Received ; | Revised ; | Accepted
DOI: xxx/xxxx

Dynamic Data Replication and Placement Strategy in Geographically Distributed data centers

Laila Bouhouch[1] | Mostapha Zbakh[1] | Claude Tadonki[2]

[1]National School of Computer Science and Systems Analysis, Mohammed V University in Rabat, Morocco.
[2]MINES ParisTech-PSL / CRI, Paris, France.

Abstract

With the evolution of geographically distributed data centers in the Cloud Computing landscape along with the amount of data being processed in these data centers, which is growing at an exponential rate, processing massive data applications become an important topic. Since a given task may require many datasets for its execution and the datasets are spread over several different data centers, finding an efficient way to manage the datasets storage across nodes of a Cloud system is a difficult problem. In fact, the execution time of a task might be influenced by the cost of data transfers, which mainly depends on two criterias. The first one is the initial placement of the input datasets during the build-time phase, while the second is the replication of the datasets during the runtime phase. The replication is explicitly consider when datasets are being migrated over the data centers in order to make them locally available wherever needed. Data placement and data replication are important challenges in Cloud Computing. Nevertheless, many studies focus on data placement or data replication exclusively. In this paper, a combination of a data placement strategy followed by a dynamic data replication management strategy is proposed, with the purpose of reducing the associated cost of all data transfers between the (distant) data centers. Our proposed data placement approach considers the main characteristics of a data center such as *storage capacity* and *read/write speeds* to efficiently store the datasets, while our dynamic data replication management approach considers three parameters: the *number of replicas* in the system, the *dependency between datasets* and tasks and the *storage capacity* of data centers. The decision of when and whether to keep or to delete replicas is determined by the fulfillment of those three parameters. Our approach estimates the total execution time of the tasks as well as the monetary cost, considering the data transfers activity. Our experiments are conducted using Cloudsim simulator. The obtained results show that our proposed strategies produce an efficient data management by reducing the overheads of the data transfers, compared to both a data placement without replication (by 76%) and the selected data replication approach from Kouidri et al. (by 52%), and by improving the financial cost.

KEYWORDS:
Big Data, Cloud Computing, Data Placement, Dynamic Data Replication, Cloudsim.

1 | INTRODUCTION

With the expansion of Internet technologies and virtualization, Cloud Computing has established a new and exciting computing paradigm for common applications in various areas such astronomy[5], physics[6] and bioinformatics[7], to name a few. It provides several services[8] like high performance computing resources, distributed computing, mass storage devices and lower cost of infrastructure (pay-as-you go model), and can scale as desired by considering a geo-distributed data center configuration.

Nowadays, various applications produce terabytes or even petabytes of data[9] that need to be efficiently analyzed, stored and processed. In addition, some data need to be exchanged and shared between different actors of the data market. Hence, scientists and industrials have adopted Cloud Computing environment as a good solution to deploy large-scale data applications and thereby simplify storage, partitioning, distribution and assignment of massive data over distant heterogeneous nodes of a given cluster[10]. Moreover, the principle of geographically distributed data centers brought a great value for these Cloud-based platforms by improving the *access times*, *availability* and *scalability* of the stored data, where the heterogeneity refers to main characteristics of a data center such as *hardware type*, *storage capacity* and *processing speed*.

There are many challenging problems for deploying big data applications in a Cloud environment[11]. Sometimes it is hard to get enough space available to store a large amount of data, and it might quiet expensive to move datasets between data centers when the tasks of of a given workflow need them (i.e. when the source data center is different from the target one where the requesting task is being executed).

In addition, the resources are provided in a cost-effective manner[3]. This trend toward economy-based systems raises new challenges for interactions between Cloud providers and users. As the user might need to adjust its current configuration, Cloud resources are provided in an elastical way and the monetary cost is determined on the pay-as-you-go basis[4]. Like any economic enterprise, Cloud service providers seek maximal profit. For this purpose, an efficient data management is essential as it contributes to obtain better performances.

However, in large-scale applications executed across different and distant data centers, a given task might require several input datasets, but there is guarantee that both the task and the required datasets are in the same data center. If the required datasets are not locally available on the same node than that of the consumer task, then they have to be transferred via the network. Thus, migrating these (large-sized) datasets obviously impacts the global execution time of the job. An inefficient data management can lead a severe penalty on the execution time as well as on the monetary cost. In the literature, many techniques are available to boost the performance of Cloud environment[16,17,18] such as *initial data placement*[19] and *data replication*[20] strategies. In fact, data placement strategies aim to reduce the total volume of data transfers. The idea is to store the datasets in the appropriate data centers before starting the workflow execution. Afterwards, *data replication* strategies, which might be dynamic, also aim to to decrease the overall cost of data transfers through the use of replicas at runtime (after using a requested dataset, it might be kept as a copy to serve further requests in a most efficient way through the election of the best source). These techniques are regularly improved by the researchers at various levels. However, most of studies focus on the placement and the replication separately[30].

In our work, we improve and combine the two techniques of initial placement of the datasets over data centers and efficient management of data replication during the workflow. The main goal is to reduce the time and cost cost of data migrations. Our two-phase strategy starts with a static placement, considered before starting the execution (build-time phase) and is based on our previous work[2]. It carefully places the datasets over the distributed data centers considering the heterogeneity of their characteristics (read/write speeds, network bandwidth). We start by computing the matrix of the transfer costs, that gives the global transfer time of every dataset from all possible locations to all possible destinations (the destination is where the dataset is requested by a given task for its execution). Then, using that costs matrix, a greedy algorithm is applied to optimally decide about the final location of the datasets. In other words, instead of reducing the number of data movements among data centers, which does not ensure reducing the time consumption, we directly focus on reducing the data transfers time cost through an appropriate formulation. Henceforth, all the required datasets for a given task will be efficiently transferred to the corresponding data center.

The second phase of our strategy is *replication*, which is performed itertaively at runtime and aims to manage (keep or delete) multiple copies of the datasets. We are aware about the fact that a replication is automatically handled during the build-time phase by the Cloud system itself, but we do not consider this explicitly in our initial placement because we need to exploit the efficiency of our proposed dynamic replication strategy. Our proposed replication strategy takes advantage of the data movement that occur during tasks execution. In other words, when a gievn dataset are transferred from one location to another, a data replication might be considered. In order be able to create a new replica when there aren't enough space, we decide which copy to

keep or delete following a specific procedure. Three parameters are considered our choice: (1) the minimum number of replicas that should exist in the entire system to ensure availability, (2) the dependency between datasets and future tasks, and (3) the storage capacity of the data centers to avoid saturation . Our main objective is to further minimize the total task execution time and financial cost through an efficient management of the datasets as previously explained.

