International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

Comparison of the Execution Step Strategies on Scheduling Data-intensive Workflows on IaaS Cloud Platforms

Jean Edgard Gnimassoun
Laboratoire de Recherche en Informatique et Télécommunication
Université de San Pedro
San Pedro, Côte d'Ivoire
gnimjean@gmail.com

Armand Kodjo Atiampo
Laboratoire de Recherche en Informatique et Télécommunication
Université Virtuelle de Côte d'Ivoire
Abidjan, Côte d'Ivoire
arkodati@gmail.com

Karim Sidibé
Laboratory of Computer and Applied Sciences
Université Alassane Ouattara
Bouaké, Côte d'Ivoire
sidibekamy@gmail.com

Tchimou N'Takpé
Laboratoire de Recherche en Informatique et Télécommunication
Université Nangui Abrogoua
Abidjan, Côte d'Ivoire
tchimou.ntakpe@gmail.com

Abstract—The IaaS platforms of the Cloud hold promise for executing parallel applications, particularly data-intensive scientific workflows. An important challenge for users of these platforms executing scientific workflows is to strike the right trade-off between the execution time of the scientific workflow and the cost of using the platform. In a previous article, we proposed an efficient approach that assists the user in finding this compromise. This approach requires an algorithm aimed at minimizing the execution time of the workflow once the platform configuration is set. In this article, we compare two different strategies for executing a workflow after its offline scheduling using an algorithm. The algorithm that we proposed in the previous study has outperform the HEFT algorithm.

The first strategy allows some ready tasks to execute earlier than other higher-priority tasks that are ready later due to data transfer times. This strategy is justified by the fact that although our scheduling algorithm attempts to minimize data transfers between tasks running on different virtual machines, this algorithm does not include data transfer times in the planned execution dates for the various tasks of the workflow. The second strategy strictly adheres to the predetermined order among tasks scheduled on the same virtual machine.

The results of our evaluations show that the best execution strategy depends on the characteristics of the workflow. For each evaluated workflow, our results demonstrate that our scheduling algorithm combined with the best execution strategy surpasses HEFT. The choice of the best strategy must be determined experimentally following realistic simulations, such as the ones we conduct here using the WRENCH framework, before conducting simulations to find the best compromise between cost and execution time of a workflow on an IaaS platform..

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

I. INTRODUCTION

Data-intensive parallel applications come from various fields and are modelled by Directed Acyclic Graphs (DAGs) [1]. The execution of these data-intensive parallel applications, including thousands of tasks on large-scale distributed infrastructures, is typically managed by a Workflow Management System (WMS) [2]–[4]. Cloud IaaS platforms hold promise for the execution of such applications, given the substantial number of parallelizable computing cores and the availability of storage resources.

The description of a scientific workflow is typically independent of the specific characteristics of the infrastructure on which it will be executed. This offers the advantage of providing users with great flexibility, allowing them to execute the same workflow on different infrastructures without having to modify their application.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

A direct consequence of this flexibility is that task dependencies (where one task produces data subsequently consumed by another task) are generally managed using files. Intermediate data produced is written to a storage medium, and then the file can be transferred over the network to another storage device where the consuming task can eventually read it.

In [5] we discussed the scheduling and execution of data-intensive parallel applications on Infrastructure as a Service (IaaS) clouds. Although workflow management systems can accommodate a customized execution infrastructure for these applications, shared storage space can become a bottleneck. To solve this problem, a data-aware scheduling algorithm has been proposed that exploits the dedicated storage space (fast SSD disks) at each virtual machine to improve data localization. This reduces data transfers over the network during workflow execution. The main aim of the algorithm is to help IaaS cloud users find a good compromise between their workflow execution time and the cost of using IaaS cloud resources, by selecting sets of VM instances on the Pareto front. But long before that, we showed that to minimize execution time for a fixed number of cores, priority should be given to large VM instances (96 cores). This study showed that after the scheduling phase (offline), there could be several scenarios during the execution phase (online). The idea that we are going to develop in this study concerns the comparison of two different strategies for the execution step of a workflow after its scheduling on an IaaS platform.

This document is structured as follows. In section 2, we review the related work on scheduling scientific workflows on cloud IaaS. In section 3, we provide a description of the platform and application models used in this study. Then, in section 4, we recall the online algorithm of [5] and detail the proposed new execution strategy that respects the scheduling order, while the performances of the two execution strategies are evaluated in section 5. Finally, we conclude this article in section 6.

II. RELATED WORKS

While some studies on scientific workflow scheduling algorithms aim to minimize dynamic energy consumption [6], [7], scheduling algorithms for scientific workflows targeting IaaS clouds generally strive to find the best compromise between workflow execution time and the number (type) of virtual machine instances used for execution. Following the pay-as-you-go model of IaaS clouds, this resource quantity often corresponds to a certain cost. A typical approach involves fixing one objective as a constraint, such as a deadline [8]–[11] or a budget [12], [13] and optimizing the other objective. Some papers directly solve this bi-objective optimization problem by selecting a solution from those comprising the Pareto front [14]–[16]. These heuristics usually involve variations of scheduling and aim to find a single solution with favorable properties.

