International Journal of Mathematics and Computer Research

ISSN: 2320-7167

Volume 12 Issue 11 November 2024, Page no. –4553-4561

Index Copernicus ICV: 57.55, Impact Factor: 8.316

DOI: 10.47191/ijmcr/v12i11.01

Efficient Workflow Scheduling for Minimizing Data Transfers and Enhancing Resource Utilization in Cloud IaaS Platforms

Jean Edgard GNIMASSOUN[1], Akanza Konan RickyN'DRI[2], Dagou Dangui Augustin Sylvain Legrand KOFFI[3]

[1]Université de San Pedro, San Pedro, Côte d'Ivoire

[2]Université Alassane Ouattara, Bouaké, Côte d'Ivoire

[3]Ecole Supérieure Africaine des Technologies d'Information et de Communication, Abidjan, Côte d'Ivoire

ARTICLE INFO

Corresponding Author:
Jean Edgard GNIMASSOUN

ABSTRACT

Cloud IaaS platforms readily provide access to homogeneous multi-core machines, whether they are physical ("bare metal") or virtual machines. Each of these machines can be equipped with high-performance SSD disks, enabling the distribution of workflow-generated files across multiple machines, which helps minimize the overhead associated with data transfers. In this paper, we propose a scheduling algorithm called SMDT-ERU (Scheduling for Minimizing Data Transfer - Enhancing Resource Utilization), designed to reduce the makespan of data-intensive workflows by minimizing data transfers between dependent tasks over the network. Intermediate files generated by tasks are stored locally on the disk of the machine where the tasks are executed.

Through experimentation, we confirm that increasing the number of cores per machine reduces the additional costs caused by network data transfers. Real-world workflow experiments demonstrate the advantages of the proposed algorithm. Our data-driven scheduling approach significantly reduces execution time and the volume of data transferred over the network, outperforming one of the leading state-of-the-art algorithms, which we have adapted to fit our assumptions.

KEYWORDS: Workflow scheduling; makespan reduction; IaaS Cloud

I. INTRODUCTION

Data-intensive parallel applications (scientific workflows), represented as Directed Acyclic Graphs (DAGs) [1], [2], originate from various fields such as biology, astronomy, and physics. They are characterized by their complex structure and high demands for computing and storage resources. The transition from grid environments to cloud computing has consistently shaped the execution of scientific workflows, which now leverage virtual, dynamic, and scalable resources. These workflows allow researchers to express the necessary steps to transform vast amounts of data generated by scientific experiments into meaningful scientific results. Typically, the execution of these data-intensive applications, composed of hundreds of computational tasks, is managed by a Workflow Management System (WMS) [3], which abstracts the complexities of resource selection, data management, and computation scheduling for the end user.

However, with the emergence of major cloud service providers such as Amazon, Google, and Microsoft, Infrastructure as a Service (IaaS) has become a viable alternative to traditional clusters and grids. IaaS combines the advantages of these systems by offering virtually unlimited resources with flexible accessibility. This capability allows WMS to design computing and storage infrastructures tailored to specific workflows, leveraging a targeted set of virtual machine instances.

One notable advantage of scientific workflows is their independence from the specific characteristics of the underlying infrastructure. This flexibility enables users to execute the same workflow on different infrastructures without modifying their applications. However, this also leads to the common practice of managing task dependencies through file transfers. Intermediate data produced by a task is typically written to disk and then transferred over the network to another storage device for consumption by another task.

While existing algorithms address the scheduling of workflows on IaaS cloud infrastructures, many do not account for the complexities associated with executing tasks on multi-core virtual machines. This oversight can lead to unnecessary data transfers, adversely affecting overall execution time. Given that Amazon EC2 allows the deployment of virtual machines with up to ninety-six parallel computing cores [4], our work aims to leverage this capability. By placing dependent tasks on the same virtual machine, we can effectively reduce communication time, which is crucial for improving workflow efficiency.

The challenge of mapping workflow tasks in a cloud computing environment remains a subject of extensive research, with several heuristics and metaheuristics proposed to optimize execution time and resource utilization. However, many existing approaches overlook the importance of data locality, focusing primarily on data produced by predecessor tasks. By considering both the execution time of individual tasks and the data transfer time between dependent tasks, our research aims to enhance workflow execution. The central question addressed in this paper is how to effectively map the tasks of a data-intensive application while minimizing file transfers over the network to achieve better execution times.

In this paper, we propose an innovative approach that leverages the unique characteristics of Amazon Web Services virtual machine instances, specifically their large number of cores and dedicated storage space on fast SSD drives. Our goal is to improve data locality, thereby reducing data transfers over the network during workflow execution, which should have a direct and beneficial impact on overall execution time.

II. RELATED WORKS

Cloud computing has significantly changed the way scientific and industrial workflows are executed, providing scalable and elastic resources to handle complex tasks. However, optimizing the scheduling of workflows in these environments remains a major challenge, especially when it comes to minimizing conflicting objectives such as makespan and execution costs while adhering to quality of service (QoS) constraints. Various algorithms and approaches have been developed to address these requirements, utilizing multi-objective methods, heuristics, and artificial intelligence (AI).

