# A Workflow Scheduling Algorithm for Reducing Data Transfers in Cloud IaaS

Jean Edgard GNIMASSOUN[1], Souleymane OUMTANAGA[4]
Laboratoire de Recherche en Informatique et Télécommunication
Institut National Polytechnique-HB
Yamoussoukro, Côte d'Ivoire

Tchimou N'TAKPE[2]
Laboratoire de Mathématiques et Informatique
Université Nangui Abrogoua
Abidjan, Côte d'Ivoire

Gokou Hervé Fabrice DIEDIE[3]
Laboratoire de Recherche en Mathématiques et Informatique
Université Peleforo Gon Coulibaly
Korhogo, Côte d'Ivoire

***Abstract*—The cloud IaaS easily offers to have homogeneous multi-core machines (whether they are "bare metal" machines or virtual machines). On each of these machines, there can be high-performance input-output SSD disks. That allows to distribute the files produced during the execution of the workflow to different machines in order to minimize the additional costs associated with transferring these files. In this paper, we propose a scheduling algorithm called WSRDT (Workflow Scheduling Reducing Data Transfers) whose purpose is to minimize the makespan (execution time) of data-intensive workflows by reducing transfers data between dependent tasks on the network. Intermediate files produced by tasks are stored locally on the disk of the machine where the tasks were executed. We experimentally verify that the increase in the number of cores per machine reduces the additional cost due to data transfers on the network. Experiences with a veritable workflow show those advantages of the algorithms presented. Data-driven scheduling significantly reduces the execution time and the volume of data transferred on the network, our approach outperforms one of the best state-of-the-art algorithms that we have adapted with our hypotheses.**

***Keywords*—*Workflow scheduling; makespan reduction; multi-cores virtual machine; data-intensive workflows; IaaS cloud***

## I. Introduction

Scientists, to run their different parallel applications, generally used clusters and grids computing. These different execution platforms quickly have showed their limits giving the ever-increasing demands for computing, storage resources, and so on. To solve this issue, cloud computing offers an illusion of infinite resources where scientists can request the resources needed to run a parallel application. Cloud computing typically offers three (03) types of services, SaaS (Software as a Service), PaaS (Platform as a Service), IaaS (Infrastructure as a Service). The use of these different services is flexible and scalable from the request of the user, via a pay-as-you-go model. With three (03) basic services, the most suitable for running parallel applications is the IaaS cloud. The providers of this service offer computing and storage resources essential for running all parallel applications that require a significant resource due to its complex structure.

Parallel applications come from several research fields such as biology, astronomy, physics, agriculture, etc., and have in common, on the one hand, their complex structure with dependencies between the different tasks, and on the other hand a need for high computing and storage service, given the large volume of data to be processed and transferred. These scientific applications are very often modeled as scientific workflow. These scientific workflows require a High-Performance Computing (HPC) environment for their execution.

The evolution of the computing environment from grid to cloud computing has always considered scientific workflows. However, with this new paradigm, scientific workflows are now executed on virtual, dynamic and scalable resources as an instance in cloud computing. The challenge of mapping workflow tasks, which is a task scheduling problem in a cloud computing environment, is the subject of several scientific studies to find algorithms to execute workflows in a reasonable time and budget. This problem of scheduling on IaaS infrastructures of cloud computing is known as NP-hard [1], for this purpose, several heuristics [2][3][4] and metaheuristics [5][6] have been proposed in the literature in order to minimize either the total execution time of the workflow (makespan), the cost of using IaaS resources in the cloud, or both.

Solutions for workflow scheduling on cloud IaaS infrastructures exist, but these algorithms generally consider the execution of a task on a VM with a single computational core. And depending on the complex structure of the workflows, this could lead to several data transfers (communications) in networks thus constituted. According to the Amazon EC2's VM deployment model, users will be able to order VMs with a maximum of ninety-six (96) parallel computing cores in the same VM[1]. However, minimizing the makespan of a workflow, one must consider in addition to the execution time of the task on the computing resource, the time of transfer from a task to its successor(s), because large volumes of data are must be transferred. Using multi-core VMs for scheduling could give better results for the makespan, as it could reduce the amount of data exchanged in the network. Indeed, if two dependent tasks running on the same VM, the communication time (between these two tasks) is assumed to be zero.

Most of the algorithms in the literature are not clairvoyant, i.e., they do not consider the location of data coming from

[1]https://aws.amazon.com/fr/ec2/instance-types/m5/

predecessors and data going to successors of a task. In order to improve the execution time of a workflow, the execution time of each task in the workflow and the data transfer time between dependent tasks must be considered. The works in the literature consider only the data coming from the predecessor tasks, i.e. from top to bottom, since the scientific application is modeled as a DAG. However, an improvement can be done on the location of the data. Moreover, the algorithms in the literature do not exploit multi-core machines for the simultaneous execution of several tasks in the same machine and the distributed storage of the data produced during the execution of the workflow.

Reducing the execution time of a scientific application means considering the execution time of each task of the application, but also the file transfer time between the different dependent tasks through the network. The main problem addressed in this paper is how to do a good mapping of the different tasks from a data-intensive application by reducing the files to be transferred in the network in order to obtain a better execution time.

The remainder of this paper is organized as follows. Section II introduces the related work in this field and section III present the platform and application models. Sections IV and V describes the proposed approach: WSRDT and section V validates the effectiveness WSRDT. Concluding remarks are given in section VI.