The Deadline Constrained Critical Path (DCCP) [8] is a list-based scheduling algorithm used in cloud computing. Its primary objective is to fulfill the user-defined workflow deadline while minimizing the overall execution cost. Both algorithms within DCCP share a common preprocessing step. During this initial phase, tasks are organized into different levels, each level being assigned a sub-deadline. The user-defined deadline is distributed non-uniformly among these levels, ensuring that levels with longer task execution times receive correspondingly longer sub-deadlines. However, DCCP employs a distinct approach in the task prioritization step. To enhance the efficiency of communication within the entire workflow, DCCP introduces the concept of the Constrained Critical Path (CCP). This involves assigning all tasks on a specific path to a single resource. DCCP identifies all CCPs within a workflow using a modified ranking method and compiles a list of these CCPs. During each step of the scheduling process, only the tasks within a CCP that are ready for execution are allocated to the appropriate resource, while the remaining tasks are held for subsequent steps.

The authors of [9] introduced a dynamic group learning distributed particle swarm optimization (DGLDPSO) method designed for large-scale optimization tasks. Furthermore, it extends the application of DGLDPSO to the domain of large-scale cloud workflow scheduling. DGLDPSO is particularly well-suited for addressing large-scale optimization challenges, owing to two key advantages. Firstly, it segments the entire population into numerous groups, employing a master-slave multigroup distributed model that facilitates the coevolution of these groups. This results in the formation of a distributed Particle Swarm Optimization (DPSO) system, enhancing algorithmic diversity. Secondly, DGLDPSO incorporates a dynamic group learning (DGL) strategy within DPSO, striking a balance between diversity and convergence. When DGLDPSO is employed in the context of large-scale cloud workflow scheduling, an adaptive renumbering strategy (ARS) is developed to tailor solutions to the unique characteristics of the available resources. This strategy ensures that the search process is purposeful and meaningful, rather than arbitrary. The article conducts experiments on both large-scale benchmark function sets and large-scale cloud workflow scheduling instances to evaluate the performance of DGLDPSO. Comparative analysis of the results demonstrates that DGLDPSO outperforms or, at the very least, matches the performance of other state-of-the-art large-scale optimization algorithms and workflow scheduling algorithms.

In their study, Wu et al. [10] introduced two deadline-constrained algorithms, namely Probabilistic Listing (ProLis) and L-ACO, with the objective of minimizing the makespan in cloud-based workflow scheduling. The ProLis algorithm plays a crucial role in this context by distributing deadlines to individual tasks, ranking these tasks, and subsequently allocating the necessary resources in a sequential manner to meet the Quality of Service (QoS) requirements for each task's execution. Additionally, L-ACO leverages Ant Colony Optimization (ACO) to construct various task-order lists. These lists are instrumental in identifying effective scheduling solutions that adhere to the deadline constraint while minimizing the makespan and associated costs. However, it's important to note that the study conducted by Wu et al. did not delve into addressing performance variations or consider the start-up/boot time of virtual machines (VMs) in their analysis.

In [11], the authors investigate a novel workflow scheduling model designed for heterogeneous Infrastructure-as-a-Service (IaaS) platforms. This model allows multiple tasks to execute concurrently on a virtual machine (VM) based on their varying demands for multiple resources. The authors introduce a list-scheduling framework as the foundation for this new multi-programmed cloud resource model. Within this framework,

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

tasks are assigned placements in a prioritized sequence, considering both existing and new VMs available on the platform. Different task prioritization methods and placement comparison techniques can be employed to achieve various scheduling objectives. Additionally, to leverage the diversity of IaaS platforms, VMs can be dynamically scaled up during the scheduling process. Subsequently, the authors present a deadline-constrained workflow scheduling algorithm named DyDL, built upon this framework. DyDL is designed to optimize the cost associated with workflow execution while adhering to specified deadlines. This algorithm prioritizes tasks based on their latest start times and assigns them placements that not only meet their latest start time requirements but also incur minimal cost increases.

Experimental results demonstrate that DyDL consistently outperforms several existing deadline-constrained workflow scheduling algorithms in the majority of test cases, highlighting its effectiveness in achieving superior schedules.