A systematic review of task scheduling algorithms in cloud computing was conducted by Krishna and Mangalampalli [5], highlighting the different methods used to manage workflow complexity. The study classified the algorithms based on their ability to balance workloads, optimize resources, and handle time constraints. This in-depth review provides a solid foundation for understanding how both classical and advanced algorithms meet the increasingly complex needs of cloud infrastructures. Yang et al. [6], in their work on classification-based workflow scheduling in cloud environments, proposed an approach that considers the diversity of workflows. Their method optimizes scheduling by classifying tasks based on their characteristics, thereby offering better resource management in dynamic environments. This approach is particularly effective in multi-tier environments where tasks vary significantly in terms of complexity and resource requirements.

Resource optimization in Infrastructure as a Service (IaaS) environments is crucial to improving overall system performance. Zhu and Tang [7] proposed a workflow scheduling approach for deadline-constrained tasks in IaaS clouds, using multi-resource packing. Their method aims to maximize resource utilization while ensuring that task deadlines are met, highlighting the challenges of managing heterogeneous resources in distributed environments. Other studies have focused on enhancing existing algorithms to adapt to hybrid cloud-edge environments. For instance, Alsadie and Alsulami [8] proposed a modified Firefly algorithm to improve workflow efficiency in hybrid cloud-edge environments. This solution reduces latency and improves task distribution by leveraging both local and remote computing resources.

Improving classical scheduling algorithms remains an area of interest for many researchers. Murad et al. [9] presented an optimized version of the Min-Min algorithm for scientific workflow scheduling in the cloud, focusing on minimizing task waiting times. Their approach offers a notable reduction in workflow execution times, particularly in compute-intensive environments. Additionally, Patil and Thankachan [10] conducted a comparative study of various scheduling algorithms in cloud environments, highlighting the strengths and weaknesses of each method based on the size and complexity of workflows.

Sukhoroslov [11] studied the scheduling of workflows with specific resource requirements in cluster environments. His work emphasizes the importance of matching the resource needs of tasks with the capabilities of available compute nodes. This approach improves task distribution and optimizes resource usage in high-performance computing (HPC) environments.

Multi-objective optimization is one of the most studied approaches for workflow scheduling, as it allows for simultaneous consideration of multiple criteria. Akraminejad et al. [12] proposed a crow search algorithm to optimize both makespan and costs in scientific cloud workflows. Similarly, Bansal and Aggarwal [13] explored the hybridization of PSO and WOA algorithms in cloud-fog environments to improve scheduling in a multi-objective framework. These works clearly demonstrate the importance of multi-objective optimization in the effective allocation of resources in large-scale cloud environments. Raeisi-Varzaneh et al. [14] also contributed to this research by introducing an improved Max–Min algorithm focused on cost reduction during task allocation in workflows, further emphasizing that resource management is crucial for ensuring efficient execution in large-scale cloud settings.

Alongside multi-objective approaches, heuristic algorithms remain widely used due to their ability to quickly provide approximate solutions in complex environments. Sun et al. [15] introduced a hybrid algorithm, ET2FA, which combines

heuristic techniques with resource adjustment methods to optimize scheduling for deadline-constrained workflows. This approach significantly reduced task execution times. Complementarily, Mangalampalli et al. [16] proposed a deep reinforcement learning-based scheduling algorithm (DRLBTSA), demonstrating that reinforcement learning algorithms can learn from execution data to optimize task allocation in the cloud. These works highlight the growing integration of AI and machine learning techniques in workflow optimization.

Comparative studies of the performance of different scheduling algorithms have also highlighted the diversity of existing solutions and their respective advantages. Kumar and Chander [17] conducted a comparative analysis of major task scheduling algorithms in cloud environments, emphasizing that hybrid methods, which combine multiple techniques, often yield the best performance. Concurrently, Sadotra et al. [18] focused on scheduling in the context of green computing, demonstrating that scheduling algorithms can be optimized to reduce the energy footprint of cloud systems, which has become an increasing concern in light of environmental considerations.

Another critical aspect of workflow scheduling in the cloud is resource management under constraints. Cloud environments are often characterized by limited resource availability, and task execution times can be uncertain. Taghinezhad-Niar et al. [19] addressed this issue by proposing an approach that takes into account uncertainties in task execution times while ensuring acceptable QoS levels. This approach allows for better resource management in unpredictable environments. Additionally, Sun et al. [20] introduced a resource optimization method called multi-resource packing, which maximizes resource utilization while adhering to time constraints. These works demonstrate that fine-grained resource management is essential for effective workflow planning.

Finally, the integration of artificial intelligence into workflow management is becoming increasingly important. Mustapha and Gupta [21] introduced a DBSCAN-inspired task scheduling algorithm for cloud infrastructures, illustrating how clustering techniques can enhance resource allocation. This approach utilizes the density of data clusters to optimize task distribution and avoid resource overloads. These new AI-based methods pave the way for more dynamic and adaptive solutions for scheduling in complex cloud environments. Heuristic algorithms and AI-based approaches continue to play a key role in optimizing workflow performance. Sandhu et al. [22] explored optimization strategies to enhance task scheduling performance in cloud computing. Their research demonstrated that integrating heuristics into scheduling processes led to significant improvements in response times and processing costs. Similarly, Mikram et al. [23] proposed a new hybrid algorithm, HEPGA, which combines evolutionary algorithms with knowledge-based scheduling approaches to solve the scheduling problem in cloud environments. Their approach has shown increased efficiency in resource management and the optimization of complex scientific workflows.