## II. Related Work

Two main approaches exist for scheduling tasks in the cloud, which are list scheduling algorithms [7][8][9] and clustering algorithms [10][11][12]. Most of the list scheduling algorithms are inspired by HEFT [13], which aims to minimize the makespan and was originally proposed for computational grid environment and has long been studied and adapted for the cloud environment.

In this section present a review of the literature on algorithms whose objective is to minimize makespan. Running a scientific workflow application in the cloud requires efficient mapping so that tasks do not have to wait too long, which could result in a very long execution time. Typically, the resources provided in the cloud to run workflows are VMs with computational units, storage, etc. Reducing the execution time of a workflow consisting of hundreds or even thousands of tasks in the cloud is a challenge, given the flexibility of available resources. The key part of resource management in a cloud environment is the mapping of these tasks to these on-demand computing resources. Rimal et al. [14] propose a model based on the public cloud (Amazon EC2), in order to minimize the makespan, the proposed algorithm is based on the principle of critical path (Critical Path: CP). Critical path tasks are assigned to different resources in the cloud, and to maximize the use of these leased resources, other tasks that are not part of the critical path are assigned to those resources already leased, taking into account the billing that is done per unit of time; knowing that a VM used during 01H01mn would be charged for 02H. This approach based on the "multi-tenant cloud" consists in deploying the tasks of the same CP at the cloud provider whose resources allow to finish these different tasks at the earliest possible time in order to reduce the completion time of the workflow. The study of Rimal et al. showed that their approach gives better results compared to the FCFS algorithm, which is not a clear-sighted algorithm because it does not take into account all the dependencies that would exist between the different tasks of the workflow. In addition to the critical path approach, Gamal et al. [10] propose a new approach based on task classification. Their task clustering approach, where groupings are done according to a certain neighborhood, minimizes the makespan based on the Min-Min [9] algorithm for mapping tasks to cloud resources. Min-Min algorithm can be used in cloud computing. Min-Min algorithm depends on execution time for scheduling tasks. Tasks with minimum execution time will be scheduled first. Tasks with long execution time have high delay. The Min-Min algorithm is not suitable for running a parallel application where the tasks are dependent just like the FCFS algorithm. Almi'ani and Lee [15] proposed a three-step approach to minimizing the makespan: (i) the partitioning step; in this step, the number of tasks assigned to each partition is first determined taking into consideration the execution time for the CP in the workflow. Since the sum of task execution times along CP (i.e., critical path length) represents the lower bound of makespan (i.e., the optimal solution), this step tries to ensure the total execution time for tasks belonging to the same partition to be less than CP length. While critical path length only includes execution times as tasks along CP are meant to assigned to the same resource, the length of tasks in any other partition should include execution times and communication times. To ensure that partitions created in the partitioning step are at the optimal granularity for the final resource allocation, tasks of different partitions are (ii) rearranged/adjusted. The optimality here primarily concerns the number of tasks in each partition with respect to the capacity of potentially assigned resource and data locality. As partitions are expected to have dependency relationships due to task precedence constraints primarily dictated by data dependencies, rearranging tasks between different partitions involves the recalculation of timing values. To execute each task the (iii) resource assignment step consists of the resource set identification to identify types of resource set allocated to partitions such that the amount of time partitions are required to wait due to the presence of the data dependencies between partitions is minimized. To assess their approach Almi'ani et al. compared their approach to HEFT, but HEFT provided better makespan compared to their approach which gives better cost of using cloud resources. The most suitable algorithms for scheduling a parallel application where tasks are dependent are list algorithms because this type of algorithm takes into account the dependencies between all the tasks in the workflow.

The Heterogeneous Earliest Finish Time Algorithm (HEFT) is a popular list-based heuristic scheduling algorithm for optimizing the makespan [13] in workflow applications, whose pseudo-code is very close to algorithm 1. The method consists of two phases: ranking and mapping. In the ranking phase (line 1) based on the (1), the order in which the tasks are being mapped is computed using the bottom-level metric (distance of the beginning task to the end task of the workflow). The idea of this ranking is to execute before those tasks having more dependent tasks than others. Further details about how to sort the tasks can be found in [13]. Once the

execution order is determined, the second phase consists in assigning each task to the resources following the order computed in the first phase. For each task and for each resource, the completion time of that task on that resource is computed. Finally, the task is mapped onto the resource where it is finished earlier. After all tasks have been mapped, the workflow can be executed.

$$bl_i = \omega_i + \max_{j \in succ(i)} (c_{i,j} + bl_j) \quad (1)$$

Where *succ (i)* is the set of immediate successors of task $v_i$, $c_{i,j}$ is the data transfers time from task $v_i$ to task $v_j$, and $\omega_i$ is the execution time of task $v_i$. Since the *bottom-level* is computed recursively by traversing the DAG upward, starting from the end task. For the end task $v_{end}$, the *bottom-level* value is equal to.

$$bl_{end} = \omega_{end} \quad (2)$$

HEFT is a very popular list scheduling algorithm that aims at minimizing the makespan when resources are fixed, but HEFT does not perform well in minimizing the makespan of a workflow when the volumes of data exchanged between tasks are large. Indeed HEFT, after having sorted the tasks according to their priorities, tries to minimize the end date of execution of each task in the order of this list. It is therefore a ''blind'' algorithm through which the decision taken for a task is final and can have a negative impact on lower priority tasks. In addition, these algorithms use a naive adaptation of HEFT for Cloud IaaS platforms using a single centralized storage service for data exchanges between tasks. This significantly increases the additional cost of inter-task data exchange.