In the context of a hybrid cloud model [12], organizations have the flexibility to safeguard their sensitive information and critical applications within the confines of a private cloud environment while simultaneously offloading other data and applications to a public cloud when necessary. This hybrid approach offers a balance between security and scalability. To ensure the preservation of data privacy in workflow applications, the authors of this study introduce a budget-constrained hybrid cloud scheduler (BCHCS). BCHCS operates as a static heuristic scheduling algorithm, capable of making informed decisions regarding the allocation of sensitive tasks to the private cloud and utilizing the resources of the public cloud for non-sensitive tasks. The primary objective is to minimize the makespan (the total duration of the workflow) while adhering to the budget constraints imposed by the user. Notably, experimental results reveal the efficacy of this proposed method. It successfully guarantees the execution of sensitive tasks within the private cloud, while also achieving a minimum of 7 percent reduction in makespan and a higher success rate in comparison to similar existing techniques. This underscores the BCHCS's ability to strike a balance between data privacy and cost-efficiency, making it a valuable tool for organizations navigating the complexities of hybrid cloud environments.

In their research, Rizvi et al. [13] introduced a workflow scheduling policy called the "fair budget scheduling algorithm." The primary objective of this algorithm is to reduce both the computational cost and the execution time of workflows. To assess its effectiveness, the authors implemented various scientific workflows and conducted a comparative analysis of the outcomes against their proposed technique. To substantiate the efficacy of their approach, the obtained results underwent verification through the analysis of variance (ANOVA) test. The ANOVA test is a statistical method used to assess the significance of differences between multiple groups or treatments. In this context, it likely helped confirm whether the fair budget scheduling algorithm indeed produced statistically significant improvements in computational cost and execution time compared to alternative methods.

The paper [15] addresses the challenge of cloud workflow scheduling by formulating it as a multi-objective optimization problem that aims to optimize both execution time and execution cost. To tackle this problem, the authors introduce a novel multi-objective ant colony system based on a co-evolutionary multiple population for multiple objectives framework. This framework involves the use of two separate ant colonies, each dedicated to handling one of the two optimization objectives, namely execution time and execution cost. Furthermore, this approach incorporates three innovative strategies to effectively address the complexities posed by multi-objective optimization: i) a fresh pheromone update rule is introduced, which is guided by a set of nondominated solutions derived from a global archive. This update mechanism helps each ant colony to adequately search for its respective optimization objective; ii) a complementary heuristic strategy is employed to ensure that a colony doesn't exclusively focus on its individual optimization objective. Instead, it cooperates with the pheromone update rule to balance the exploration of both objectives, enhancing the overall search and iii) an elite study strategy is introduced to enhance the solution quality of the global archive. This strategy aims to bring the archive closer to the global Pareto front, thereby improving the quality of the overall solutions. The authors conducted experimental simulations using five real-world scientific workflows while considering the characteristics of the Amazon EC2 cloud platform. The results of these experiments demonstrate that the proposed algorithm outperforms both state-of-the-art multi-objective optimization approaches and constrained optimization approaches.

In their work, Zhou et al. [16] introduced the Fuzzy Dominance Sort based Heterogeneous Earliest-Finish-Time (FDHEFT) algorithm, designed to optimize the cost and makespan of workflows executed on Infrastructure-as-a-Service (IaaS) cloud platforms. This algorithm takes a fuzzy dominance sorting approach to efficiently schedule tasks.

However, the authors noted a limitation in their approach. They pointed out that these methods heavily rely on prior expert knowledge and have a static global perspective. This means that they are not well-suited to capturing the dynamic nature of workflow scheduling. In dynamic environments, where conditions and requirements change over time, relying solely on static knowledge may not adequately address the evolving needs of workflow execution. This limitation underscores the need for more adaptive and flexible scheduling approaches that can respond to real-time changes and dynamic workload conditions in cloud environments.

III. PLATFORMS AND APPLICATIONS MODELS

In this paper, our platform model is built upon a standard IaaS cloud configuration. We deploy multiple virtual machine (VM) instances within a single datacenter on physical servers. Specifically, we focus on a set of VMs similar to Amazon EC2 M5 instances [17], specifically M5d instances, which come with local storage on NVMe SSD drives. In contrast, regular M5 instances rely on the Amazon Elastic Block Storage (EBS) service for data storage. You can find detailed characteristics of the available M5d instances in Table I.

The instance series in question provides a range of virtual cores (vCPUs) from 2 to 96, each with a consistent 4GiB of memory per core. Amazon typically deploys these instances on nodes equipped with Intel Xeon Platinum 8000 series processors. The unique feature of M5d instances is the attachment of fast block-level storage on SSD drives, directly linked to the instance's lifespan. Our objective in this study is to harness this rapid storage, shared by the vCPUs within an instance but dedicated to them, for storing intermediate files generated during workflow execution. This approach minimizes

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

network data transfers for tasks scheduled on the same virtual machine, with only the workflow's entry and exit files stored on an external storage node.

The network bandwidth available to other instances or the EBS varies with the instance size. We assume that only the largest instances, capable of utilizing an entire node (i.e., with 48, 64, or 96 vCPUs), guarantee network bandwidths of 10, 20, and 25 Gbps, respectively. For smaller instances, ranging from 2 to 32 cores, the available bandwidth is proportionate to the number of cores, set at 208.33 Mbps per core. All virtual machine instances initiated for a given workflow are interconnected through a single switch.

