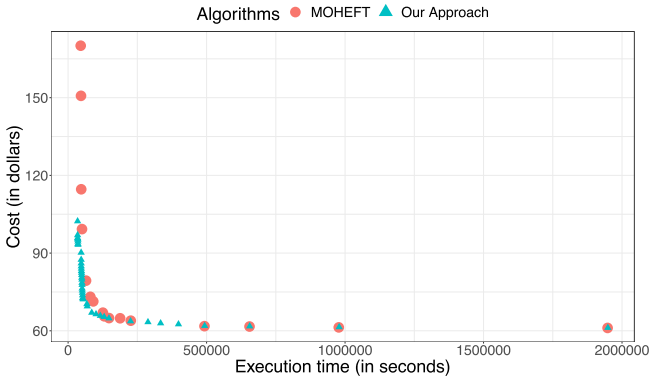



**FIGURE 6** Pareto fronts for Epigenomics

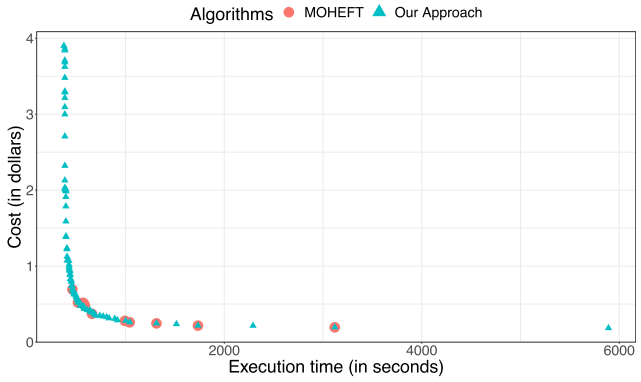



**FIGURE 7** Pareto fronts for Montage

**TABLE 3** Outcomes of the first three simulations used to set up the simulation driven determination of a cost-performance efficient set of instances

| Workflow | $c_{max}$ | $E_{c_{max}}$ | $C_2$ |
|---|---|---|---|
| CyberShake | 374 | 310.266s | $0.463 |
| Epigenomics | 246 | 9h 31m 54s | $61.17 |
| Montage | 662 | 375.45s | $0.185 |

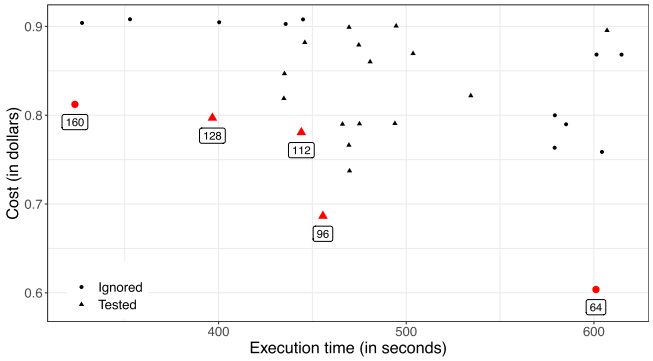



**FIGURE 8** List of candidate configurations for the CyberShake workflow

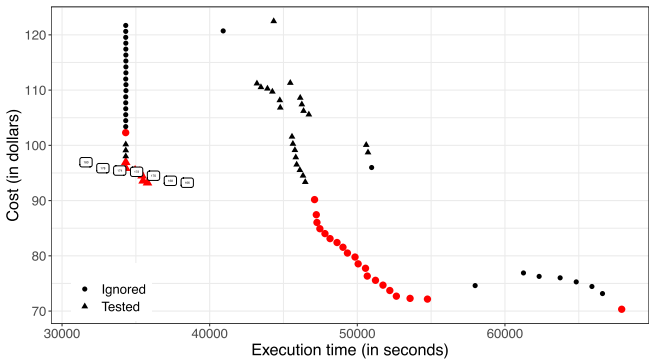



**FIGURE 9** List of candidate configurations for the Epigenomics workflow

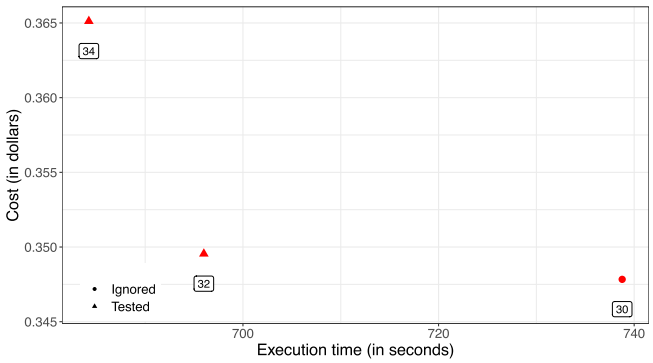



**FIGURE 10** List of candidate configurations for the Montage workflow

degree of parallelism ($C_{max}$), the execution time achieved when using this particular number of cores ($E_{c_{max}}$), and the minimum cost a user would have to pay to execute the workflow on a single 2-core instance ($C_2$).