In all these studies, the VMs considered are heterogeneous and are in fact distinguished by their differences in terms of the number of cores. The authors therefore assume that the workflow tasks are parallel tasks that can run on any number of cores. However, in reality, the workflows on which they make their assessments are inspired by real workflows studied by Juve et al. [16]. In their study, Juve et al. show that almost all tasks in real workflows are single-core. There is only one task in one of the studied workflows that can use up to two cores. In this study, the workflows considered are therefore comprised of single-core tasks only.

All the studies in the literature do not take into account that one can take advantage of the rental of multi-core machines containing local storage disks in order to reduce the makespan by reducing data transfers. However, storing all the files used and produced by a workflow on a single central storage service can cause contention on the network. This study is based on the use of local VM disks to propose an algorithm to minimize makespan. The algorithm will also take advantage of the fact that the same multicore VM can be used to execute several tasks in parallel.

## III. Platform and Application Models

In this paper, the platform model is based on a typical IaaS cloud configuration. Multiple virtual machine (VM) instances are deployed on physical servers within a single datacenter. More precisely, a set of VMs like Amazon EC2 M5 instances is considered. Specifically, these are the M5d instances that are provided with local storage on the NVMe SSD, while regular M5 instances must rely on Amazon Elastic Block Storage (EBS) to store the data. Table I 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 4GiB. These instances are typically deployed by Amazon on nodes featuring an 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. This work, aim at leveraging this fast storage that is shared by the vCPUs of an instance to store the intermediate files produced during the execution of a workflow, hence reducing the number of data transfer 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.

In terms of network connectivity with other instances or the Elastic Block Storage (EBS) service, the available bandwidth depends on the size of the instance. Only the largest instances that can exploit a full node, i.e., with 64 or 96 vCPUs, have a guaranteed network bandwidth of 20 and 25 Gbps respectively. For smaller instances, i.e., from 2 to 16 cores, the bandwidth is proportional to the vCPUs.

TABLE I. Characteristics of the AWS M5d Instance Types

| Model | vCPU | Memory (GiB) | Instances Storage (GiB) | Network Bandwidth (Gbps) | EBS Bandwidth (Mbps) | Cost ($/H) |
|---|---|---|---|---|---|---|
| M5d.large | 2 | 8 | 1 x 75 NVMe SSD | Up to 10 | Up to 3,500 | 0,113 |
| M5d.xlarge | 4 | 16 | 1 x 150 NVMe SSD | Up to 10 | Up to 3,500 | 0,226 |
| M5d.2xlarge | 8 | 32 | 1 x 300 NVMe SSD | Up to 10 | Up to 3,500 | 0,452 |
| M5d.4xlarge | 16 | 64 | 2 x 300 NVMe SSD | Up to 10 | 3,500 | 0,904 |
| M5d.8xlarge | 32 | 128 | 2 x 600 NVMe SSD | 10 | 5,000 | 1,808 |
| M5d.12xlarge | 48 | 192 | 2 x 900 NVMe SSD | 10 | 7,000 | 2,712 |
| M5d.16xlarge | 64 | 256 | 4 x 600 NVMe SSD | 20 | 10,000 | 3,616 |
| M5d.24xlarge | 96 | 384 | 4 x 900 NVMe SSD | 25 | 14,000 | 5,424 |

In this study, large VMs are preferred in each platform, because they allow multiple tasks to be executed in parallel. In their study Juve et al. [16] have shown that each task in a real scientific workflow is a single-core activity, i.e. can only be executed on a single computing core, rather than using all the cores of a VM. It is on this same principle that this study is based. Thus, a user who wants to run his parallel application in the cloud must rent a number of cores in total, the proposed approach provides him with a platform that would minimize the application completion time. For example, if the user wants to use 100 cores in total, the platform will consist of a VM with 96 cores and a VM with 4 cores. In the case of 200 cores total, the platform will consist of three VMs, two VMs of 96 cores and one of 8 cores and so on.

The scientific workflows (cf. Fig. 1) to schedule are represented by Directed Acyclic Graphs (DAGs) G = {$V$, $\mathcal{E}$}where $V$= $\{v_i \mid i = 1, \dots, V\}$ is a set of vertices representing the computational tasks of the workflow and $\mathcal{E} = \{e_{i,j} \mid (i,j) \in \{1, \dots, V\} \times \{1, \dots, V\}\}$ is a set of edges between vertices, representing either a data dependency, i.e., a file transfer, or a flow dependency between two tasks. 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.

Notations such as $Input_i^k$ (resp. $Output_i^k$), represent 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, two quantities associated with each task of the workflow that will be used during the planning process have been defined. The *Local Input Volume* of task $v_i$ on machine $M_j$, or $LIV_{i,j}$, as the sum of the size of the files that $v_i$ takes as input that are locally stored on $M_j$. Respectively, 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. 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 EBS goes through a dedicated network connection, which is taken into account in the simulated infrastructure. As for the network connections between VMs, One of the assumptions made in this study is that the bandwidth of the dedicated connection between VM and EBS is proportional to the number of cores for small VMs with up to 16 cores (i.e., 218.75 Mbps per core).