In summary, optimizing workflow scheduling in cloud environments continues to prompt intensive research, with significant contributions in the fields of heuristic algorithms, multi-objective approaches, and AI-based methods. Challenges related to resource management, cost minimization, and makespan improvement remain critical, and hybrid or AI-based solutions have proven promising in addressing these issues in ever-evolving cloud environments.

III. PLATFORM AND APPLICATIONS MODELS

In this paper, we construct our platform model based on a standard IaaS cloud configuration. A variety of virtual machine (VM) instances are deployed across physical servers within a single datacenter. Specifically, we focus on a range of VMs comparable to Amazon EC2 M5 instances. In particular, we examine M5d instances, which come equipped with local storage on NVMe SSD drives, unlike the standard M5 instances that rely on Amazon Elastic Block Storage (EBS) for data storage.

Table ***I*** outlines the specifications of the available M5d instances.

The number of virtual cores (vCPUs) within this instance series spans from 2 to 96, maintaining a fixed memory allocation of 4GiB per core. Amazon typically deploys these instances on nodes that feature Intel Xeon Platinum 8000 series processors. A distinctive aspect of the M5d instances is the inclusion of fast block-level storage on SSD drives, which is tied to the instance's lifespan. Our objective in this study is to utilize this rapid storage, shared among the vCPUs yet dedicated to them, for storing intermediate files generated during workflow execution. This approach aims to minimize data transfers across the network for tasks allocated to the same virtual machine. Only the input and output files of the workflow will be saved on an external storage node.

The network bandwidth with other instances or the EBS is contingent on the instance size. We assume that only the largest instances capable of utilizing a full node—specifically those with 48, 64, or 96 vCPUs—are guaranteed network bandwidths of 10, 20, and 25 Gbps, respectively. For smaller instances, ranging from 2 to 32 cores, we consider the available bandwidth to be proportional to the number of cores, set at 208.33 Mbps per core. All virtual machine instances initiated for executing a specific workflow are interconnected via a single switch.

Regarding the M5d instances, the connection from a VM to the EBS is established through a dedicated network link, which we incorporate into our simulated infrastructure. For the network connections between VMs, we presume that the bandwidth of the dedicated connection between a VM and the EBS is proportional to the number of cores for smaller VMs with up to 32 cores (i.e., 218.75 Mbps per core).

The scientific workflows we aim to schedule are modeled as Directed Acyclic Graphs (DAGs), denoted as G = { T, $\mathcal{E}$}, where T = {t^i | i = 1,…, T} represents the set of vertices corresponding to the computational tasks of the workflow, and $\mathcal{E}$ = {$e^{i,j}$ | (i,j) ϵ {1,…,T} x {1,…,T}} is the set of edges between these vertices, indicating either data dependencies (i.e., file transfers) or control flows between tasks. Our focus is on workflows composed of a large number of sequential tasks that run on a single core, as this mirrors real-world scientific applications. Each task within the workflow has a predefined (estimated) execution time, requires specific input files to begin, and generates output files upon completion.

Our primary focus is on workflows composed of a substantial number of sequential tasks, each of which executes on a single core. This pattern is common in real-world scientific applications. Each task within the workflow has a predefined or estimated duration, requires a set of input files to begin its execution, and produces a set of output files upon completion.

To represent the input and output files for a given task t^i, we use the notation $Input_k^i$ for input files and $Output_k^i$ for output files, where k denotes the file index.

When an output file generated by one task t^i is needed as an input by another task t^j, this establishes a data dependency between t^i and t^j, represented by the $e^{i,j}$. Furthermore, there are input files that are not produced by any tasks within the workflow; these are referred to as the workflow's entry files and serve as the starting point for its execution.

Conversely, the output files that are not required by any subsequent tasks in the workflow are referred to as the exit files.

To aid in the scheduling process, two important quantities are defined for each task in the workflow. These metrics are essential for making effective scheduling decisions: the Local Input Volume (LIV) of task t^i on machine vm_k, denoted as LIV_k^i, represents the total size of the input files required by task t^i that are locally available on vm_k; similarly, the Local Output Volume (LOV) of t^i on machine vm_k, denoted as LOV_k^i, represents the cumulative size of the output files generated by task t^i. These output files are needed by the successors of task t^i, and the successors are also scheduled on vm_k. If a file is used by multiple successors, its size is counted as many times as there are successors. The LIV of an entry task and the LOV of an exit task are set to zero by definition.

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

During workflow execution, all intermediate files those generated by one task and consumed by another are stored locally on the SSD storage of one or more machines. In contrast, the entry and exit files of the workflow are stored on the Elastic Block Store (EBS) service, which is accessible to all machines involved in the workflow. The time required to transfer a file from one machine to another includes the time to read the file from the source machine's disk, the duration of the network transfer, and the time to write the file to the destination machine's disk.