We evaluated the performance of our strategies through simulations using Cloudsim tool. We illustrate that our strategy not only reduce the time consumption of data migrations but also decrease the monetary cost.

The rest of the paper is organized as follows: Section 2 presents a summary of the related works on data placement and data replication strategies. In section 3, we present and describe our model. Section 4 explains in details our static data placement algorithm followed by our dynamic data replication algorithm. Section 5 present a case study and discusses the experimental results obtained using an extension of Cloudsim toolkit. Finally, we draw our conclusions and future work in section 6.

2 | RELATED WORK

We present the related literature and give a summary of some existing works on data placement (static) and on data replication (dynamic). Finding an optimal data placement is considered as a critical issue in geographically distributed data center systems, mainly because of the important cost of each transfer. A reasonable data placement strategy definitely reduces the time overhead of data migration. Several disseminated studies propose different techniques related to that concern.

Authors of[21] propose a K-means algorithm to solve the data placement problem. They distribute the data over the data centers based on a dependency matrix that they first compute, and they chose to store strongly interdependent datasets into the same data center. In[22], the authors consider a data placement algorithm based on the genetic algorithm paradigm, where a better solution is produced at each new generation. They get a better approximated-optimal solution for placing and scheduling datasets compared to a Monte Carlo algorithm and the exhaustive search algorithm. The work in[23] contains two contributions: (1) a genetic algorithm based on a heuristic approach, (2) a load balancing procedure over the data centers taking into account the constraints related to the workload and the storage capacity. The combination of the two strategies reduces significantly the amount of data transfers. Another contribution is described in[24], where the datasets are clustered based on the correlation between them. The size of the datasets is considered as the main factor for the correlation that is used for the clustering procedure. In[25], the proposed data placement strategy aims to decrease the total amount of data migrations between virtual machines based on a genetic algorithm. The most interdependent datasets are placed on the same virtual machine. The data transfer rate between virtual machines is also considered.

However, as far as we know, most of the afromentioned works mainly focus on reducing the number of data migrations rather than the time consumption of data transfers, which is the current interest. In fact, while processing a data intensive application in a geographically distributed data centers, multiple datasets are migrated from one data center to another distant location. Considering the heterogeneity of the data centers in terms of storage capacities, processing and read/write speeds and the bandwidth of the connections, the total cost of data transfers will definitely affect the overall efficiency. So, only reducing number of data movements over distributed data centers is not equivalent to decreasing the data migrations time cost. For this reason, in this work, we propose a data placement strategy in order to minimize time to transfer datasets when running large-scale applications in a Cloud Computing environment and consequently improve the global performance.

Beside data placement technique in managing big data applications, there is also the data replication technique which is adopted to create multiple copies of datasets and geographically disperse them. The main goal of replicating data is to improve data availability by saving bandwidth consumption which is a considerable obstacle that impacts data access, reduce the data transfers amounts/costs and minimize execution time when tasks are processed in the Cloud. Given this, data replication management is considered one of the hot spot researches in large-scale distributed systems. The data replication strategies can be categorized into two types: static replication[12,13,14,15] and dynamic replication[26,27,28,29,30,31]. Static data replication strategies create and manage replicas manually and are incapable to be adjusted with the various changes especially in a large-scale Cloud systems. While the dynamic data replication strategies are able to automatically generate and delete replicas as things keep going. Hence, studies defined new dynamic techniques to deal with this major issue.

Authors in work[26] suggest a data replication strategy to ensure a good response time of tenant's query while satisfying the Cloud provider in term of economic profitability. The replication process is triggered only if it doesn't effect the expenditures of the provider and if the estimated response time of the query is greater than the response time threshold. In[27], authors propose a two-step big data replicas placement strategy where the first step aims to group tasks using the same replicas. While the second

step aims to group datasets that are used together by the same tasks. The main goal of this work is to reduce the data movement among data centers when executing tasks and progress task parallel execution. Authors of[28] suggested a replica placement algorithm to minimize the response time of the workflow while reducing the number of replicas. The replica placement strategy established some parameters such as the size of replica, the number of replicas accesses and when the replica was requested lately, in order to rationally decide about which datasets to replicate and which one no to due of the storage limitations. Results of this work show the efficiency of the proposed algorithm regarding the response time, the network usage, the replication frequency and also the storage usage. The data replication strategy proposed in[29] aims to minimize the cost of data communication and data storage. Their approach is based on three parameters: datasets dependencies, access latency for each dataset and data centers capacities. The replication process is done in the pre-built phase, after the initial data placement is accomplished. In[30], authors combined job scheduling and dynamic replication strategies into one framework to better minimize the total response time and data movement among virtual machines. The most frequently used datasets are replicated and stored in the machine that contains enough storage space and doesn't already store the replicate dataset. This work shows better results when strategies are applied together. The study in[31] proposed an adaptive geo-replication strategy inspired by Ant Colony algorithm. Their algorithm dynamically determines the number of replicas, the location of replicas and when replicas are deleted from a site based on predefined thresholds. Read and write operations on data centers were taken into consideration to implement the strategy and results provided an optimal solution in terms of availability/accessibility, scalability and latency.

To the best of our knowledge, the afromentioned contributions on data replication approaches do not take into account the replicated data resulting from data movement during task execution, which we address in our paper. In fact, we proposed a replica management strategy to take advantage of these replicated datasets and that occur as a result of dataset transfers, then reuse them for the execution of other tasks. In this manner, we will have more datasets available on the data centers and less data movement. Considering the number of existing replicas, the dependency between datasets and tasks that are not yet executed, and the storage capacity, during the execution of a task, some datasets will be kept and others will be removed from a specific data center to ensure the execution of the task. In our study, we also address the economic aspects of the replication process in Cloud systems as well as the concept of geographically distributed data centers which are not much studied in many works. Moreover, the objective of our study is to combine both strategies, the quantities and costs of data migration will be progressively reduced due to the availability of datasets across data centers; therefore, the overall task execution time and the total monetary cost of replication will be further improved.

3 | SYSTEM MODEL

In order to reduce the time cost of data transfers among data centers in a Cloud Computing environment, we seek a good datasets management through an optimal placement at the build-time phase and a skillful replication at runtime. We first list the basic elements of our model.

3.1 | Symbols and notations

Table 1 depicts the essential symbols and notations.