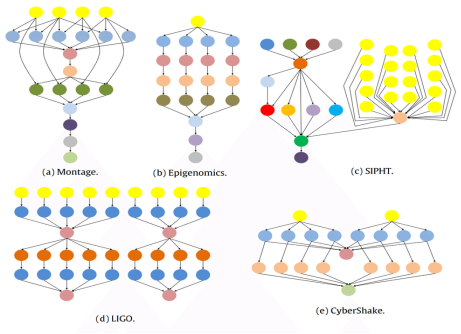


Fig. 1. Some Examples of Scientific Workflows.

During the execution of the workflow, all the intermediary files, i.e. 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 an external storage 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.

## IV. A Planning Algorithm to Minimize Data Transfer over the Network

The proposed planning algorithm aims at leveraging two main characteristics of the target IaaS cloud platform, i.e. multi-core instances and a fast-local storage space shared among cores, to minimize the impact of data transfers on the execution of data-intensive scientific workflows.

In this section, the assumption is that the provisioning of virtual machine instances has been done. Then, the objective of this algorithm is to schedule the set $V$ of $V$ tasks composing the workflow on a set $M$ of $n$ VMs instances. These instances can have different sizes. Each of them has a unique id, the largest instances having the smallest ids. How the set of instances is defined will be explained in Section V.

Algorithm 1 starts by building a sorted scheduling list that contains all the tasks of the workflow (lines 1-2). The tasks are sorted by decreasing *bottom level* value [13].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. 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 V (line3-7). The selected machine $M_j$ in $M$ is the one that first minimizes the start time of $v_i$ (denotes as $st_j(v_i)$) and then maximizes the volume of the input files needed by $v_i$ for its execution that are already locally stored on $M_j$. The rationale is that between two virtual machines able to start $v_i$ start its execution at the same time, the algorithm favor the one that minimizes the amount of data transfer over the network.

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, each machine is managed as a job and resource manager will

do. In particular, this study leverage the available information on the (estimated) duration of each task to implement a conservative backfilling mechanism [17] when building the local schedule. Keeping such *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 $M$ of the execution of $v_i$, it is essential to update the *usage profile* of $M$ (line 6). These usage profiles of the virtual machines are also used in second step of Algorithm 1 in which the tasks in this initial schedule are rearranged to further reduce the amount of data transfers over the network.

This rearrangement step (lines 8 to 11) browses the workflow DAG *level by level* from the bottom to the top. The motivation of this second step 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 considered. It is indeed impossible to account for the locality of the data needed by the direct descendants of a task when scheduling it at their placement is not determined yet. This may lead to avoidable data movements.

Level 0 is the topmost level of the DAG that comprises all the entry tasks of the workflow. For each of the other tasks, the level is recursively computed as the maximum level of its predecessors plus one. Finally, $L$ denote the number of levels in the workflow.

**Algorithm 1** Mapping workflow tasks without rearrangement

```
1   Compute bl_i of each task v_i
2   Sort V by decreasing bl_i values
3   for all v_i ∈ V do
4    M ← {M_j ∈ M | 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 ← tasks in level l sorted by decreasing bl values
10   Rearrange (V_l) ► see Algorithm 2
11  end for
```

The principle of the rearrangement step is described in Algorithm 2. It start by saving the current start time and mapping (denoted as $st^c(v_i)$ and $M^i$) for each task $v_i$ in $V_l$ (lines 2 and 3). Then, the local volume $LV_{i,j}$ for task $v_i$ on machine $M_j$ (lines 4 to 7) is determined. Also, the local volume for the current mapping of $v_i$ (line7) is saved before cancelling this mapping (line 8). This last action 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, i.e. $LV_{i,k} \geq LV_i^c$, and reduce the starting time of the task, i.e. $st_k(v_i) \leq st^c(v_i)$. Where $st_k(v_i)$ is the new start time of $v_i$ on $M_k$.

The main loop in Algorithm 2 (lines 11 to 32) aims at iteratively improving the mappings for tasks in $V_l$. At each step, the algorithm first try to find a better mapping (lines 15 to 21) for each task by considering the machine that leads to the greatest increase 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.

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

```
1   for all v_i ∈ V_l do
2    st^c(v_i) ← current start time of v_i
3    M^i ← current mapping of v_i
4    for all M_j ∈ M do
5    LV_{i,j} ← LIV_{i,j} + LOV_{i,j}
6    end for
7    LV_i^c ← current local volume of v_i
8    cancel the current mapping of v_i
9   end for
10  level_is_rearranged ← FALSE
11  while ¬ level_is_rearranged do
12   level_is_rearranged ← TRUE
13   for all v_i ∈ V_l do
14   Sort M by decreasing LV_{i,j} value
15   while LV_{i,j} ≥ LV_i^c do
16   if st_j(v_i) ≤ st^c(v_i) then
17   map v_i on M_j
18   update the usage profile of M_j
19   break
20   end if
21   end while
22   if v_i is mapped on M^i or
     st_{M^i}(v_i) > st^c(v_i) then ► no better mapping
23   V_l ← V_l \ {v_i} ► mapping is definitive
24   level_is_rearranged ← FALSE
25   end if
26   end for
27   if ¬level_is_rearranged then
28   for all v_i ∈ V_l do
29   cancel the current mapping of v_i
30   end for
31   end if
32  end while
```

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 to 26). However, this decision may invalidate some of the migrations (e.g., the task with higher priority mapped on $M_i$ ). Then, all the tentative mappings determined in this step (lines 28 to 30) are cancelled and another rearrangement of the remaining tasks is searched. 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.