The CyberShake and Montage applications can complete in about 5-6 min for a budget under one dollar. However, these two applications requires multiple runs on different data sets to produce scientific results. Then, despite these short execution time and low cost, optimizing one or both metrics is still interesting. Epigenomics has a much longer execution time of several hours which implies a greater budget for a single execution, but also more room for optimization.

Figures 8–10 show all the configurations that meet the condition of not degrading the best execution time and the best cost by more than a factor of two respectively for the CyberShake, Epigenomics, and Montage applications. These figures distinguish the configurations that are actually simulated during the binary and exhaustive searches (depicted by triangles) from those that were ignored by our algorithm (depicted by circles). The larger (red) triangles and circles identify configurations on the Pareto front, while the smaller (black) triangles and circles are the dominated solutions.

**TABLE 4** Number of simulations and time needed to return a set of VM instance configurations to the user

| Workflow | # runs | Init | Binary | Exhaustive | Total |
|---|---|---|---|---|---|
| CyberShake | 24 | 18.4 s | 74.3 *s* | 121 *s* | 213.7 s |
| Epigenomics | 35 | 35.1 s | 8.5 *s* | 142 *s* | 185.6 s |
| Montage | 12 | 106 s | 143.2 *s* | 0 *s* | 249.2 s |

Finally, the labels indicate the respective number of cores of each configuration on the Pareto front. For the sake of readability, we only display the labels for the simulated dominating solutions for the Epigenomics workflow.

For the CyberShake application (Figure 8), the binary search starts at $C_{max} = 374$ and stops at 128 after eight iterations. The first encountered configuration that meets the required conditions has 94 cores. Then, we exhaustively simulate all the configurations between 96 and 126 cores. Among these configurations, only three are on the Pareto front with respectively 96, 112, and 128 cores. We also see that our algorithm ignores three dominating configurations with 64, 144, and 160 cores. The one with 64 cores is clearly not a good candidate, as it leads to a much larger execution time. The two other ignored configurations are better contenders and could have been returned to the user as well. With 160 cores, the execution time is reduced by 73 s for a cost increase of \$0.019 with regard to the best configuration returned by our algorithm with 128 cores.

For the Epigenomics application (Figure 9), $c_{max} = 246$. The binary search only needs two iterations to find two consecutive candidate configurations with 124 and 186 cores respectively. Then, 30 configurations have to be simulated in the exhaustive search for candidates. Among these configurations, seven are on the Pareto front with respectively 166, 168, 170, 172, 174, 178, and 180 cores. Our algorithm ignores about 21 dominating configurations, all but one having less than 124 cores. However, they cannot be considered as interesting for the user. Indeed, there is an important gain on the execution time when going from 124 to 170 cores ($\approx$ 4.25 hours) for a cost increase of only \$3. The last ignored configuration with 190 cores increases the cost by almost \$6 but reduces the execution time by less than 5 s with regard to the configuration with 180 cores that our algorithm selects.

For the Montage application (Figure 10), the selection is entirely done by the binary search. Starting from $c_{max} = 662$, it ends by finding two consecutive candidate configurations with 32 and 34 cores. Then, there is no need for the exhaustive search part. We can see in Figure 10 that only three configurations meet our initial requirements. Among them, our algorithm clearly favors the reduction of the execution time. The selected configuration with 34 cores reduces the execution time by 55 s for an extra cost of \$0.02 compared to using 30 cores.

Table 4 summarizes the number of simulations and the time needed to return a set of configurations to the user. We decompose this time in three parts: (i) *Init* corresponds to the determination of $c_{max}$, $E_{c_{max}}$, and $C_2$; (ii) *Binary* is the time spent in the binary search; and (iii) *Exhaustive* sums up the duration of the runs in the exhaustive search.

The decomposition of this time to solution differs from one workflow to another but we are able to determine a suitable set of configurations for all workflows in less than 5 min. The binary search for the Montage application ends by finding two candidates that have respectively 32 and 34 cores. There is thus no need for an exhaustive search. The binary search for Epigenomics stops after two iterations but let a large search space to cover (from 124 to 186 cores) making the exhaustive search the most time-consuming part of our algorithm. However, it is composed of independent simulations that can be launched in parallel to reduce the time to solution.