3.2 | Description of the model

We now define the model as follows:

1. **Data centers graph:** We consider a system of geographically distributed and interconnected data centers. We model the system with a graph $G = (M, w, r, B)$. $M = \{m_i, 1 \leq i \leq p\}$ is the set of data centers. Each data center m_i is characterised by its storage capacity c_i, its read speed r_i and its write speed w_i. Next, we consider matrix $B = (b_{ij})_{1 \leq i,j \leq p}$ to model the

TABLE 1 Parameters and variables of our model

Inputs		
$M = (m_i)$		m_i designates the i^{th} data center
$C = (c_i)$		c_i is the storage capacity of data center m_i
$R = (r_i)$		r_i is the read speed of data center m_i
$W = (w_i)$		w_i is the write speed of data center m_i
$B = (b_{ij})$		b_{ij} is the bandwidth of the connection between data centers m_i and m_j
$\Psi = (\psi_{ij})$		ψ_{ij} is the elementary data transfer time between data centers m_i and m_j
$T = (t_i)$		t_i designates the i^{th} task
$\alpha = (\alpha_i)$		α_i is the index of the data center where task t_i is assigned to (i.e. m_{α_i})
$D = (d_i)$		d_i designates the i^{th} dataset
$V = (v_i)$		v_i is the volume of dataset d_i
$F = (f_{ij})$		$f_{ij} = 1$ if d_i is required by t_j, and 0 otherwise (F is the datasets to tasks assigments matrix)
$\tau = (\tau_{ik})$		τ_{ik} is total cost of all necessary transfers of d_i from data center m_k
$E = (e_{ij})$		$e_{ij} = 1$ if there is a copy of d_i stored in data center m_j, and 0 otherwise (E is the replication matrix)
$Q = (q_i)$		q_i is the total monetary cost of the data storage associated to task t_i (per unit of volume)
$P = (p_i)$		p_i is the total monetary cost of all data transfers associated to task t_i (per unit of volume)
Objective function		
WCC		Total workflow data transfers cost
Outputs		
$\Phi = (\phi_i)$		ϕ_i is the index of the data center where dataset d_i is (indicated to be) placed (i.e. m_{ϕ_i})
$S = (s_i^{(j)})$		$s_i^{(j)}$ is the index of the chosen source data center for d_i required by t_j (i.e. $m_{s_i^{(j)}}$)

interconnection between the data centers, where $b_{ij}, 1 \leq i, j \leq p$ is the bandwidth of the connection between data centers m_i and m_j. We define matrix $\Psi = (\psi_{ij})$ as follows:

$$\psi_{ij} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } m_i \text{ and } m_j \text{ aren't linked} \\ \frac{1}{b_{ij}} & \text{otherwise} \end{cases} \quad (1)$$

Considering matrix Ψ is because we are interested in the transfer time, which is proportional to the volume of the dataset and $\frac{1}{b_{ij}}$ (it is the product of both). Thus, the cost of transfering a dataset of volume v from m_i to m_j is given by:

$$t(v, i, j) = v \times \psi_{ij}. \quad (2)$$

2. **Definition of the tasks:** We assume a fixed number of tasks to be sumitted, each of which provided with its workload (not considered here) and the required datasets for its execution. As indicated in Table 1 , we denote the group of task as $T = \{t_1, t_2, \dots, t_n\}$, where n is the number of tasks. An allocation of the tasks to the data center is provided (as input) through vector $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$, i.e. t_i is executed on data center m_{α_i}. The tasks can be executed in parallel, but each one is processed in only one data center (no task migration and no preemption).

3. **Datasets to tasks assignment:** The set of datasets is $D = \{d_1, d_2, \dots, d_m\}$, and dataset d_i has volume v_i. A task might require one or several datasets and a single dataset may be consumed by several different tasks. This led us to a $m \times n$

matrix F to model the assignment of the datasets and the tasks:

$$f_{ij} = \begin{cases} 1 & \text{if } d_i \text{ is required by } t_j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

4. **Constraints**: The only constraint that is considered in this paper is related to the data center storage capacity. The total volume of the datasets stored in a given data center should be lower than its capacity.

5. **Data placement solution:** This the (initial) datasets to data centers assignment that is provided as the result of a given strategy. We use vector $\Phi \in N^{|D|}$ where ϕ_i is the index of the data center that should house d_i, $i \in 1, \dots, m$. Thus

$$d_i \text{ is stored in data center } m_{\phi_i} \tag{4}$$

Our data placement algorithm is executed first to provide a datasets to data centers assignment (i.e. before the execution of the tasks) and the objective is to minimize the cost of data migrations (i.e. moving datasets from one data center to another) while considering the storage capacity constraint.

6. **Data replication**: We consider that a given dataset might be replicated in order to create several source alternatives. We consider a $m \times p$ matrix E to indicate the data centers that hold a copy (replica) of a dataset:

$$e_{ij} = \begin{cases} 1 & \text{if } d_i \text{ has a replica in } m_j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Our data replication strategy aims to complete the data placement solution by taking profit of the datasets movements during the runtime. The idea is to create copies on-the-fly of the datasets so as to be able to chose the best source for any given dataset required, by a task, at a given data center. Next section explains more clearly both strategies and illustrates the benefit of combining the two approaches.

3.3 | Problem formulation

Considering a workflow W and a Cloud system C, the workflow communication cost WCC is the total data transfer cost for executing all the tasks of the workflow W in the Cloud C. WCC can be estimated using formulas 6 and 7.

$$WCC(W, C) = \sum_{j=1}^{|T|} \sum_{i=1}^{|D|} f_{ij} \times \tau^{(i)}(s_i^{(j)}, \alpha_j) \tag{6}$$

where

$$\tau^{(i)}(s_i^{(j)}, \alpha_j) = \begin{cases} (\frac{1}{r_{s_i^{(j)}}} + \frac{1}{w_{\alpha_j}} + \frac{1}{b_{s_i^{(j)}\alpha_j}}) \times v_i & \text{if } s_i^{(j)} \neq \alpha_j \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The transfer cost of task t_j, $1 \leq j \leq |T|$, is computed by considering all involved datasets (i.e. datasets i, $1 \leq i \leq |D|$, such that $f_{ij} \neq 0$). Equation 7 sums up this cost for all the tasks. Regarding Equation 7, two cases should be considered:

- **If** dataset d_i is not locally avalaible, thus it should be migrated from its source data center $m_{s_i^{(j)}}$ to the target data center m_{α_j} (where task t_j is assigned to). The transfer cost is calculated considering the *read/transfer/write* sequence as follows: (1) d_i is read from $m_{s_i^{(j)}}$ for a duration given by $\frac{v_i}{r_{s_i^{(j)}}}$; (2) d_i is migrated (through the network) at a cost given by $\frac{v_i}{b_{s_i^{(j)}\alpha_j}}$; (3) finally d_i is written in m_{α_j} (to be locally avalaible for t_j) at a cost given by $\frac{v_i}{w_{\alpha_j}}$. The main constraint related to storage is that the aggregated volume of the datasets stored within a given data center should not exceeds its capacity. In contrast, when data replication occurs, the storage capacity constraint is handled by selecting the right dataset(s) to be delete as we will explain in Section 4.2 through our proposed strategy.

- **Otherwise,** since d_i is stored in the same data center which runs task t_j, no transfer is needed.