## V. Results and Discussion

To evaluate this approach with HEFT, a simulator based on the WRENCH project[2][18] was wrote, a Cyber-Infrastructure 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 [19][20] on top of which services to simulate the execution of

[2] https://wrench-project.org

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 behaviours when compared to actual systems.

### A. Determinig the Data Transferred through Simulation

The 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 multi-core virtual machines that share a fast storage space given as input.

In Table II, VS is the size of VM used; mksp is the makespan; TVF is Total volume of files and VFT is the volume of file transferred. This table provides a detailed description of the set of files for the five literature's workflows and the volumes of file to be transferred if either Algorithm 1 (without rearrangement) or Algorithm 2 (with rearrangement) was applied. These transferred files are obtained after simulation on platforms where each VM has 16 or 96 cores. With platforms with 96 cores per VM, there's a slight difference in the volume of files transferred compared to platforms with 16 cores. On the other hand, compared to the total volume of each workflow, the rearrangement approach allows to transfer fewer files across the network. This is explained by the fact that before executing a task, algorithms 1 and 2 have to search on one hand for each task the VM on which its parent tasks have stored the maximum amount of files (because after its execution, each task stores locally i.e. on the VM where it has executed all the output files) and on the other hand, with rearrangement minimizing file transfers to the child tasks.

The use of large VMs allows several tasks to run in parallel and favours the execution of dependent tasks on the same VM, thus allowing negligible communication time between these tasks (i.e. for dependent tasks running on the same VM).

Table II shows that when the rearrangement approach is applied, 21.33%, 0.9% and 49.36% of files are respectively avoided being transferred for Epigenomics, CyberShake and Montage workflows. The platforms used in this study have a total number of cores of 384, 288 and 972 for the CyberShake, Epigenomics and Montage workflows, respectively. These platforms are multiples of 96 and are greater than the total number of cores used in parallel for each workflow.

To generate the different platforms and avoid wasting resources, i.e. avoid leasing resources that will not be used, the number of tasks that can be executed in parallel for each of the workflows is determined. To do this, the platform is oversized, i.e. this platform has as many cores as there are tasks in the workflow. In the case of the workflows used in this study, there are 1000 tasks, so the platform with 1000 cores is considered. This allowed to determine the total number of cores that could be used in parallel for each of the workflows. Thus, for CyberShake, 374 cores are used in parallel while for Epigenomics and Montage it is respectively 246 and 662 cores used in parallel. This preliminary study will be used in section V in order to determine the limit of platforms to be used for each workflow in experiments.

TABLE II. IMPACT OF REARRANGEMENT

| wf | VS | Rearrangement | | No Rearrangement | | TVF |
|---|---|---|---|---|---|---|
| | | mksp | VFT | mksp | VFT | |
| Cyb | 16 | 593.83 | 382 | 1247.98 | 386 | 400.39 |
| | 96 | 311.03 | 315 | 351.1 | 317 | |
| Epi | 16 | 34227.2 | 1221.65 | 34231.6 | 1222.13 | 1230.93 |
| | 96 | 34330.4 | 1219.81 | 34314.2 | 1219.84 | |
| Mont | 16 | 380.312 | 10.56 | 380.89 | 10.81 | 17.32 |
| | 96 | 375.44 | 8.77 | 376.1 | 9.33 | |

### B. Impact of Rearrangement's Step

To evaluate the contributions, three real-world scientific workflows from the five scientific workflows in the Pegasus Gallery[3] are used, as these three workflows (mentioned above) are data-intensive compared to the other two. These applications are:

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

This assessment start by evaluating the impact of the size of the virtual machine on the execution time of the scientific workflow. For each workflow, we consider infrastructures where we vary (i.e. increase) the maximum number of cores per VM (from 2 cores to 96 cores maximum). For a total number of cores to be used, we generate platforms with 2 cores per VM, 4 cores per VM, ..., 96 cores per VM.

On Figs. 2, 3, and 4, platforms composed of 2 cores per VM provide poor execution times compared to platforms of 32 cores per VM and themselves provide poor execution times compared to platforms of 96 cores per VM. The increasing of the total number of cores per VM provides good execution times. It is for this reason that this study favor large VMs (i.e. VMs with several cores) because these VMs can execute several tasks in parallel and considerably reducing the makespan.

In Figs. 2, 3 and 4, max_ft represents the minimum bound if there was no communication during the execution of the workflow. The execution time obtained on platforms with large VMs (i.e. having 96 cores) is approaching this theoretical minimum limit. Here, the rearrangement step (Algorithm 2) is applied to measure the performance of the initial offline planning.

Different behaviors for each of the three considered workflows are observed. First, the number of total cores used has almost no influence on the execution time for the CyberShake application while for Epigenomics, a plateau is

[3] https://pegasus.isi.edu/workflow_gallery

observed from two total cores used. For Montage, the execution time decreases up to seven total cores used. This evolution of the execution time is directly related to the level of parallelism a workflow can exploit, i.e., how many tasks can be executed concurrently. Second, 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, the CyberShake workflow which produces much more intermediate data than 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.