IV. OPTIMIZING TASK PLANNING TO REDUCE DATA TRANSFERS AND IMPROVE EXECUTION

The goal of this algorithm is to minimize the total execution time (makespan), by accounting for both the parallel execution of tasks and reducing data transfers across the network. In other words, the study does not explicitly consider data transfer time, as the algorithm is designed to minimize network transfers. However, completely avoiding these transfers is not always achievable due to the complexity of task dependencies.

The algorithm begins by constructing a sorted scheduling list that includes all the tasks in the workflow (lines 1-2). The tasks are ordered based on their decreasing bottom level values. The rank of a task t^i, denoted $Rank^i$, is the length of the longest path from t^i to the end of the workflow. This rank includes the estimated duration of all tasks along this path, including t^i. Following the approach in [], we also account for the estimated data transfer costs when computing the rank values of tasks. This prioritization ensures that the most critical tasks are given higher priority, while also respecting the dependencies between tasks. Next, the algorithm assigns an initial mapping for each task t^i in the set T (lines 3-7). The chosen virtual machine vm from the set VM is the one that: (i) minimizes the start time of t^i and (ii) maximizes the volume of input files already stored locally for t^i. The reasoning behind this is that when multiple virtual machines

can start t^i at the same time, the algorithm favors the machine that reduces data transfers over the network, thus improving overall efficiency. Since all the virtual machine instances in consideration are multi-core, scheduling a task t^i on a machine vm requires maintaining a local schedule within the virtual machine. To optimize the use of the available cores, each virtual machine is managed similarly to how a job and resource manager would handle tasks. Keeping track of the number of processors available on each virtual machine is essential for determining the earliest possible start time for a new task on that machine (i.e., st_k^i). Once vm is selected for the execution of t^i, the number of processors available on vm is updated accordingly (line 6).

The workflow is traversed level by level, from the bottom to the top (lines 8-40). The reasoning behind this approach is that during the initial placement, which proceeds from top to bottom, only the data volume from a task's direct predecessors is considered. It is not feasible to factor in the locality of data required by a task's direct descendants since their placements have not yet been determined. As a result, this can lead to unnecessary data movements that could otherwise be avoided.

We define *level 0* as the topmost level of the Directed Acyclic Graph (DAG), which contains all the entry tasks of the workflow. For each subsequent task, its level is recursively computed as the maximum level of its predecessors plus one. Ultimately, we denote the total number of levels in the workflow as L.

We begin by saving the current start time ${}_cst^i$ and the current mapping vm^i for each task t^i in T^l (lines 11-12). Next, we calculate the local volume LV_k^i for task t^i on machine vm_k (lines 13-15). Afterward, we store the local volume for the current mapping of t^i (line 16) before canceling the current mapping (line 17).

Canceling the mapping of all the tasks at a given level creates idle processors of different machines. These processors can be leveraged to enhance data locality by "migrating" certain tasks from one machine to another. The conditions for migrating a task t^i from its previous mapping to a new mapping on vm_k are twofold: it must improve data locality, i.e., $LV_k^i \geq {}_cLV^i$, and reduce the task's start time, i.e., $st_k^i \leq {}_cst^i$ (lines 21-27).

At each step, the algorithm attempts to find a better mapping for each task by prioritizing the machine that offers the largest increase in local data volume. If the task can also start earlier on this machine, it is selected for a new tentative mapping. The first option (lines 28–31) is to execute this task and update the processor count, provided that its ready time matches its calculated start time (line 29), which does not account for data transfer time.

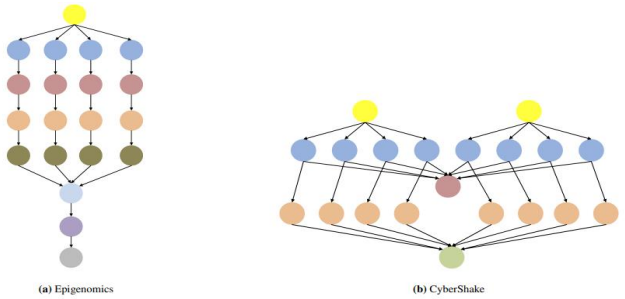
The second option (lines 32–37) applies if the ready time of a task is strictly earlier than its calculated start time (line 32). In this case, if all predecessor tasks t^j, whose calculated finish time coincides with the calculated start time of the ready task t^i, have finished their execution (i.e., they have released at least one processor) (line 33), then t^i is executed, and the count of available processors is updated (lines 34–35). Note that the calculated finish and start times do not factor in data transfer time but rather consider the data transfer volumes.