In line with the M5d instance description, our simulated infrastructure takes into account the dedicated network link from a VM to the EBS. Regarding network connections between VMs, we assume that the bandwidth of the dedicated connection between a VM and the EBS scales with the number of cores for smaller VMs, up to 32 cores (equivalent to 218.75 Mbps per core).

While it has been shown in reference [18] that bandwidth depends on factors like file sizes, the number of files, and instance types, our simulations operate under the assumption of reliable Quality of Service (QoS) and adherence to the performance characteristics defined by the cloud provider for the allocated resources.

The scientific workflows we aim to schedule are represented as Directed Acyclic Graphs (DAGs), denoted as G = G={V,Ɛ}. In this representation, V = $\{v_i \mid i = 1,\ldots,V\}$ represents a set of vertices, which correspond to the computational tasks within the workflow, and Ɛ = $\{ e_{i,j} \mid (i,j) \in \{1,\ldots,V\} \text{ x } \{1,\ldots,V\}\}$ represents a set of edges connecting these vertices. These edges serve two primary purposes: they either signify a data dependency, indicating a file transfer requirement, or they represent a flow dependency between two tasks.

Our specific focus is on workflows that consist of a significant number of sequential tasks, each of which runs on a single core. This characteristic is typical of real scientific applications [19]. Each task within the workflow has a predefined or estimated duration, necessitates a set of *input files* to initiate its execution, and generates a set of *output files* upon completion. To describe these input and output files for a given task v_i, we use the notation $Input_i^k$ (for input files) and $Output_i^k$ (for output files), where "k" represents the file index. When an output file produced by one task v_i, is required as input by another task v_j, this creates a data dependency between v_i and v_j, which is represented by the edge $e_{i,j}$. Additionally, there are input files that are not generated by any of the tasks within the workflow, and these are referred to as the *entry files* of the workflow. These entry files serve as the starting point for the workflow's execution.

Conversely, the output files that remain unused by any task within the workflow are referred to as the exit files of the workflow. To facilitate the scheduling process, two quantities are defined for each task within the workflow. These quantities are crucial for making scheduling decisions: the *Local Input Volume* of task v_i on machine VM_j, denoted as $LIV_{i,j}$, is calculated as the cumulative size of input files that task v_i requires, and these input files are locally available on VM_j; the *Local Output Volume* of task v_i on machine VM_j, denoted as $LOV_{i,j}$, represents the cumulative size of output files produced by task v_i. These output files are utilized by the successors of task v_i, and these successors are also scheduled on VM_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.

In the workflow execution process, all intermediate files, which are files generated by one task and used by another, are stored locally on the SSD storage of one or possibly multiple machines. In contrast, the entry and exit files of the workflow are stored on the EBS (Elastic Block Store) service, which is accessible by all the machines involved in the workflow. 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. Execution Step Strategies

In our previous study [5], we introduced a two-step offline algorithm for scientific workflow scheduling. In the first step, the goal is to find, for each task in the workflow, the machine that will execute it earliest and store the maximum of its input files. The second step, called the rearrangement step, traverses the workflow level by level, from bottom to top. During the initial placement, which is performed from top to bottom, only data volumes from direct predecessors of a task are considered. It's not possible to account for the data localization required by a direct descendant of a task until the scheduling of that task is determined. This can lead to data transfers that could be avoided.

The objective of this algorithm is to minimize the execution time (makespan), taking into consideration not only the parallel execution of certain tasks but also the reduction of data transfers over the network. In other words, this study does not include data transfer time, as the algorithm aims to avoid transfers over the network. However, this is not always feasible due to the complexity of dependencies that exist between tasks.

The online phase of our previous study [5] is based on the algorithm shown in Algorithm 1, where the bottom level (bl_i) (line 1) is computed during the offline phase. Ready tasks are sorted by priority (line 2), which takes into account the start time (st_i), the bottom level (bl_i), and the task identifier. For each ready task (lines 3 - 9), the VM on which it was scheduled to execute is determined during the offline phase [5] (line 4). If a free processor exists on this VM (line 5), the task executes on the VM (line 6) without considering other tasks, and the count of free processors is updated on the VM (line 7).

Algorithm 1

```
1: Compute bl_i for each task v_i
2: Sort ready_tasks by priority
3: for all v_i ∈ ready_tasks do
4:     VM_k ← mapping of v_i
5:     if proc_k > 0 then
6:         execute v_i on VM_k
7:         update proc_k
8:     endif
9: endfor
```

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

TABLE I. CHARACTERISTICS OF THE AWS M5D INSTANCES.