The mathematical programming formulation of our problem could be written as follows:

$$\begin{aligned} &\min_{s} \sum_{j=1}^{|T|} \sum_{i=1}^{|D|} f_{ij} \times \tau^{(i)}(s_i^{(j)}, \alpha_j) \\ &\text{subject to} \\ &\sum_{j=1}^{|D|} v_j \delta_{is_j^{(i)}} \leq c_i, i = 1, \cdots, |M| \end{aligned} \tag{8}$$

where δ is the Kronecker symbol (i.e. $\delta_{ij} = 1$ if $i = j$ and 0 otherwise). This is what we intend to solve by a heuristic approach.

3.4 | Financial cost model

Unlike traditional models or federated systems such as a data grid, the Cloud systems is essentially based on the pay-as-you-go model[26]. To really benefit from this flexibility, a cost-efficient management by the provider is necessary, as well as by the user.

3.4.1 | Provider's revenue

A provider's income is essentially the amount received from the Cloud users[32]. The user typically pays a fee that is associated with the delivered services. The exact amount to be paid is determined based on defined service parameters during a billing period. Both the provider and the user agree on the suitable services for a given working session. In our data replication strategy, we assume that the service level agreement is satisfied. The user is not charged for the number of maintained replicas. It is the responsibility of the provider to store the replicas in the Cloud. The user only pays for the used services and does not care about how many replicas the provider create.

3.4.2 | Provider's expenses

The Cloud service provider has a certain number of expenses[33]. For every task execution, three types of cost are resulting when required datasets are migrated from remote data centers:

- $P = (p_i)$: is the cost of the network usage related to task t_i for the involved datasets migrations and replications.
- $Q = (q_i)$: is the cost of the storage space for the replicas created when executing t_i.
- $Penalty = (penalty_i)$: is the penalty paid by the provider to the user when there is a violation of the response time delay. The data replication strategy aims to contribute in reducing this penalty by providing several potential sources for a specific data request so as to allow for a better choice.

Our goal is to propose a replication strategy that carefully balances the monetary cost and performance improvment. With respect to the types of cost previously described, we estimate an economic cost, denoted by *FinancialCost*, using equation 9:

$$FinancialCost = Q + P + Penalty \tag{9}$$

4 | PROPOSED APPROACH

In this section, we explain the main stages of our suggested approach that integrates two techniques in order to reduce the communication cost while migrating datasets among data centers. The first technique is a data placement strategy that is executed in the build-time phase and the second technique is a data replication strategy that is executed during the runtime phase.

4.1 | Data placement strategy

Our proposed data placement algorithm is executed in the build-time before starting the execution of the tasks. It seeks an optimal initial placement for the datasets. The procedure has mainly two steps presented in Algorithms 1 & 2 and explained in next sections 4.1.1 & 4.1.2. In the first step, the total transfer time of the datasets between the data centers is computed based on

the volume of the datasets, the read/write speeds of the disks and the network bandwidth. In the second step, a greedy algorithm is applied based on the previously calculated transfer time to assign each dataset to the optimal data center. Our algorithm is a heuristic, thus it should provide a good solution (not necessarily optimal) in an affordable time overhead. Even if we are aware of the internal replication made by the Cloud system, we do not consider it explicitly in our placement algorithm.

As we know, a given dataset may be either stored in the same data center where its consumer(s) task(s) is/are executed, in that case no migration is needed, or stored in a distant data center and thus an explicit transfer is needed.

If dataset d_i is stored in data center m_j and is required by a task assigned to data center m_k ($j \neq k$). The time to transfer d_i from its source m_j to the target m_k, $1 \leq j, k \leq p$ and $1 \leq i \leq m$, is given by equation 10:

$$\tau_{jk}^{(i)} = v_i \times (\frac{1}{r_j} + \frac{1}{w_k} + \frac{1}{b_{jk}}) \tag{10}$$

Since dataset d_i could be used by many tasks, its total migration time from m_j, taking into account all the tasks during the whole execution, is given by:

$$\tau_{ij} = \sum_{k=1}^{|T|} f_{ik} \times \tau_{j\alpha_k}^{(i)} = \sum_{k=1}^{|T|} f_{ik} (\frac{1}{r_j} + \frac{1}{w_{\alpha_k}} + \frac{1}{b_{j\alpha_k}}) \times v_i \tag{11}$$

where (reminder):

$$f_{ik} = \begin{cases} 1 & \textit{if } d_i \in \textit{is used by } t_k \\ 0 & \text{otherwise} \end{cases}$$

Using equation (11), the aggregated transfer time matrix $\tau = (\tau_{ij})$, where $1 \leq i \leq m$ and $1 \leq j \leq p$, is thereby calculated.

4.1.1 | Compute transfer time matrix

Algorithm 1 Calculate data transfer time matrix

Input:

$M = (m_1, m_2, \dots, m_p)$: Set of data centers
$T = (t_1, t_2, \dots, t_n)$: Set of tasks
$D = (d_1, d_2, \dots, d_m)$: Set of datasets
$C = (c_1, c_2, \dots, c_p)$: Capacities of the data centers
$V = (v_1, v_2, \dots, v_m)$: Volumes of the datasets
$F = (f_{ij})$: Matrix of relationship between datasets and tasks

Output:

$\tau = (\tau_{ij})$: Matrix of datasets transfer time from a fixed source to all target data centers
$\sigma = (\sigma_{ij})$: Matrix τ sorted row by row (the aim to facilitate the selection of the best values at the row level)

// Computation of τ matrix
for $i \in D$ **do**
  for $j \in M$ **do**
    $\tau(i, j) \leftarrow 0$
    for $k \in T$ **do**
      if $f(i, k) == 1$ **then**
        if $j \neq \alpha_k$ **then**
          $\tau(i, j) \leftarrow \tau(i, j) + (\frac{1}{r_j} + \frac{1}{w_{\alpha_k}} + \frac{1}{b_{j\alpha_k}}) \times v_i$
        end if *// Otherwise the dataset and the task are within the same data center*
      end if
    end for
  end for
// Sort elements of each row $\tau(i, :)$ in ascending order
  $[\sigma(i, :)] \leftarrow sort(\tau(i, :))$ *// $\sigma_i(j) = k$, the k^{th} element of $\tau(i, :)$*
end for

Algorithm 1 calculates the data transfer time matrix as mentioned in the previous section. The algorithm complexity is depicted as follows: $\mathcal{O}(I \times J \times K)$, where $I = card(M)$, *the number of data centers,* $J = card(T)$, *the number of tasks and* $K = card(D)$, *the number of datasets*.