We conclude this evaluation by measuring, for each of the configurations selected by the algorithm presented in Section 4, the gain in terms of execution time and volume of data transferred over the network with regard to configurations made of the same set of virtual machine instances but where all the files are stored on the EBS service. The results of this comparison are given in Table 5.

The proposed data-aware planning allows for a significant reduction of the volume of data transferred over the network by leveraging the fast storage offered by M5d instances for the CyberShake and Montage applications. Data movement is divided by almost a factor two for CyberShake and four for Montage. The gain is, however, very limited for Epigenomics as most of the data involved corresponds to the input files of the workflow. Part of this reduction of data transfers has an impact on execution time for CyberShake which produces more intermediate files while Montage's execution time is dominated by computation.

## 6 | RELATED WORK

While some studies on workflow scheduling algorithms aim at minimizing dynamic energy consumption,[21] scientific workflow planning or scheduling algorithms that target IaaS clouds usually aim at finding the best compromise between the execution time of the workflow and the number and type of virtual machine instances used for execution. Following the pay-as-you-go model of the IaaS clouds, this amount of resources usually corresponds to a certain *cost*. A typical approach is to fix one of the objective and thus consider it as a constraint, that is, a delay[22-24] or a budget,[25,26] and to optimize the other objective. Some articles directly solve this bi-objective optimization problem by selecting a solution among those composing

**TABLE 5** Reduction of volume of transferred data and execution time w.r.t. using the shared EBS service

| | | Reduction | |
|---|---|---|---|
| **Workflow** | **# cores** | **Data transfers** | **Execution time** |
| CyberShake | 96 | 62.35% | 10.79% |
| | 112 | 53.31% | 7.45% |
| | 128 | 45.99% | 8.68% |
| Epigenomics | 166 | 0.91% | 0% |
| | 168 | 0.91% | 0% |
| | 170 | 0.91% | 0% |
| | 172 | 0.91% | 0% |
| | 174 | 0.91% | 0% |
| | 178 | 0.91% | 0% |
| | 180 | 0.91% | 0% |
| Montage | 32 | 96.22% | 0.60% |
| | 34 | 89.02% | 0.04% |

the Pareto front.[19,27,28] These heuristics usually consist in variations of classical list scheduling and aim at finding a single solution with good properties. The principles behind the multi-objective heterogeneous earliest finish time (MOHEFT) algorithm proposed in Reference 19 are very close to those of the present work. MOHEFT is an extension of the seminal HEFT list heuristic[13] that solves a bi-criteria optimization problem. MOHEFT[19] builds several intermediate workflow schedules in parallel in each step instead of a single schedule and uses dominance relationships and crowding distance[20] to ensure the diversity of tradeoff solutions. In their article, the authors of MOHEFT obtain better results compared to competing algorithms from the literature. However, they consider that each workflow that can use all the cores of the VM on which it is scheduled, while we keep the initial sequential nature of the tasks composing the studied workflow. Moreover, we rely in this work on realistic simulations to assess the performance of multiple solutions and provide the user with a limited set of candidates on the Pareto front.

Clustering several computational tasks together to execute them within a single virtual machine is another classical approach in workflow planning. However, it is usually done to reduce the overhead associated to the execution of fine grain tasks with short execution times[29] or improve fault tolerance.[30] In this work, task clustering aims at improving data locality and reducing the amount of data transfers.

The optimization of data transfers during the scheduling of a data-intensive workflow has also been studied. The authors in Reference 31 propose an evolutionary algorithm minimizing the data transfer among tasks and the total execution time of the workflow. To this end, output files are duplicated on a shared storage space and on the machine where it was produced. Tasks assigned on a machine where some of their input files are located will thus spare a data transfer. In our work, we favor transfers between VM instances rather than from the share storage space. In Reference 32, workflows are modeled as hypergraphs. The authors propose a partitioning algorithms of these hypergraphs that minimizes the number of file transfers. Each partition is then assigned to a VM instance. However, the authors do not consider any environment characteristics such as processing and storage capacity or and transfer rates. Only the total size of all transferred files is used to evaluate the quality of a solution.

Leveraging the performance of fast SSD storage to handle the large amount of intermediate data produced by data-intensive scientific workflows has been investigated for HPC infrastructures. Burst-buffers[33] allow to reduce the stress on the parallel file system of a supercomputer by positioning a nonvolatile storage between the processor's memory and the file system. Taking advantage of this extra storage layer improves I/O performance.[34] A characterization of the I/O pattern that a workflow should exhibit to benefit of burst buffers is given in Reference 35. In this article, we propose to see fast storage as an extra storage layer between cores in a virtual machine and the shared block storage of an IaaS cloud.