Fig.5 shows the impact of the rearrangement step on the execution time. For the Montage workflow, 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, rearranging the offline planning to further reduce the amount of transfers over the network can only improve the execution time.

In this study, two offline scheduling algorithms are proposed whose objective is to minimize makespan by reducing file transfers over the network. Algorithm 1 performs the mapping by only considering the files comes from the parent tasks. As for algorithm 2, it rearranges the mapping of algorithm 1 by considering the tasks level by level in order to improve the mapping of algorithm 1 by reducing the transfer of files to the child tasks.

After showing the gain of rearrangement on CyberShake and Epigenomics workflows, Fig. 6 shows more details of rearrangement on CyberShake. Approach with rearrangement gives better results compared to the approach without rearrangement. In the next part of this work we will compare this approach with the HEFT algorithm.

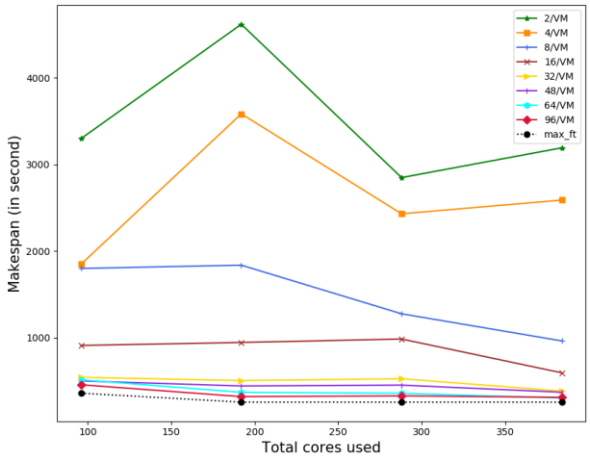


Fig. 2. Evolution of the Makespan of the CyberShake Workflow with Variation of the Total Number of Cores per VM.

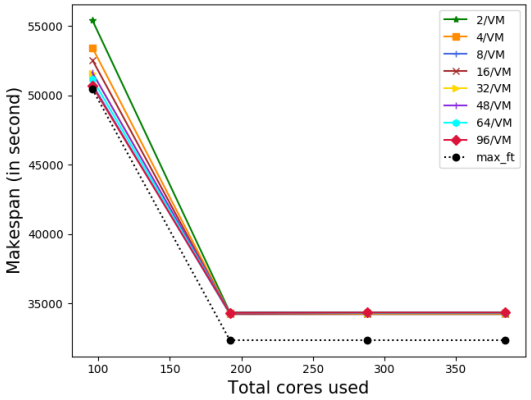


Fig. 3. Evolution of the Makespan of the Epigenomics Workflow with Variation of the Total Number of Cores per VM.

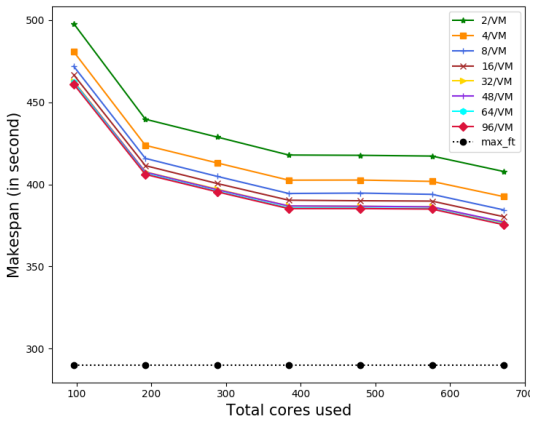


Fig. 4. Evolution of the Makespan of the Montage Workflow with Variation of the Total Number of Cores per VM.

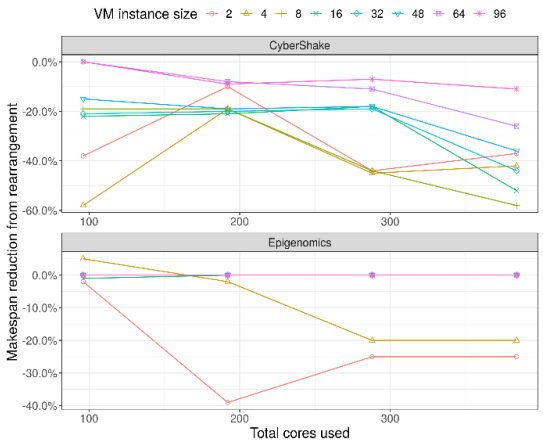


Fig. 5. Impact of Rearrangement Step on the Makespan (Execution Time) for different VM Instance Size, using Algorithm 1 vs Algoritm 2.

For data-intensive applications, such as the scientific workflow CyberShake it is essential to perform a good mapping of the different tasks, in order to have a good execution time. For this type of application, approach with rearrangement gives better results (cf. Fig.6). Since the use of large VMs has an impact on the makespan, the evaluation of both approaches is based on platforms whose total number of cores is a multiple of 96.