Model	vCPU	Memory (GiB)	Instance Storage (GiB)	Network Bandwidth (Gbps)	EBS Bandwidth (Mbps)
m5d.large	2	8	1 x 75 NVMe SSD	Up to 10	Up to 3,500
m5d.xlarge	4	16	1 x 150 NVMe SSD	Up to 10	Up to 3,500
m5d.2xlarge	8	32	1 x 300 NVMe SSD	Up to 10	Up to 3,500
m5d.4xlarge	16	64	2 x 300 NVMe SSD	Up to 10	3,500
m5d.8xlarge	32	128	2 x 600 NVMe SSD	10	5,000
m5d.12xlarge	48	192	2 x 900 NVMe SSD	10	7,000
m5d.16xlarge	64	256	4 x 600 NVMe SSD	20	10,000
m5d.24xlarge	96	384	4 x 900 NVMe SSD	25	14,000

The principle of Algorithm 2 is to avoid executing a task that could delay another more prioritized task. Similar to Algorithm 1, Algorithm 2 begins by sorting all ready tasks by priority (line 2). For each ready task (lines 3 - 16), if there is at least one available processor on the VM designated to execute the task (line 5), then we have two options. The first option is to execute this ready task and update the processor count if its $ready\_time$ equals its calculated $start\_time$ (lines 5 - 8). The calculated $start\_time$ (line 6) does not take data transfer time into account. As for the second option (lines 9 - 14), if the $ready\_time$ of a ready task is strictly less than its calculated $start\_time$ (line 9) and all tasks (v_k) whose calculated $finsh\_time$ equals the calculated $start\_time$ of the ready task (v_i) have finished their execution (*i.e.*, released at least one processor) (line 10), we execute v_i and update the count of available processors (line 11 - 12). Note that the calculated $finish\_time$ and $start\_time$ do not account for data transfer time, but rather the data transfer volumes.

V. DISCUSSION

A. *Comparative analysis of two strategies*

To evaluate the two execution strategies, we employed two data-intensive parallel applications (scientific workflows) from the Pegasus Gallery [20]. These workflows are designed to represent real-world scientific applications.

The characteristics of these applications are provided in Table II. For applications that do not generate a significant number of files during their execution (such as Montage), the results of Algorithms 1 and 2 remain identical.

Algorithm 2

1: Compute bl_i for each task v_i
2: Sort $ready\_tasks$ by priority
3: **for all** $v_i \in ready\_tasks$ **do**
4: $VM_k \leftarrow$ mapping of v_i
5: **if** $proc_k > 0$ **then**
6: **if** $rt_i = st_i$ **then**
7: execute v_i on VM_k
8: update $proc_k$
9: **else if** $rt_i < st_i$ **then**
10: **if** all tasks $v_j \mid ft_j = st_j$ are computed **then**
11: Execute v_i on VM_k
12: update $proc_k$
13: **end if**
14: **end if**
15: **end if**
16: **end for**

TABLE II. 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

- 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.

Our simulations are based on WRENCH 1.5-83d60e and SimGrid 3.23.3-f2ae928, whose source code is available at the following address[1].

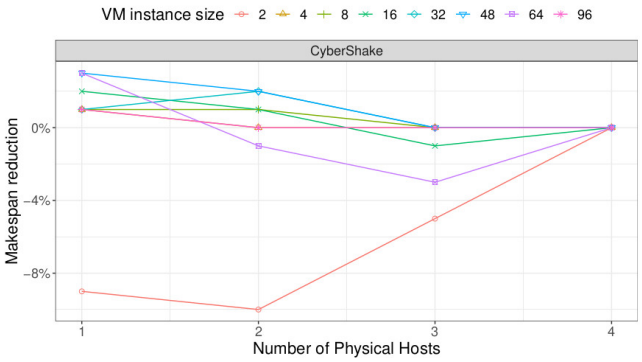
In this section, we evaluate ours execution algorithms, Algorithm 1 (from the previous study [5]) and Algorithm 2 (for this new study), considering the scheduling algorithms of N'Takpé et al. [5] and HEFT [21]. The original HEFT algorithm uses Algorithm 2; we combine the principles of HEFT with Algorithm 1 to assess its impact.

For each workflow, we consider infrastructures composed of different numbers of physical hosts, each with 96 cores. For each number of physical hosts, we impose the use of a specific instance size among those described in Table I. Bandwidth depends on the number of cores per VM, so if VMs have fewer cores, the execution times are higher.