## 7 | CONCLUSION AND FUTURE WORK

Infrastructure as a service clouds now allows scientists to execute their data intensive workflows on tailored infrastructures that match the computing and storage requirements of these applications. Determining the set of virtual machine instances that have to compose these infrastructures is a complex tasks, usually delegated to workflow management systems. A key to performance is to be able to leverage the characteristics of virtual machines instances.

In this article, we proposed an original data-aware planning algorithm that map tasks in a way that increases local storage on fast SSD drives. We also proposed a simulation-driven approach to select a set of virtual machines instances by solving a cost-performance optimization problem. We assessed the performance of the proposed algorithms on three popular scientific workflows with different characteristics. The presented experimental results showed that our algorithm is capable to return a narrow set of configurations to a user in a reasonable time. We also shown that for these configurations we were able to significantly reduce the volume of data transferred over the network, which, for one of the considered workflows, translates into a reduction of the execution time by 7% to 10% when compared to relying only on the shared Elastic Block Storage service to store intermediate data.

As part of our future work, we plan to further improve the realism and speed of the simulator underlying this work by leveraging the new features provided by the latest stable release of WRENCH. Moreover, we would like to compare the simulated executions with actual runs on the AWS computing cloud with M5d instances in order to confirm the impact of the proposed algorithms. An interesting extension to this work would be to make the stopping conditions of the binary search configurable parameters so that users can favor either a shorter execution time or a lowest cost. We also plan to investigate a complementary approach where one of the objective is fixed, that is, either a given budget or a fixed deadline.

## ACKNOWLEDGMENTS

The authors would like to thank Rafael Ferreira da Silva, Henri Casanova, and all the WRENCH development team for their valuable help in the design of the proposed WRENCH-based simulator.

## DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

## ORCID

*Tchimou N'Takpé* https://orcid.org/0000-0002-8291-1347

## REFERENCES

1. Taylor I, Deelman E, Gannon D, Shields M. *Workflows for e-Science: Scientific Workflows for Grids*. Springer Publishing Company; 2014.
2. Albrecht M, Donnelly P, Bui P, Thain D. Makeflow: a portable abstraction for data intensive computing on clusters, clouds, and grids. Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies (SWEET@SIGMOD); 2012; Scottsdale, AZ.
3. Deelman E, Vahi K, Juve G, et al. Pegasus, a workflow management system for science automation. *Future Gener Comput Syst*. 2015;46:17-35.
4. Fahringer T, Prodan R, Duan R, et al. ASKALON: a development and grid computing environment for scientific workflows. Workflows for e-Science; 2007.
5. Amazon elastic compute cloud (EC2). Accessed January 2020. https://aws.amazon.com/ec2/
6. Google compute engine. Accessed July 2019. https://cloud.google.com/compute
7. Microsoft Azure. Accessed July 2019. https://azure.microsoft.com/
8. Ferreira da Silva R, Casanova H, Tanaka R, et al. Developing accurate and scalable simulators of production workflow management systems with WRENCH. *Future Gener Comput Syst*. 2020;112:162-175. doi:10.1016/j.future.2020.05.030
9. AWS M5 instances. Accessed January 2020. https://aws.amazon.com/ec2/instance-types/m5/
10. Mathá R, Ristov S, Fahringer T, Prodan R. Simplified workflow simulation on clouds based on computation and communication noisiness. *IEEE Trans Parallel Distrib Syst*. 2020;31(7):1559-1574. doi:10.1109/TPDS.2020.2967662
11. Juve G, Chervenak A, Deelman E, Bharathi S, Mehta G, Vahi K. Characterizing and profiling scientific workflows. *Future Gener Comput Syst*. 2013;29(3):682-692.
12. Gerasoulis A, Yang T. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. *J Parallel Distrib Comput*. 1992;16(4):276-291. doi:10.1016/0743-20107315(92)90012-2010C
13. Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans Parallel Distrib Syst*. 2002;13(3):260-274. doi:10.1109/71.993206
14. Mu'alem A, Feitelson D. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans Parallel Distrib Syst*. 2001;12(6):529-543.
15. The WRENCH Project; 2019. https://wrench-project.org
16. Casanova H, Giersch A, Legrand A, Quinson M, Suter F. Versatile, scalable, and accurate simulation of distributed applications and platforms. *J Parallel Distrib Comput*. 2014;74(10):2899-2917.
17. Pegasus workflow gallery; Accessed July 2019. http://pegasus.isi.edu
18. Bharathi S, Chervenak A, Deelman E, Mehta G, Su MH, Vahi K. Characterization of scientific workflows. Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS); 2008:Austin, TX.
19. Durillo JJ, Prodan R. Multi-objective workflow scheduling in Amazon EC2. *Cluster Comput*. 2014;17(2):169-189.
20. Deb K, Agrawal S, Pratap A, Meyarivan T. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. Proceedings of the International Conference on Parallel Problem Solving from Nature; 2000; Paris, France.
21. Shu T, Wu CQ. Energy-efficient mapping of large-scale workflows under deadline constraints in big data computing systems. *Future Gener Comput Syst*. 2020;110:515-530. doi:10.1016/j.future.2017.07.050
22. Abrishami S, Naghibzadeh M, Epema DH. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Gener Comput Syst*. 2013;29(1):158-169.
23. Arabnejad V, Bubendorfer K, Ng B, Chard K. A deadline constrained critical path heuristic for cost-effectively scheduling workflows. Proceedings of the 8th IEEE/ACM Intl. Conf. on Utility and Cloud Computing; 2015; Limassol, Cyprus.