Algorithm SMDT-ERU

1: Compute $Rank^i$ for each task t^i
2: Sort T by decreasing $Rank^i$ values
3: **for all** $t^i \in T$ **do**
4: $vm \leftarrow \{vm_k \in VM \mid st_k^i$ is minimal **and** LIV_k^i is maximal$\}$
5: Map t^i on vm
6: Update $proc_k$
7: **endfor**
8: **for** l = L to 0 **do**
9: $T^l \leftarrow$ tasks in level l sorted by decreasing $Rank$ values
10: **for all** $t^i \in T^l$ **do**
11: ${}_cst^i \leftarrow$ current start time of t^i
12: $vm^i \leftarrow$ current mapping of t^i
13: **for all** $vm_k \in VM$ **do**
14: $LV_k^i \leftarrow LIV_k^i + LOV_k^i$
15: **endfor**
16: ${}_cLV^i \leftarrow$ current local volume of t^i
17: Cancel the current mapping of t^i
18: **endfor**
19: **for all** $t^i \in T^l$ **do**
20: sort VM by decreasing LV_k^i values
21: **while** $LV_k^i \geq {}_cLV^i$ **do**
22: **if** $st_k^i \leq {}_cst^i$ **then**
23: Map t^i on vm_k
24: Update $proc_k$
25: **break**
26: **endif**
27: **endwhile**
28: **if** $proc_k \geq 0$ **then**
29: **if** $rt_k^i = st_k^i$ **then**
30: Execute t^i on vm_k
31 Update $proc_k$
32: **else if** $rt_k^i < st_k^i$ **then**
33: **if** all tasks $t^j \mid ft_k^j = st_k^j$ are computed **then**
34: Execute t^i on vm_k
35: Update $proc_k$
36: **endif**
37: **endif**
38: **endif**
39: **endfor**
40: **endfor**

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



[24]. These workflows are designed to simulate real-world scientific applications. The key characteristics of these applications are summarized in Table II and their structure depicted in Figure 1. Epigenomics: is a data processing pipeline designed to automate the execution of various genome sequencing operations. Cybershake: is an application developed by the Southern California Earthquake Center aimed at characterizing earthquake hazards. Our simulations were carried out using WRENCH [25], [26], [27] version 2.3 and SimGrid [28] version 3.36.

To generate the various platforms and prevent resource wastage specifically, to avoid leasing resources that will remain unused the number of tasks that can be executed in parallel for each workflow is assessed. This involves oversizing the platform, meaning it is provisioned with as many cores as there are tasks in the workflow. In this study, both workflows contain 1,000 tasks, leading to the consideration of a platform with 1,000 cores. This approach enabled us to identify the total number of cores that could be utilized concurrently for each workflow. For CyberShake, 374 cores are utilized in parallel, while for Epigenomics, 246 cores are employed concurrently. This preliminary analysis will inform the next section, where we will establish the limits of platforms to be used for each workflow in our experiments. This evaluation begins by analyzing the effect of virtual machine size on the execution time of scientific workflows. For each workflow, we examine infrastructures where we incrementally increase the maximum number of cores per VM, ranging from 2 to 96 cores. Thus, for a given total number of cores to be utilized, we create platforms with 2 cores per VM, 4 cores per VM, and so on, up to 96 cores per VM.

As illustrated in Figures 2 and 3, platforms with 2 cores per VM exhibit poor execution times compared to those with 32 cores per VM, and the latter also perform worse than platforms with 96 cores per VM. Increasing the total number of cores per VM results in better execution times. Consequently, this study advocates for the use of larger VMs (i.e., VMs with multiple cores), as they can execute several tasks in parallel, significantly reducing the makespan.

In Figures 2 and 3, the term max_ft represents the theoretical minimum bound, assuming no communication occurs during workflow execution. The execution times obtained on platforms with large VMs (i.e., those with 96 cores) approach this theoretical minimum.

Distinct behaviors are observed among the three workflows considered. First, the total number of cores used has minimal impact on execution time for the CyberShake application, while the Epigenomics workflow shows a plateau in execution time starting from the use of two total cores. Second, although execution time decreases as the size of the virtual machine instances increases, the improvement becomes marginal for sizes beyond 32 cores. Notably, the CyberShake workflow, which generates significantly more intermediate data than the other two

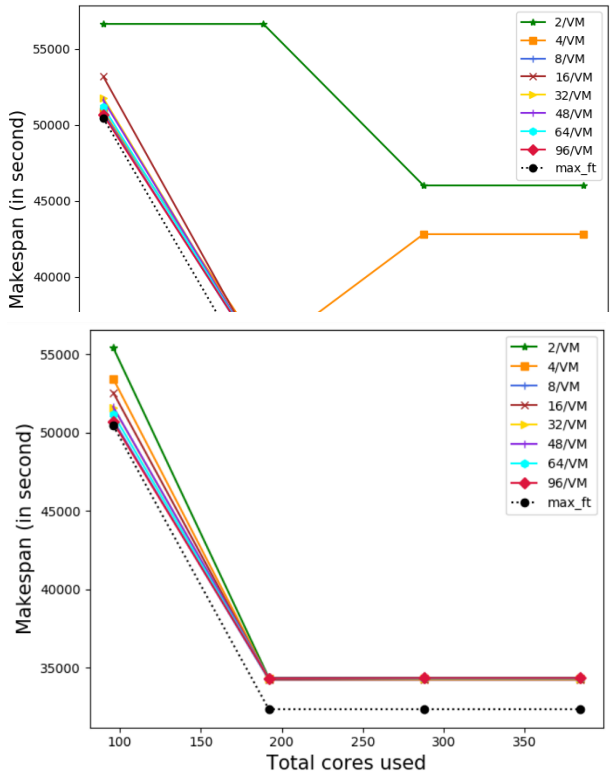


Figure 3: Evolution of the Makespan of the Epigenomics Workflow with Variation of the Total Number of Cores per VM.