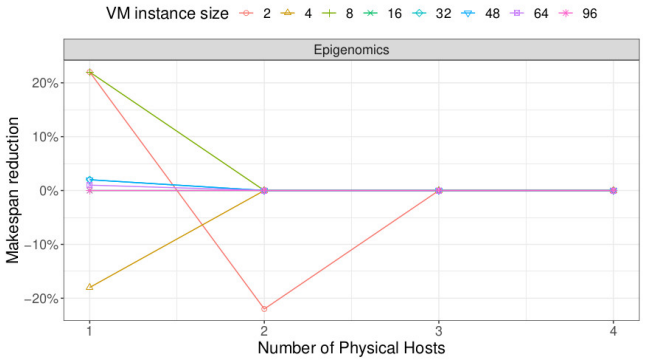
Figure 1 illustrates the gain of execution time considering the algorithm proposed in [5]. Algorithm 2 has been specifically designed to minimize transfer times between workflow tasks. It takes into account communication constraints and task dependencies to efficiently execute task scheduling. Therefore, it is reasonable that Algorithm 2 reduces transfer times compared to Algorithm 1 with the Epigenomics workflow (Fig. 1b), which may not be as sophisticated in managing data transfers. Algorithm 2 confirms the study in [5], which emphasizes data locality, i.e., reducing data transfers between compute nodes, to minimize transfer times. For the CyberShake workflow (Fig. 1a), Algorithm 1 provides better results mainly on platforms with 2 cores per VM.

[1] github.com/GnimEd/gasn

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

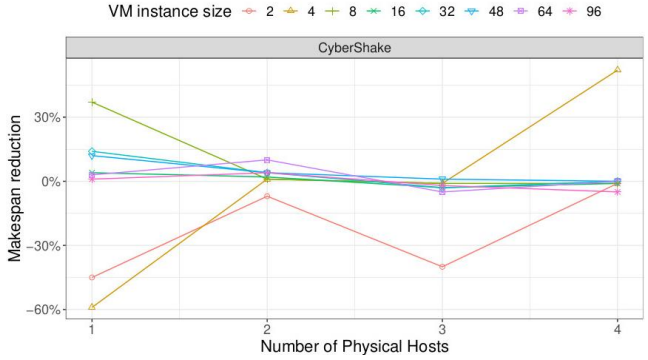


(a) Gain between our scheduling algorithm combined with Algorithm 1 versus Algorithm 2.

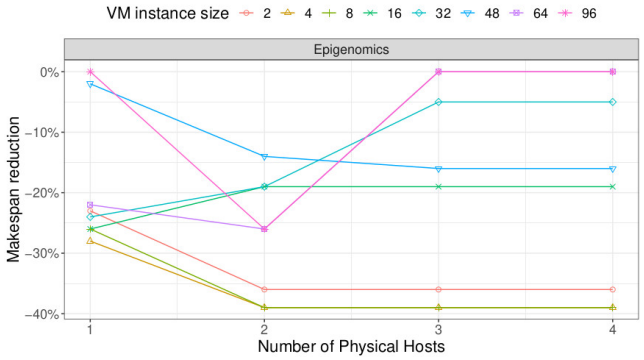


(b) Gain between our scheduling algorithm combined with Algorithm 2 versus Algorithm 1.

Figure 1. Makespan reduction algorithm 1 versus algorithm 2.



(a) Gain between HEFT combined with Algorithm 1 versus HEFT combined with Algorithm 2.



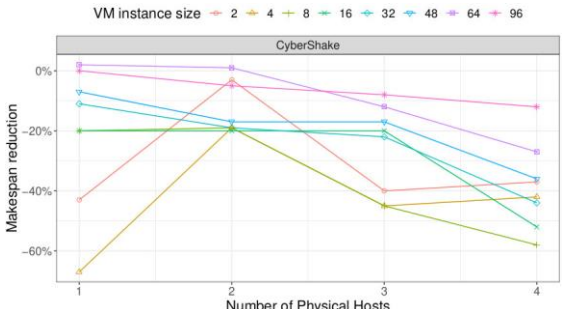
(b) Gain between HEFT combined with Algorithm 2 versus HEFT combined with Algorithm 1.

Figure 2. Makespan reduction for HEFT algorithm.

Through Figure 2, we compare the gain of the makespan obtained from HEFT [21] using the two Algorithms (1 and 2) of execution. The results of Fig. 2a show that Algorithm 2 gives better results with the CyberShake workflow mainly for platforms with 2 cores per VM. This is due to several factors. First, the Algorithm 2 can more efficiently take advantage of the parallelism offered by platforms with 2 cores per VM, which allows it to reduce the overall execution time of the workflow. Second, the Algorithm 2 can achieve strict task scheduling, more appropriate for the CyberShake workflow, taking into account specific characteristics of the latter, such as task dependencies and resource requirements. It is important to note that these observations are specific to the CyberShake workflow and platforms with 2 cores per VM. The performance of the two algorithms may vary depending on the characteristics of the workflow and the platforms used. However, in our case study, the Algorithm 2 turns out to be more efficient in terms of makespan reduction for the CyberShake workflow on platforms with 2 cores per VM. While in Fig. 2b, we observe a mixed result from the Algorithm 2 on the Algorithm 1, with the Epigenomics workflow.