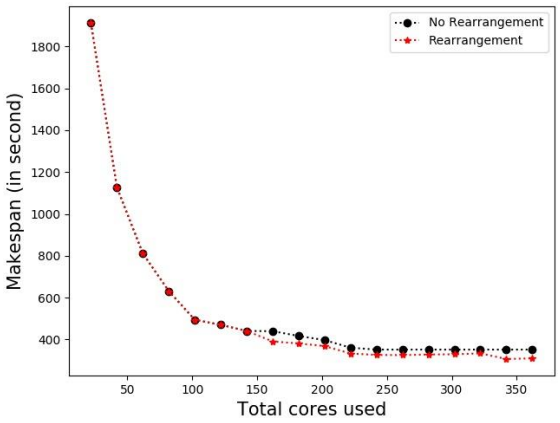


Fig. 6. Impact of Rearrangement on CyberShake Scientific Workflow.

### C. Comparaison of HEFT and WSRDT Algorithms

The main objective of this study is to minimize the execution time of an application. To achieve this goal, the major contribution of this paper is the reduction of files to be transferred during the execution of the application, which have a considerable impact on the execution time. In order to evaluate the performance of this algorithm, the proposed approach is compared to HEFT which is a very popular heuristic in the scheduling of parallel applications. HEFT is adapted to the IaaS cloud resources and to the simulation environment. The results of simulations show that approach proposed provides good results compared to HEFT.

In the results of Figs.7, 8 and 9, we use platforms whose total number of cores used varies from 2 cores to 374 cores for CyberShake, from 2 to 246 cores for Epigenomics and up to 662 cores for Montage, by increments of 2. The principle of platform generation was explained in section III.

If there are more VMs in a platform, then there will be several files to transfer on the network. It is true that large VMs are prioritized for the execution of applications, but the important element for which this choice is do, is the use full bandwidth for this type of machine that we have summarized in Table I. With the HEFT algorithm, the more VMs there are in the platform, the more files will be transferred in the network. On the other hand, with approach proposed, files are reduced on two levels for each task, i.e. the reduction coming from the parent tasks (Algorithm 1) and the reduction going to the child tasks (rearrangement of the mapping of the Algorithm 2). These two approaches allow us to have good results compared to HEFT as is the case in Figures 7, 8 and 9.

These experiences allow to show that our approach provides better results summarized through the following gains. The CyberShake workflow allows to obtain a gain of 32.22% on the makespan compared to HEFT, as for the Epigenomics workflow, a gain of 44.54% on the makespan and a gain of 18.62% on the makespan with the workflow Montage are obtained.

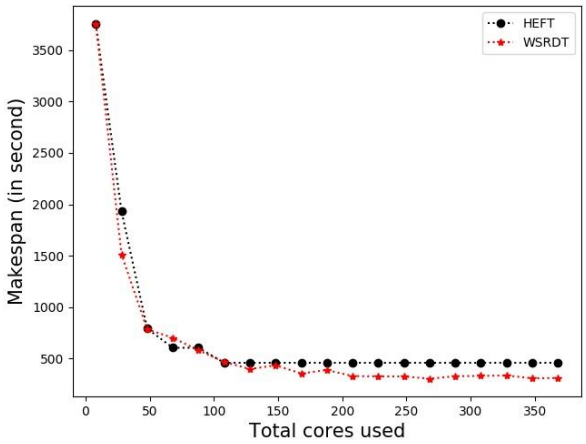


Fig. 7. Evaluation Results for the CyberShake Workflow.

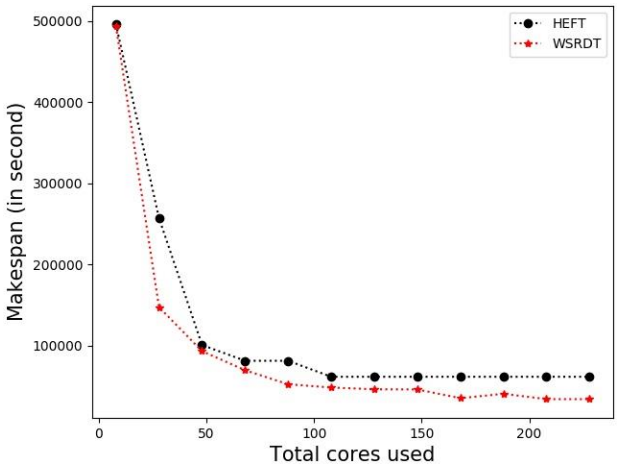


Fig. 8. Evaluation Results for the Epigenomics Workflow.

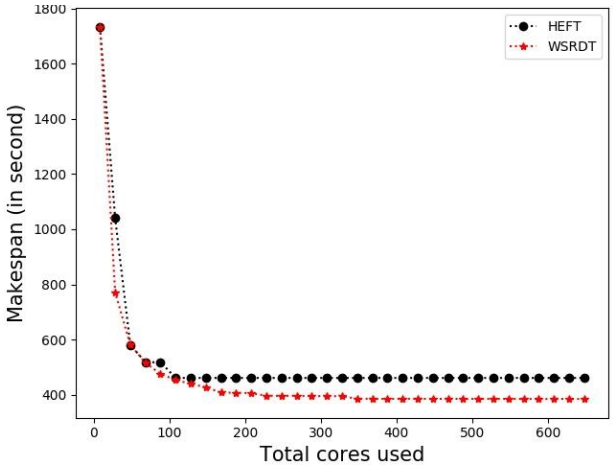


Fig. 9. Evaluation Results for the Montage Workflow.

## VI. Conclusion

Infrastructure as a Service Clouds now allows scientists to execute their data intensive workflows 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 task, usually delegated to Workflow Management Systems. A key to performance is to be able to leverage the characteristics of virtual machines instances.