24. Wang ZJ, Zhan ZH, Yu WJ, et al. Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling. *IEEE Trans Cybern*. 2020;50(6):2715-2729. doi:10.1109/TCYB.2019.2933499
25. Rezaeian A, Abrishami H, Abrishami S, Naghibzadeh M. A budget constrained scheduling algorithm for hybrid cloud computing systems under data privacy. Proceedings of the 2016 IEEE International Conference on Cloud Engineering (IC2E); 2016; Berlin, Germany.
26. Shu T, Wu CQ. Performance optimization of hadoop workflows in public clouds through adaptive task partitioning. Proceedings of the IEEE Conference on Computer Communications (INFOCOM); 2017; Atlanta, GA.
27. Zhou X, Zhang G, Sun J, Zhou J, Wei T, Hu S. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. *Future Gener Comput Syst*. 2019;93:278-289.
28. Chen ZG, Zhan ZH, Lin Y, et al. Multiobjective cloud workflow scheduling: a multiple populations ant colony system approach. *IEEE Trans Cybern*. 2019;49(8):2912-2926. doi:10.1109/TCYB.2018.2832640
29. Chen W, Ferreira da Silva R, Deelman E, Sakellariou R. Using imbalance metrics to optimize task clustering in scientific workflow executions. *Future Gener Comput Syst* 2015; 46: 69–84. Funding Acknowledgements: NSF IIS-0905032 and NSF FutureGrid 0910812 doi: 10.1016/j.future.2014.09.014
30. Chen W, Ferreira da Silva R, Deelman E, Fahringer T. Dynamic and fault-tolerant clustering for scientific workflows. *IEEE Trans Cloud Comput*. 2016;4(1):49-62. Funding Acknowledgements: NSF IIS-0905032, NSF ACI SI2-SSI 1148515, and NSF FutureGrid 0910812. doi:10.1109/TCC.2015.2427200
31. Szabo C, Sheng QZ, Kroeger T, Zhang Y, Yu J. Science in the cloud: allocation and execution of data-intensive scientific workflows. *J Grid Comput*. 2014;12(2):245-264.
32. Çatalyürek Ü, Kaya K, Uçar B. Integrated data placement and task assignment for scientific workflows in clouds. Proceedings of the 4th International Workshop on Data-Intensive Distributed Computing; 2011; San Jose, CA.
33. Wang T, Mohror K, Moody A, Sato K, Yu W. An ephemeral burst-buffer file system for scientific applications. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, (SC); 2016; Salt Lake City, UT.
34. Ferreira da Silva R, Callaghan S, Deelman E. On the use of burst buffers for accelerating data-intensive scientific workflows. Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science (WORKS); 2017; Denver, Co.
35. Daley C, Ghoshal D, Lockwood G, Dosanjh S, Ramakrishnan L, Wright N. Performance characterization of scientific workflows for the optimal use of burst buffers. Proceedings of the 11th Workshop on Workflows in Support of Large-Scale Science (WORKS); 2016; Salt Lake City, UT.

**How to cite this article:** N'Takpé T, Edgard Gnimassoun J, Oumtanaga S, Suter F. Data-aware and simulation-driven planning of scientific workflows on IaaS clouds. *Concurrency Computat Pract Exper*. 2022;34(14):e6719. doi: 10.1002/cpe.6719