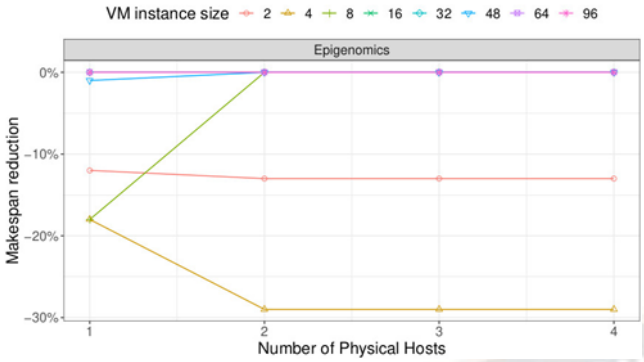
Through the Figure 3, we highlight the reduction of the execution time of our scheduling algorithm [5] and HEFT. For the CyberShake workflow, our scheduling algorithm combined with the Algorithm 1 gives better results compared to HEFT combined with the Algorithm 1 (see Fig. 3a) on all platforms.

The reductions on the one hand in the volumes of data on the network and on the other hand in the transfer time of these data have an impact on the makespan. Fig. 3b illustrates the gain on the execution time of our scheduling algorithm combined with the Algorithm 2 compared to the HEFT scheduling algorithm combined with the Algorithm 2. We notice a significant gain on platforms with 2 and 4 cores per VM. This reduction is explained by the fact that if there are fewer cores per VM, this implies higher transfer times because the bandwidths are proportional to the number of cores per VM (cf. Table I).



(a) Gain between our scheduling algorithm combined with Algorithm 1 versus HEFT combined with Algorithm 1.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*



(b) Gain between our scheduling algorithm combined with Algorithm 2 versus HEFT combined with Algorithm 2.

Figure 3. Makespan reduction for our scheduling algorithm versus HEFT.

B. Limitations

The results of the study are highly dependent on the specific characteristics of the workflows evaluated. Each workflow may have a different structure, size and resource requirements, which means that the conclusions drawn from the study may not be generalizable to other types of workflows. Consequently, it is essential to recognize that the effectiveness of strategies and algorithms may vary according to the nature of the workflows.
The study points out that the choice of algorithm depends on the type of workflows. This means that users must select the algorithm according to the specific characteristics of their workflows. However, this adds operational complexity, as there is no single solution for all types of workflows, which can make implementation of the approach less practical in real environments.

One of the study's implementation strategies is based on the assumption that data transfer times are not taken into account. This assumption may not be realistic in all situations, as data transfers between tasks can have a significant impact on actual workflow performance. By neglecting these transfer times, the study could underestimate actual costs and execution times.
The results of the study are mainly based on simulations. Although simulations are useful for controlled experiments, they may not fully reflect the complexity of real cloud environments. Results obtained in simulation may differ significantly from what happens in real conditions due to unforeseen and unmodeled behavior.

The study does not explicitly address the scalability of the proposed approach. This omission is important because the effectiveness of an approach can vary according to the size and complexity of workflows. Failure to consider scalability may limit the applicability of the approach on a large scale.
In summary, these limitations underline the importance of taking into account the specific characteristics of workflows, the realism of assumptions, validation in real environments, and consideration of scalability when designing and applying the proposed approach.

VI. CONCLUSION AND FUTURE WORK

The makespan represents the total time required to complete all tasks. By choosing the version of the algorithm best suited to the type of workflow, we can aim for a significant reduction in makespan. This translates into a faster execution of the workflow, enabling the desired results to be achieved in a shorter time.

Makespan has a direct impact on the cost of using cloud resources, which is billed on a per-use basis. By choosing the right version of the algorithm, we can optimize the use of available resources, avoiding waste and unnecessary expenditure. By choosing the best version of the algorithm for the type of workflow, we can achieve an optimum compromise between makespan and the cost of using cloud resources.
As part of our future work, we aim to validate the effectiveness of our proposed algorithms, we intend to conduct comparisons between simulated executions and real-world runs on the AWS (Amazon Web Services) cloud platform, specifically utilizing M5d instances. This empirical validation will help us confirm the practical impact and performance of the algorithms. Another avenue of research involves exploring a multi-objective approach where one of the optimization objectives is fixed. In other words, we'll investigate scenarios where either a predefined budget or a fixed deadline is set as a constraint, and the scheduling algorithm operates within these bounds to optimize the other objective.