For every data d_i, we run through each data center m_j to estimate the needed time τ_{ij} of moving d_i from the data center m_j to every other location m_{α_k} where one of its consumer tasks t_k is assigned to (line 9 to 19). τ_{ij} in line 15 is measured using:

- Data size v_i
- Read speed r_j of data center m_j, which is initially designed as the potential location of d_i
- Write speed w_{α_k} of data center m_{α_k} where d_i should be migrated to for the execution of task t_k
- Bandwidth $b_{j\alpha_k}$ between the two data centers m_j and m_{α_k} (task t_k is executed on data center m_{α_k})

Once the matrix τ is calculated, each of its rows (line 20) is sorted in ascending order and the corresponding permutation is provided by the array σ_i. More precisely, $\sigma_i(j)$ is the index of the data center corresponding to the j^{th} migration time cost of d_i in the ascending order. The goal is to get a better data center for each dataset, starting with the one that yields the smallest transfer time, otherwise the second one, and so one. We would also like to point out that the data transfer matrix is computed only once during our simulation and then used each time it is needed.
As an example, we consider:

- 3 data centers $\{m_1, m_2, m_3\}$
  with $(c_1, c_2, c_3) = (80, 40, 120)$, $(r_1, r_2, r_3) = (1, 2, 3)$, $(w_1, w_2, w_3) = (1.5, 2, 1)$ and $(b_{12}, b_{13}, b_{23}) = (5, 4, 6)$.
- 3 tasks $\{t_1, t_2, t_3\}$ with $\alpha = (1, 3, 2)$.
- 4 datasets $\{d_1, d_2, d_3, d_4\}$ with $(v_1, v_2, v_3, v_4) = (45, 20, 25, 50)$.

Matrix F is given by:

$$F = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{12}$$

The matrix F shows that d_1 is used by t_2 and t_3 (first line), d_2 by t_1 and t_3 (second line), and so on. The corresponding graph of the relationship between the datasets and the tasks is depicted in Figure 1 :

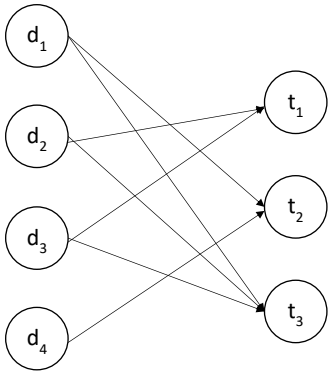

FIGURE 1 Relationship between datasets and tasks

Then matrix τ is computed by Algorithm1-line 15:

$$\tau = \begin{pmatrix} 177.75 & 75 & 45 \\ 34 & 27.33 & 45 \\ 42.5 & 34.17 & 56.25 \\ 112.5 & 83.33 & 0 \end{pmatrix} \tag{13}$$

$\tau(1,1)$ indicates global expected time to migrate d_1 from m_1 to m_{α_2} and m_{α_3}, where the consumer tasks t_2 and t_3 are respectively assigned. Eventually, we sort τ matrix row by row as shown in line 20 of Algorithm 1 and get new matrix σ below:

$$\sigma = \begin{pmatrix} 3 & 2 & 1 \\ 2 & 1 & 3 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{pmatrix} \tag{14}$$

$\sigma(1,1) = 3$, means that the smallest transfer time of d_1 in comparison to other data centers is obtained with data center m_3 and the corresponding cost is $\tau(1,3) = 45$ units of time. In contrast, the last column of the same row (thus the same dataset) yields the highest migration time of 177.75 (i.e. $\tau(1,1)$ and $\sigma(1,3) = 1$) obtained with data center m_1.

4.1.2 | Algorithm for an efficient placement

To obtain an efficient datasets placement, we implement Algorithm 2, which is the refinement of the results from Algorithm 1. For this purpose, a greedy algorithm is applied and iteratively assigns datasets to the appropriate data centers so as to reduce the total transfer time. After calculating the complexity of the Algorithm 2, we have the following result: $\mathcal{O}(K^2)$, where K = *card(D), the number of datasets*.

We consider a variable c as an index indicating the current column within related to transfer matrix τ, a variable s to present the number of placed datasets and a binary variable *placed* that indicates if a dataset is placed or not.

As long as s is less than the overall number of datasets and c doesn't exceed the total number of data centers in the system (Algorithm 2-line 9), the following steps are done:

- We initialize *max* and k to -1 (line 10-11). k is the index of the most costly choice, this will be use for a relocation in case of storage space saturation.
- We individually examine the datasets and for each one:
  1. We check if the dataset is already stored (line 13). If so we check the next one. Otherwise, we select the column c (current) of $\sigma(i)$ that returns index ℓ (line 14) and is supposed to be a good initial placement for now. Next, we check (line 15) if the transfer time is greater than the current maximum (*max*) and if the remaining capacity storage (c_ℓ) allows to store the corresponding volume v_i.
     If the two conditions are satisfied, then we update *max* with the current transfer time and k with the index of the current dataset.
     Otherwise, we keep the previous values of *max* and k.
  2. The main goal of the bloc from line 12 to line 20 is to go over all unplaced datasets and choose the one with the biggest transfer time that can be stored in the selected data center. Thereby, the transition to the following position $\sigma(i+1)$ is avoided. The latter certainly cost more, considering that we are dealing with an ascending order.
     The idea is, for a specific data center (column c), we search for the eligible dataset with the highest migration time if placed in data center number c without violating the capacity storage constraint. Since the rows of matrix τ are sorted in an ascending order, m_c gives the lowest transfer time. This is a max/min procedure and it places earlier the dataset with the highest potential transfer cost.
- At the end of the iteration on the current column, either we selected one dataset (i.e. $k \neq 1$) or we couldn't mainly because of the capacity storage constraint that was always blocking:
  1. In the first case (line 21 to 26), d_k will be stored in data center number c (considering the sorting, we consider $\sigma[k,c]$ instead of c). Then we set $\phi_k = \ell$ (line 24), where $\ell = \sigma[k,c]$ specifies that d_k is already stored (line 23). Next, we refresh the storage capacity of the selected data center (line 25). Finally, we increment s to move to the next dataset.