workflows, performs worse when relying on the local storage of smaller instances (i.e., those with up to eight cores) compared to a configuration with one core per VM where all intermediate data is stored on the EBS service. This occurs because utilizing too many small VMs on a single host (i.e., up to 48 instances with two cores each) increases the volume

of data transfers between instances and causes network contention. In contrast, in the baseline configuration, each VM benefits from a dedicated network connection to the shared storage service.The main objective of this study is to minimize the execution time of an application. To achieve this goal, the main contribution of this paper is the reduction of files transferred during the execution of the application, which has a considerable impact on the execution time. To evaluate the performance of this algorithm, the proposed approach is compared to HEFT [29] and Max-Min [30], two very popular heuristics for parallel application scheduling. HEFT andMax-Min have been adapted to IaaS cloud resources and the simulation environment. The simulation results show that the proposed approach offers good results compared to Max-Min and often similar results to HEFT.

In the results of Figures 4 and 5, we use platforms where the total number of cores used ranges from 2 to 374 cores for CyberShake, and from 2 to 246 cores for Epigenomics, in increments of 2.

Large VMs are preferred for each platform as they allow multiple tasks to be executed in parallel. Juve et al. [31] demonstrated that each task in a real scientific workflow is a single-core activity, meaning it can only run on a single computing core, rather than utilizing all the cores of a VM. This study follows the same principle. Therefore, a user wishing to run a parallel application in the cloud needs to rent a total number of cores, and the proposed approach provides a platform that minimizes the application's completion time. For example, if a user wants to utilize 100 cores in total, the platform will consist of one VM with 96 cores and another with 4 cores. For a total of 200 cores, the platform will include three VMs: two with 96 cores and one with 8 cores, and so on.

When a platform contains more virtual machines (VMs), more files will need to be transferred across the network. While large VMs are prioritized for application execution, the key reason behind this choice is to maximize bandwidth usage for these machines, as summarized in Table I. With the HEFT and Max-Min algorithms, an increase in the number of VMs on the platform leads to a higher volume of file transfers over the network. However, the proposed approach reduces file transfers at two levels for each task: reducing transfers from parent tasks and reducing transfers to child tasks. This strategy allows us to achieve better results compared to HEFT and Max-Min, as shown in Figures 4 and 5.

These experiments demonstrate that our approach yields better results, as summarized by the following improvements. The CyberShake workflow achieves a 74.06% reduction in makespan compared to HEFT and 90.37% reduction in makespan compared to Max-Min (c.f. Figure 4). For the Epigenomics workflow, our approach achieves a 2.85% improvement in makespan compared to the HEFT algorithm, and 12.99% compared to the Max-Min algorithm (c.f. Figure 5). This relatively modest gain can be explained by the limited amount of data processed during the execution of the

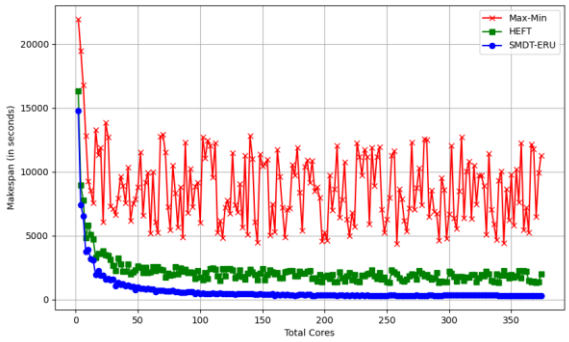


Figure 4: Evaluation Results for the CyberShake Workflow.

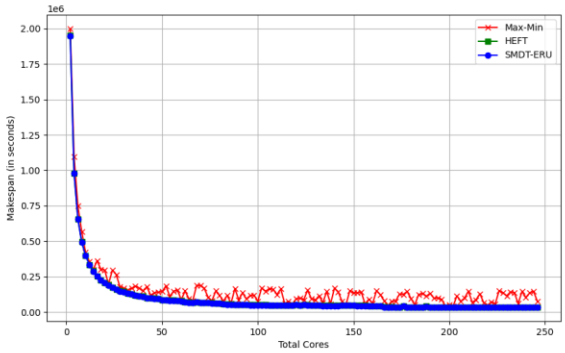


Figure 5: Evaluation Results for the Epigenomics Workflow.

workflow, amounting to 13.21 GB (c.f. Table II), calculated as the difference between the total number of files and the input files. In this context, the makespan is not significantly impacted by data transfer since the data volumes are small, which limits the effect of our optimization on this particular aspect.

However, for the CyberShake workflow, the more significant improvement in makespan is due to the notable reduction in data transferred over the network. During the execution of the CyberShake workflow, 249.63 GB of data are processed (c.f. Table II). This large volume can cause a noticeable degradation in the total execution time, primarily due to data transfers between tasks. These transfers act as a bottleneck, slowing down the workflow's execution.

Our approach stands out by employing a rigorous task scheduling strategy and local storage for preserving the output data of each task. This minimizes data transfers across the network, leading to a substantial reduction in makespan, especially in scenarios where handling large volumes of data is a key factor.