REFERENCES

[1] I. Taylor, E. Deelman, D. Gannon, et M. Shields, Workflows for e-Science: Scientific Workflows for Grids. Springer Publishing Company, Incorporated, 2014.
[2] E. Deelman et al., « Pegasus, a Workflow Management System for Science Automation », Future Generation Computing Systems, vol. 46, p. 17-35, 2015.
[3] J. Liu, E. Pacitti, P. Valduriez, et M. Mattoso, « A Survey of Data-Intensive Scientific Workflow Management », J Grid Computing, vol. 13, nº 4, p. 457-493, déc. 2015, doi: 10.1007/s10723-015-9329-8.
[4] R. Ferreira Da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, et E. Deelman, « A characterization of workflow management systems for extreme-scale applications », Future Generation Computer Systems, vol. 75, p. 228-238, oct. 2017, doi: 10.1016/j.future.2017.02.026.
[5] T. N'Takpé, J. Edgard Gnimassoun, S. Oumtanaga, et F. Suter, « Data-aware and simulation-driven planning of scientific workflows on IaaS clouds », Concurrency and Computation: Practice and Experience, vol. 34, nº 14, p. e6719, 2022.
[6] T. Shu et C. Q. Wu, « Energy-Efficient Mapping of Large-Scale Workflows Under Deadline Constraints in Big Data Computing Systems », Future Generation Computer Systems, vol. 110, p. 515-530, 2020, doi: 10.1016/j.future.2017.07.050.
[7] J. Thaman et M. Singh, « Green cloud environment by using robust planning algorithm », Egyptian Informatics Journal, vol. 18, nº 3, p. 205-214, nov. 2017, doi: 10.1016/j.eij.2017.02.001.
[8] V. Arabnejad, K. Bubendorfer, B. Ng, et K. Chard, « A Deadline Constrained Critical Path Heuristic for Cost-Effectively Scheduling Workflows », in Proceedings of the 8th IEEE/ACM International Conference on Utility and Cloud Computing (UCC), Limassol, Cyprus, déc. 2015, p. 242-250.
[9] Z.-J. Wang et al., « Dynamic Group Learning Distributed Particle Swarm Optimization for Large-Scale Optimization and

Its Application in Cloud Workflow Scheduling », IEEE Transactions on Cybernetics, vol. 50, nº 6, p. 2715-2729, 2020, doi: 10.1109/TCYB.2019.2933499.

[10] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia, et J. Wen, « Deadline-Constrained Cost Optimization Approaches for Workflow Scheduling in Clouds », IEEE Trans. Parallel Distrib. Syst., vol. 28, nº 12, p. 3401-3412, déc. 2017, doi: 10.1109/TPDS.2017.2735400.

[11] Z. Zhu et X. Tang, « Deadline-constrained workflow scheduling in IaaS clouds with multi-resource packing », Future Generation Computer Systems, vol. 101, p. 880-893, déc. 2019, doi: 10.1016/j.future.2019.07.043.

[12] A. Rezaeian, H. Abrishami, S. Abrishami, et M. Naghibzadeh, « A Budget Constrained Scheduling Algorithm for Hybrid Cloud Computing Systems Under Data Privacy », in Proceedings of the 2016 IEEE International Conference on Cloud Engineering (IC2E), Berlin, Germany, avr. 2016, p. 230-231.

[13] N. Rizvi et D. Ramesh, « Fair budget constrained workflow scheduling approach for heterogeneous clouds », Cluster Comput, vol. 23, nº 4, p. 3185-3201, déc. 2020, doi: 10.1007/s10586-020-03079-1.

[14] M. Hosseinzadeh, M. Y. Ghafour, H. K. Hama, B. Vo, et A. Khoshnevis, « Multi-Objective Task and Workflow Scheduling Approaches in Cloud Computing: a Comprehensive Review », J Grid Computing, vol. 18, nº 3, p. 327-356, sept. 2020, doi: 10.1007/s10723-020-09533-z.

[15] Z.-G. Chen et al., « Multiobjective Cloud Workflow Scheduling: A Multiple Populations Ant Colony System Approach », IEEE Transactions on Cybernetics, vol. 49, nº 8, p. 2912-2926, 2019, doi: 10.1109/TCYB.2018.2832640.

[16] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, et S. Hu, « Minimizing Cost and Makespan for Workflow Scheduling in Cloud Using Fuzzy Dominance Sort Based HEFT », Future Generation Computer Systems, vol. 93, p. 278-289, avr. 2019.

[17] Amazon Elastic Compute Cloud (EC2), https://aws.amazon.com/ec2. Accessed 29 May 2023

[18] R. Mathá, S. Ristov, T. Fahringer, et R. Prodan, « Simplified Workflow Simulation on Clouds based on Computation and Communication Noisiness », IEEE Trans. Parallel Distrib. Syst., vol. 31, nº 7, p. 1559-1574, juill. 2020, doi: 10.1109/TPDS.2020.2967662.

[19] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, et K. Vahi, « Characterizing and Profiling Scientific Workflows », Future Generation Computer Systems, vol. 29, nº 3, p. 682-692, 2013.

[20] Pegasus Workflow Gallery, http://pegasus.isi.edu/workflow_gallery. Accessed 13 May 2023

[21] H. Topcuoglu, S. Hariri, et M.-Y. Wu, « Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing », IEEE Transactions on Parallel and Distributed Systems, vol. 13, nº 3, p. 260-274, 2002, doi: 10.1109/71.993206.