Algorithm 2 Algorithm for an efficient datasets placement

Input: */* Refinement of the datasets placement following a greedy approach */*

1: c: index of the current column in the transfer cost matrix
2: s: number of already placed datasets
3: $placed$: binary array indicating the selection status

4: $c \leftarrow 1$
5: $s \leftarrow 0$
6: **for** ($i \leftarrow 1$ *to* m) ***do***
7: $placed[i] \leftarrow 0$
8: ***end for***
9: ***while*** ($s < m$ && $c \leq p$) ***do***
10: $max \leftarrow -1$
11: $k \leftarrow -1$
12: ***for*** $i \leftarrow 1$ *to* m ***do***
13: ***if*** $(placed[i] \neq 1)$ ***then***
14: $\ell \leftarrow \sigma[i, c]$ *// id of our* c^{th} *acceptable data center choice for dataset* d_i
15: ***if*** $((\tau(i, \ell) > max)$ && $(c_\ell - v_i > 0))$ ***then***
16: $max \leftarrow \tau(i, \ell)$
17: $k \leftarrow i$ *// if needed because of space constraint, we will relocate* d_k
18: ***end if***
19: ***end if***
20: ***end for***
21: ***if*** $(k \neq -1)$ ***then***
22: $\ell \leftarrow \sigma[k, c]$ *// id of the acceptable data center for dataset* d_k
23: $placed[k] \leftarrow 1$
24: $\phi[k] \leftarrow \ell$ *//* d_k *will be stored in data center* m_ℓ
25: $c_\ell \leftarrow c_\ell - v_k$ *// update of the capacity of data center* m_ℓ *as it receives dataset k*
26: s++
27: ***else***
28: c++
29: ***end if***
30: ***end while***
31: ***if*** $(s = m)$ ***then***
32: *Data placed successfully !*
33: ***else***
34: *Problem with data placement !*
35: ***end if***

2. In the second case, no compatible data was found, so we increase c in order to consider the next column.

- After having iterate over all the datasets and all the data centers in the system, we check if $(s = m)$, which means that our data placement problem was successfully resolved and that every dataset was assigned to a data center (line 31,32). Otherwise, an issue was encountered (because of the capacity storage constraint) and some datasets couldn't be placed (line 34).

We apply the steps explained above to our previous illustrative example. First, we create an intermediate τ' matrix that corresponds to the transfer time values of σ matrix (equation 14) to help us iterating over the unplaced datasets and pick the one

with the greatest transfer time. τ' matrix is given by equation 15:

$$\tau' = \begin{pmatrix} 45 & 75 & 177.75 \\ 27.33 & 34 & 45 \\ 34.17 & 42.5 & 56.25 \\ 0 & 83.33 & 112.5 \end{pmatrix} \tag{15}$$

$\tau'(1,1) = 45$ is the best transfer time for d_1 and should be stored in m_3 ($\sigma(1,1) = 3$); m_1 ($\sigma(1,3)$) is the last location that can store d_1 with the biggest transfer time $\tau'(1,3) = 177.75$.

Then, we start by the first column in σ matrix looking for the dataset that has the biggest time for migration. Based on the first column of τ' matrix, we can see that d_1 ($v_1 = 45$) has the largest transfer time ($\tau'(1,1) = 45$) and can be stored in m_3 ($\sigma(1,1) = 3$ and $c_3 = 120$) without violating the storage capacity constraint. Now, d_1 is marked as a placed dataset and c_3 the capacity of m_3 is updated to $120 - 45 = 75$. We repeat the same process to place the next dataset, and this time the dataset with the biggest transfer time 34.17 is d_3. We check if m_2 ($\sigma(2,1) = 2$ and $c_2 = 40$) can store d_3 ($v_3 = 25$). Thus, d_3 is saved in m_2 and c_2 is refreshed to $40 - 25 = 15$. The next dataset to place is d_2 with a transfer time of $\tau'(2,1) = 27.33$ and that corresponds to m_2 ($\sigma(1,1) = 2$). But since m_2 ($c_2 = 15$) has not enough space to store d_2 ($v_2 = 20$), then we skip to the next dataset which is d_4. d_2 will be placed later when we move to the second column of σ matrix. d_4 that has a volume of 50 should be placed in m_3 ($\sigma(3,1) = 3$) since $c_3 = 75$. By that, the new c_3 is 35 and d_4 is also marked as placed dataset. After going through the first column of σ matrix and since we still have d_2 that is not already placed, then we need to iterate within the second column of σ matrix. Therefore, we check if m_1 ($\sigma(2,2) = 1$) can store d_2 with a transfer time of 34. c_1 equals 80 and v_2 equals 20, thus d_2 is successfully placed in data center m_2 and c_1 is updated to 60.

The final data placement solution is depicted in Figure 2.

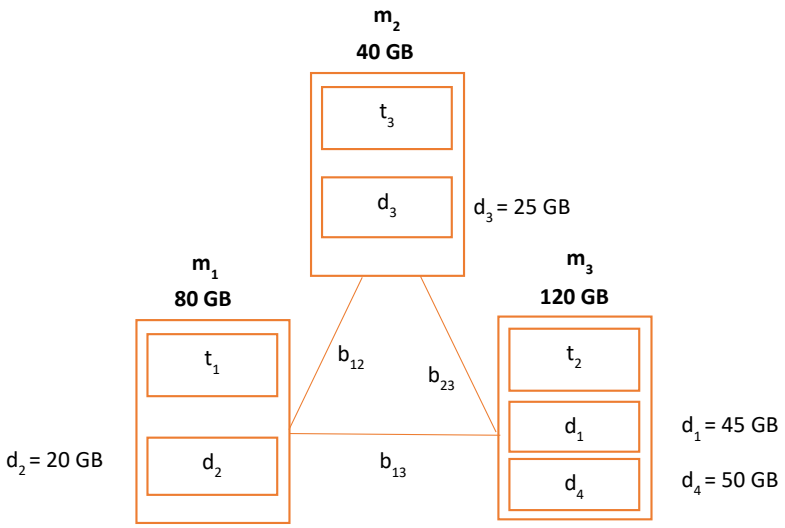

FIGURE 2 A sample data placement solution

4.2 | Data replication placement

The proposed dynamic data replication algorithm is executed at the runtime phase and intends to manage existing and replicated datasets while tasks are being processed. Our proposed technique is implemented following two steps as described in section 4.2.1 & 4.2.2 (Algorithm 3 & 4 successively). The first step tries to find the best data center source that owns a copy of the required dataset for the execution of a given task. The second step aims to manage the replication of the datasets during their transfers. It decides about the datasets to delete in case storage space saturation. The suppression process is based on the number

of replicas and the dependency between datasets and tasks. It worth noting that many works deal with replication in the build-time phase by finding a better number of replicas and a better placement before starting the execution of the tasks. Our idea in this work is to create and reuse the replicas during the transfers of the datasets.

4.2.1 | Source data center selection

The goal of this step is to find the best data center source to get the data needed in a given data center. The main idea is to choose the data center with the smallest transfer time as done by Algorithm 3.

For each task t_i, we go through all the datasets to check which ones are required for the execution of that task t_i (line 18) as follows considering a specific dataset d_j:

1. **If** d_j is available locally (i.e. the same data center m_{α_i} where t_i is executed - line 19). Though, d_j is either initially placed and in that case we talk about m_{ϕ_j} (initial placement result of Algorithm 2) or already stored as a replica in m_{α_i} via a migration process and in that case we talk about $e_{\alpha_i,j} = 1$.
   If one of the conditions is true, then d_j is indeed available locally and can be directly consumed by t_i. In that case, we increment the execution time of t_i (line 20).