VI. CONCLUSION AND FUTURE WORK

Infrastructure as a Service (IaaS) clouds now enable scientists to execute their data-intensive workflows on infrastructures tailored to the computing and storage requirements of these applications. Determining the optimal set of virtual machine instances to form these infrastructures is a complex task, often handled by Workflow Management Systems (WMS). A critical factor in achieving high performance is the ability to effectively leverage the specific characteristics of virtual machine instances.

In this paper, we first demonstrated the benefits of using multi-core machines, as increasing the number of cores allows multiple tasks to be executed on the same machine, with the bandwidth connecting the machine to the switch being proportional to the number of cores. We then proposed scheduling algorithms that minimize the makespan by reducing data transfers between dependent tasks over the network. Finally, the experimental results showed that our proposed approach yields better results than both HEFT and Max-Min, two of the most effective list-scheduling algorithms.

As part of our future work, we plan to compare the simulated executions with actual runs on the AWS computing cloud using M5d instances to validate the impact of the proposed algorithms. Additionally, we aim to explore the multi-objective aspect of the scheduling problem, allowing users to prioritize either shorter execution times or lower costs. We will propose a complementary approach where one of the objectives is fixed—either a predefined budget or a set deadline.

REFERENCES

1. X. Meng et L. Golab, « Parallel Scheduling of Data-Intensive Tasks », in *Euro-Par 2020: Parallel Processing*, M. Malawski et K. Rzadca, Éd., Cham: Springer International Publishing, 2020, p. 117-133. doi: 10.1007/978-3-030-57675-2_8.
2. D. C. M. de Oliveira, J. Liu, et E. Pacitti, *Data-Intensive Workflow Management*. Springer Nature, 2022.
3. S. Pellegrini, F. Giacomini, et A. Ghiselli, « A Practical Approach for a Workflow Management System », in *Grid Middleware and Services: Challenges and Solutions*, D. Talia, R. Yahyapour, et W. Ziegler, Éd., Boston, MA: Springer US, 2008, p. 279-287. doi: 10.1007/978-0-387-78446-5_18.
4. « Calcul — Types d'instances Amazon EC2 — AWS », Amazon Web Services, Inc. Consulté le: 16 octobre 2024. [En ligne]. Disponible sur: https://aws.amazon.com/fr/ec2/instance-types/
5. M. S. R. Krishna et S. Mangalampalli, « A Systematic Review on Various Task Scheduling Algorithms in Cloud Computing », *EAI Endorsed Trans. Internet Things*, vol. 10, 2024, doi: 10.4108/eetiot.4548.
6. L. Yang, Y. Xia, X. Zhang, L. Ye, et Y. Zhan, « Classification-Based Diverse Workflows Scheduling in Clouds », *IEEE Trans. Autom. Sci. Eng.*, vol. 21, nº 1, p. 630-641, janv. 2024, doi: 10.1109/TASE.2022.3217666.
7. Z. Zhu et X. Tang, « Deadline-constrained workflow scheduling in IaaS clouds with multi-resource packing », *Future Gener. Comput. Syst.*, vol. 101, p. 880-893, déc. 2019, doi: 10.1016/j.future.2019.07.043.
8. D. Alsadie et M. Alsulami, « Enhancing Workflow Efficiency: A Modified Firefly Algorithm for Hybrid Cloud-Edge Environments », 8 août 2024, *Research Square*. doi: 10.21203/rs.3.rs-4623299/v1.
9. S. Murad *et al.*, « OPTIMIZED MIN-MIN TASK SCHEDULING ALGORITHM FOR SCIENTIFIC WORKFLOWS IN A CLOUD ENVIRONMENT », *J. Theor. Appl. Inf. Technol.*, vol. 100, p. 480-506, janv. 2022.
10. A. Patil et B. Thankachan, « Review on a comparative study of various task scheduling algorithm in cloud computing environment », *Turk. J. Comput. Math. Educ. TURCOMAT*, vol. 11, nº 3, p. 1396-1401, 2020.
11. O. Sukhoroslov, « Scheduling of Workflows with Task Resource Requirements in Cluster Environments », in *Parallel Computing Technologies*, V. Malyshkin, Éd., Cham: Springer Nature Switzerland, 2023, p. 177-196. doi: 10.1007/978-3-031-41673-6_14.
12. R. Akraminejad, N. Khaledian, A. Nazari, et M. Voelp, « A multi-objective crow search algorithm for optimizing makespan and costs in scientific cloud workflows (CSAMOMC) », *Computing*, vol. 106, nº 6, p. 1777-1793, juin 2024, doi: 10.1007/s00607-024-01263-4.
13. S. Bansal et H. Aggarwal, « A multiobjective optimization of task workflow scheduling using hybridization of PSO and WOA algorithms in cloud-fog computing », *Clust. Comput.*, vol. 27, nº 8, p. 10921-10952, nov. 2024, doi: 10.1007/s10586-024-04522-3.
14. M. Raeisi-Varzaneh, O. Dakkak, Y. Fazea, et M. G. Kaosar, « Advanced cost-aware Max–Min workflow tasks allocation and scheduling in cloud computing systems », *Clust. Comput.*, vol. 27, nº 9, p. 13407-13419, déc. 2024, doi: 10.1007/s10586-024-04594-1.
15. Z. Sun, B. Zhang, C. Gu, R. Xie, B. Qian, et H. Huang, « ET2FA: A Hybrid Heuristic Algorithm for Deadline-Constrained Workflow Scheduling in Cloud », *IEEE Trans. Serv. Comput.*, vol. 16, nº 3, p. 1807-1821, mai 2023,

doi: 10.1109/TSC.2022.3196620.