In this paper, first showed the interest of using multi-core machines, because by increasing the number of cores per machine, several tasks are executed on this machine and the bandwidth linking a machine to the switch is proportional to the number of cores of the machine. Then we proposed scheduling algorithms that minimize the makespan by reducing data transfers between dependent tasks on the network. Finally, the results of the experiments showed that the proposed approach gives better results than HEFT, which is one of the best list-scheduling algorithms.

As part of our future work, we plan 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. We also plan to study the multi-objective aspect of the scheduling problem so that users can favor either a shorter execution time or a lowest cost by proposing a complementary approach where one of the objective is fixed, i.e., either a given budget or a fixed deadline.

## References

[1] J. K. Lenstra and A. H. G. Rinnooy Kan, “Complexity of Scheduling under Precedence Constraints,” Oper. Res., vol. 26, no. 1, pp. 22–35, 1978, doi: 10.1287/opre.26.1.22.

[2] V. Arabnejad, K. Bubendorfer, B. Ng, and K. Chard, “A Deadline Constrained Critical Path Heuristic for Cost-Effectively Scheduling Workflows,” in Proc. IEEE/ACM 8th Int. Conf. Utility and Cloud Computing (UCC), 2015, pp. 242–250, doi: 10.1109/UCC.2015.41.

[3] V. Arabnejad, K. Bubendorfer, and B. Ng, “A budget-aware algorithm for scheduling scientific workflows in cloud,” IEEE Int. Conf. High Perform. Comput. Commun., pp. 1188–1195, 2007.

[4] P. Lu, G. Zhang, Z. Zhu, X. Zhou, J. Sun, and J. Zhou, “A Review of Cost and Makespan-Aware Workflow Scheduling in Clouds,” J. Circuits, Syst. Comput., p. 1930006, 2018.

[5] X. Wang, B. Cao, C. Hou, L. Xiong, and J. Fan, “Scheduling budget constrained cloud workflows with particle swarm optimization,” in 2015 IEEE Conference on Collaboration and Internet Computing (CIC), 2015, pp. 219–226.

[6] L. Singh and S. Singh, “A genetic algorithm for scheduling workflow applications in unreliable cloud environment,” in International Conference on Security in Computer Networks and Distributed Systems, 2014, pp. 139–150.

[7] K. Almi’ani, Y. C. Lee, and B. Mans, “On efficient resource use for scientific workflows in clouds,” Comput. Networks, vol. 146, pp. 232–242, 2018, doi: 10.1016/j.comnet.2018.10.003.

[8] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, and S. Hu, “Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT,” Futur. Gener. Comput. Syst., vol. 93, pp. 278–289, 2019, doi: 10.1016/j.future.2018.10.046.

[9] Z. G. Chen, K. J. Du, Z. H. Zhan, and J. Zhang, “Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm,” in Proc. IEEE Congress Evolutionary Computation (CEC), 2015, pp. 708–714, doi: 10.1109/CEC.2015.7256960.

[10] H. Gamal El Din Hassan Ali, I. A. Saroit, and A. M. Kotb, “Grouped tasks scheduling algorithm based on QoS in cloud computing network,” Egypt. Informatics J., vol. 18, no. 1, pp. 11–19, 2017, doi: 10.1016/j.eij.2016.07.002.

[11] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, “Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds,” Futur. Gener. Comput. Syst., vol. 29, no. 1, pp. 158–169, 2013, doi: 10.1016/j.future.2012.05.004.

[12] A. Deldari, M. Naghibzadeh, and S. Abrishami, “CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud,” J. Supercomput., vol. 73, no. 2, pp. 756–781, 2017.

[13] H. Topcuoglu, S. Hariri, and Min-You Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” IEEE Trans. Parallel Distrib. Syst., vol. 13, no. 3, pp. 260–274, 2002, doi: 10.1109/71.993206.

[14] B. P. Rimal and M. Maier, “Workflow Scheduling in Multi-Tenant Cloud Computing Environments,” IEEE Trans. Parallel Distrib. Syst., vol. 28, no. 1, pp. 290–304, 2017, doi: 10.1109/TPDS.2016.2556668.

[15] K. Almi’ani and Y. C. Lee, “Partitioning-Based Workflow Scheduling in Clouds,” in Proc. IEEE 30th Int. Conf. Advanced Information Networking and Applications (AINA), 2016, pp. 645–652, doi: 10.1109/AINA.2016.83.

[16] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, “Characterizing and profiling scientific workflows,” Futur. Gener. Comput. Syst., vol. 29, no. 3, pp. 682–692, 2013, doi: 10.1016/j.future.2012.08.015.

[17] D. G. Feitelson and A. M. Weil, “Utilization and predictability in scheduling the IBM SP2 with backfilling,” in Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998, 1998, pp. 542–546.

[18] H. Casanova, S. Pandey, J. Oeth, R. Tanaka, F. Suter, and R. F. da Silva, “WRENCH: A Framework for Simulating Workflow Management Systems,” in 2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS), 2018, pp. 74–85.

[19] H. Casanova, “Simgrid: A toolkit for the simulation of application scheduling,” in Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001, pp. 430–437.

[20] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, “Versatile, scalable, and accurate simulation of distributed applications and platforms,” J. Parallel Distrib. Comput., vol. 74, no. 10, pp. 2899–2917, 2014.