2. **Otherwise** (line 21 to line 44), d_j is in different(s) source(s). For its local availability for t_i, d_j should be migrated from a chosen source (if multiple ones) to m_{α_i} while considering data center's capacity.
   Basically, we have two steps:

   (a) Find the best distant source from where to read d_j for t_i. To achieve that, we go through all the data centers (line 22) and see which ones are storing d_j (original or a replica) (line 23).
   For each d_j found in a specific m_k, we compute its migration time $mt^{(i)}(j,k)$ from m_k (source node) to m_{α_i} (destination node), then save it in matrix $mt^{(i)}$ (line 24).
   At the end, every d_j will have a row j in matrix $mt^{(i)}$ filled with estimated time to transfer it from every distant data center to m_{α_i}. If d_j is not found in a distant source then $mt^{(i)}(j,k)$ is equal to ∞ (line 26).
   As last, we sort the row j that corresponds to d_j in $mt^{(i)}$ matrix in ascending order and we get matrix $mtS^{(i)}$ where $mtS^{(i)}(j,k) = l$ is the l^{th} element of $mt_j^{(i)}$ (line 30). Then we select the first element in $mtS^{(i)}(j,0)$ which gives the best data center source m_{s_j} that providing d_j with the smallest transfer time $mt^{(i)}(j,s_j)$ (line 31).
   The goal of this phase is to select the best data center from where we can read our dataset, transfer it and write it where the consumer task is executed. All the migrated datasets while executing tasks are considered as a replicas and may be deleted if needed in case of space issue.
   Now, the best source is successfully chosen.

   (b) Verify m_{α_i}'s capacity where d_j should be migrated to. Lines from 32 to 40 in Algorithm 3 and the function in Algorithm 4 depict the way we deal with data centers capacities and how we manage the replicated datasets.

4.2.2 | Select datasets to delete

As mentioned before while executing tasks, several datasets are moving from one location to another so as to be locally available on the node where the consumer task is assigned to. As we create replicas on the fly, this might lead to have many replicas. Due to storage limitation, keeping all the replicas may prevent storing new incoming useful data or creating new replicas. Thus, we need to delete some copies whenever necessary. We propose a dynamic data replication management strategy, which is an extension of our previous work[2] about data placement. Our strategy (explained in section 4.2.2) aims to decide about which datasets to remove or keep.

When m_{α_i} don't have enough space available for d_j (line 32 in Algorithm 3), we execute "SelectDatasetToDelete" function (Algorithm 4), to chose which datasets to delete so as to free enough space.
In fact, the function takes two parameters: the index of the concerned data center and the arrival time $arrivalTimeTask$ of the task for which we want to gather the required datasets. The steps of this function are:

1. We iteratively examine the tasks (Algorithm 4, line 2 to line 6) looking for the ones having their arrival time (w_i) greater than $arrivalTimeTask$ (line 3), which means they are not yet executed at that time, and add them to vector δ. Then, we compute the dependency between these tasks and the datasets as explained below.

Algorithm 3 Data replication placement - Multiple sources

Input:

p: Number of data centers
$M = (m_1, m_2, \dots, m_p)$: Set of data centers
$C = (c_1, c_2, \dots, c_p)$: Capacities of the data centers
n: Number of tasks
$T = (t_1, t_2, \dots, t_n)$: Set of tasks
$\omega = (\omega_1, \dots, \omega_n)$: Arrival time of tasks for the execution
m: Number of datasets
$D = (d_1, d_2, \dots, d_m)$: Set of datasets
$V = (v_1, v_2, \dots, v_m)$: Volumes of the datasets
f_{ij}: Relationship between datasets and tasks ($f_{ij} = 1$ if t_j needs d_i)
$\alpha = (\alpha_1, \dots, \alpha_n)$: Array of tasks assignment
$wcc = (wcc_1, \dots, wcc_n)$: Workflow communication cost, where wcc_i is the execution time of t_i
ϕ : Index of optimal placement of datasets
E: Matrix to specify if d_j is a replica in m_i so $e_{ij} = 1$

Output: $S = (s_j^{(i)})$: Index of chosen data center as source to migrate d_j when executing t_i

// Migrate data

for $i \leftarrow 1$ *to* n ***do***
  $wcc_i \leftarrow 0$ *// Execution time of* t_i
  for $j \leftarrow 1$ *to* m ***do***
    if $(f_{j,i} \neq 0)$ ***then***
      if $((m_{\alpha_i} = m_{\phi_j})$ *or* $(e_{\alpha_i,j} = 1))$ ***then*** *// Verify if* d_j *is stored in local (as original or replica)*
        $wcc_i \leftarrow wcc_i + 0$ *// No data migration is needed*
      else *//* d_j *isn't stored in local and should be migrated to* m_{α_i}
        for $k \leftarrow 1$ *to* p ***do***
          if $((m_k = m_{\phi_j})$ *or* $(e_{k,j} = 1))$ ***then*** *// Verify if* d_j *is available in distant sources*
            $mt^{(i)}(j,k) \leftarrow (\frac{1}{r_k} + \frac{1}{w_{\alpha_i}} + \frac{1}{b_{k\alpha_i}}) \times v_j$ *// Compute time to migrate* d_j *(as original or replica) from the distant source* m_k *to local source* m_{α_i} *for* t_i *execution*
          else
            $mt^{(i)}(j,k) \leftarrow \infty$
          end if
        end for
        //Sort row 'j' of migration time matrix corresponding to d_j*, in ascending order*
        $[mtS^{(i)}(j,:)] \leftarrow sort(mt^{(i)}(j,:))$ *//* $mtS^{(i)}(j,k) = l$*, the* l^{th} *element of* $mt^{(i)}(j,:)$
        $s_j^{(i)} \leftarrow mtS^{(i)}(j,0)$ *// Select placement with the smallest transfer cost as a source*
        while $((c_{\alpha_i} - v_j) < 0)$ ***do*** *//* m_{α_i}*'s capacity is not enough to store* d_j
          $selectedData \leftarrow$ SELECTDATASETTODELETE(α_i, ω_i) *// Decide which dataset to remove from* m_{α_i} *to free up space and store the migrated* d_j*,* ω_i *is the arrival time of* t_i
          if $(selectedData \leftarrow null)$ ***then***
            break; // PB : No replicated dataset was found to be deleted.
          else
            $m_{\alpha_i}.delete(selectedData)$
            $c_{\alpha_i} \leftarrow c_{\alpha_i} + v_{selectedData}$
          end if
        end while
        $wcc_i \leftarrow wcc_i + f_{ij} \times (\frac{1}{r_{s_j^{(i)}}} + \frac{1}{w_{\alpha_i}} + \frac{1}{b_{s_j^{(i)}\alpha_i}}) \times v_j$ *// Update execution time of* t_i *that consumes* d_j
        $e_{\alpha_i,j} \leftarrow 1$
        $c_{\alpha_i} \leftarrow c_{\alpha_i} - v_j$
      end if
    end if
  end for
  $wcc \leftarrow wcc + wcc_i$ *// Compute global workflow communication cost*
end for

2. We browse over the datasets in m_{α_i} denoted by $m_{indexDC}$. We only consider the replicas (Algorithm 4, line 12) and select which one to delete. The deletion process is based on:

   (a) **Dependency between tasks and datasets** (Algorithm 4, line 13 to line 17): this factor aims to specify how many unexecuted tasks require each replicated dataset. So, for each t_k in δ, we verify which task requires the replica d_j (when f_{kj} is equal to 1 - line 14). Then, we increment the value of $depC_j$, which corresponds to how many tasks are using d_j.