16. S. Mangalampalli, G. R. Karri, M. Kumar, O. I. Khalaf, C. A. T. Romero, et G. A. Sahib, « DRLBTSA: Deep reinforcement learning based task-scheduling algorithm in cloud computing », *Multimed. Tools Appl.*, vol. 83, nº 3, p. 8359-8387, janv. 2024, doi: 10.1007/s11042-023-16008-2.
17. S. Kumar et S. Chander, « Comparative Analysis Of Task Scheduling Algorithms In Cloud Environment In Terms of Their Future Prospective And Risk », *Webology*, vol. 18, nº 5, 2021, Consulté le: 15 octobre 2024. [En ligne]. Disponible sur: https://www.webology.org/data-cms/articles/20220212054937pmwebology%2018%20(5)%20-%2059%20pdf.pdf
18. P. Sadotra, P. Chouksey, M. Chopra, R. Koser, et R. Rawat, « Research Review on Task Scheduling Algorithm for Green Cloud Computing », in *Scalable Modeling and Efficient Management of IoT Applications*, IGI Global, 2025, p. 137-152. doi: 10.4018/979-8-3693-1686-3.ch007.
19. A. Taghinezhad-Niar, S. Pashazadeh, et J. Taheri, « QoS-aware online scheduling of multiple workflows under task execution time uncertainty in clouds », *Clust. Comput.*, vol. 25, nº 6, p. 3767-3784, déc. 2022, doi: 10.1007/s10586-022-03600-8.
20. Z. Sun, C. Gu, H. Huang, et H. Zhang, « T2FA: A Heuristic Algorithm for Deadline-Constrained Workflow Scheduling in Cloud with Multicore Resource », in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, sept. 2021, p. 345-354. doi: 10.1109/CLOUD53861.2021.00048.
21. S. M. F. D. S. Mustapha et P. Gupta, « DBSCAN inspired task scheduling algorithm for cloud infrastructure », *Internet Things Cyber-Phys. Syst.*, vol. 4, p. 32-39, janv. 2024, doi: 10.1016/j.iotcps.2023.07.001.
22. R. Sandhu, M. Faiz, H. Kaur, A. Srivastava, et V. Narayan, « Enhancement in performance of cloud computing task scheduling using optimization strategies », *Clust. Comput.*, vol. 27, nº 5, p. 6265-6288, août 2024, doi: 10.1007/s10586-023-04254-w.
23. H. Mikram, S. El Kafhali, et Y. Saadi, « HEPGA: A new effective hybrid algorithm for scientific workflow scheduling in cloud computing environment », *Simul. Model. Pract. Theory*, vol. 130, p. 102864, janv. 2024, doi: 10.1016/j.simpat.2023.102864.
24. « https://pegasus.isi.edu/workflow_gallery/ », Pegasus WMS. Consulté le: 16 octobre 2024. [En ligne]. Disponible sur: https://pegasus.isi.edu/workflow_gallery/
25. H. Casanova *et al.*, « Developing accurate and scalable simulators of production workflow management systems with WRENCH », *Future Gener. Comput. Syst.*, vol. 112, p. 162-175, nov. 2020, doi: 10.1016/j.future.2020.05.030.
26. H. Casanova, R. Tanaka, W. Koch, et R. Ferreira Da Silva, « Teaching parallel and distributed computing concepts in simulation with WRENCH », *J. Parallel Distrib. Comput.*, vol. 156, p. 53-63, oct. 2021, doi: 10.1016/j.jpdc.2021.05.009.
27. H. Casanova, S. Pandey, J. Oeth, R. Tanaka, F. Suter, et R. Ferreira Da Silva, « WRENCH: A Framework for Simulating Workflow Management Systems », in *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, Dallas, TX, USA: IEEE, nov. 2018, p. 74-85. doi: 10.1109/WORKS.2018.00013.
28. « SimGrid Home ». Consulté le: 16 octobre 2024. [En ligne]. Disponible sur: https://simgrid.org/
29. H. Topcuoglu, S. Hariri, et Min-You Wu, « Performance-effective and low-complexity task scheduling for heterogeneous computing », *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, nº 3, p. 260-274, mars 2002, doi: 10.1109/71.993206.
30. « Advanced cost-aware Max–Min workflow tasks allocation and scheduling in cloud computing systems | Cluster Computing ». Consulté le: 18 octobre 2024. [En ligne]. Disponible sur: https://link.springer.com/article/10.1007/s10586-024-04594-1
31. G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, et K. Vahi, « Characterizing and profiling scientific workflows », *Future Gener. Comput. Syst.*, vol. 29, nº 3, p. 682-692, mars 2013, doi: 10.1016/j.future.2012.08.015.