   (b) **Number of existing replicas of the dataset** (Algorithm 4, line 18 to line 22): this factor aims to specify the number of existing replicas for each dataset. So, for each data center m_i (line 18) we verify if it stores d_j as a replica copy (line 19). Whenever a replica is found, the value of $repC_j$ is increased by one. ($repC_j$ corresponds to existing number of replicas of d_j)
   Since a dataset might be replicated many times, only those having more that $maxRep$ replicas (here equals 2) are eligible for deletion. Another step is added (Algorithm 4, line 24 to line 26) to present what we have previously explained by adding each resulting dataset to the $tempList$ list.

3. Once we establish our factors, now we can apply our strategy in Algorithm 4 (line 30 to line 39) to store the migrated dataset locally. The approach we proposed in our work considers a lexicographic sorting to take the decision. The goal of this sorting algorithm is to pick d_i that has the lowest $depCount_i$ (the least used d_i from the tasks not executed yet). In case we have many datasets with the same $depCount$, we consider the one having the greatest $repCount$ (the most duplicated dataset across the other data centers).

4. As result to the lexicographic sorting, a new list $sortList$ is created, then we choose the first element on it, which is denoted by $selectedData$ in Algorithm 4, line 40. $selectedData$ shows the index of the best dataset to remove. Finally, the result is returned to the main Algorithm 3 in line 33.

An exception is thrown if no $selectedData$ is found (Algorithm 3, line 35), in that case the system has faced an issue and therefore stops. Otherwise (line 37 & 38), $d_{selectedData}$ is deleted from m_{α_i} and its capacity is henceforth incremented.
In some cases, removing $d_{selectedData}$ does not free enough space for dataset d_j that we want to migrate. To fix this problem we keep processing the $SelectDatasetToDelete$ function until we get the a sufficient amount of available space to transfer dataset d_j (loop in Algorithm 3, line 32).
Once out of the loop, data center m_{α_i} can finally receive d_j. The migration process is started and the task t_i can eventually consume d_j; the execution time of the task is updated (line 41); we indicate that d_j is stored in m_{α_i} (line 41) and update the capacity of m_{α_i} (line 43).

N.B. : The whole process as described above is repeated for each required dataset for task t_i, and the execution time is updated accordingly (line 20 or 31) to give the workflow communication cost wcc_i suitable for t_i.
Moreover, the estimated complexity of Algorithms 3 & 4 is depicted in Table 2, where *I = card(M), the number of data centers, J = card(T), the number of tasks and I = card(D), the number of datasets*.

TABLE 2 Algorithms Complexity

Algorithm	Asymptotic complexity
Algorithm 3	$\mathcal{O}(K^4 \times J \times I + K^3 \times I^2 \times J + K^3 \times J^2 \times I + K^2 \times J^2 \times I + J \times K \times I)$
Algorithm 4	$\mathcal{O}(K^2 + K \times I + K \times J)$

To evaluate our algorithm, we sum the wcc_i of every task to get the global communication cost wcc (Algorithm 3, line 47) that will be used in the experiments described in section 5.

Algorithm 4 Data replication placement - Datasets to delete from the destination node

```
1: function SELECTDATASETTODELETE(indexDC, arrivalTimeTask)
2:     for i ← 1 to n do
3:         if (ω_i > arrivalTimeTask) then
4:             δ.add(i)
5:         end if
6:     end for
7:     tempList ← null
8:     maxRep ← 2  // threshold of the number of replicas for deletion consideration
9:     for j ← 1 to m do
10:        depC_j ← 0  // dependency counter
11:        repC_j ← 0  // replica counter
12:        if ((e_indexDC,j = 1)) then      // Verify if d_j is a replica in m_indexDC
13:            for k ← 1 to  δ.length do
14:                if (f_kj = 1) then
15:                    depC_j ← depC_j + 1
16:                end if
17:            end for
18:            for i ← 1 to p-1 do       // (p-1) is the number of data centers except m_indexDC
19:                if (e_ij = 1) then
20:                    repC_j ← repC_j + 1
21:                end if
22:            end for
23:        end if
24:        if (repC_j > maxRep) then
25:            tempList.add(d_j)       // This array wil be eventually sorted
26:        end if
27:    end for
28:    sortList ← null
29:    length ← tempList.length
30:    for i ← 1 to m do       // Lexicographic sorting
31:        bestDataset ← i
32:        for j ← 1 to length do
33:            if (depC_j ≠ depC_i && depC_j ≥ depC_i) or (depC_j = depC_i && repC_j ≥ repC_i) then
34:                bestDataset ← j
35:            end if
36:        end for
37:        sortList.add(bestDataset)
38:        tempList.remove(bestDataset)
39:    end for
40:    selectedData ← sortList(0)    // The first dataset in sortList is the best choice for deletion
41: end function
```

5 | EXPERIMENTS AND DISCUSSION

5.1 | Simulation set-up

In order to inspect the behavior of our algorithms and evaluate their impacts, we compare the following strategies:

1. **Build-time strategy**[2]: the datasets are (efficiently) stored before runtime and immediately deleted once consumed (this was the focus of our previous work).

2. **Kouidri's strategy[30]:** he suggests to keep the most used replicas, storing them where there is enough space.

3. **Combination of data placement and replication strategies:** describes our two-steps approach.

For our experiments, we consider 1000 tasks, assuming that each task is assigned to one data center (no task migration). The number of data centers is 5, 10, 15 and 25, with a uniform distribution within the range [1PB, 25PB] for the storage capacities. In addition, for each task, we randomly assign atb most five different datasets. We limit the number of datasets to 5, 10, 25, 50, 75, 100 and 200. The sizes of the datasets are uniformly distributed within the range [1TB, 100TB]. Overall description is shown in Table 3.

We generate different scenarios of workloads and for each, we consider 100 executions and take the mean value as the final result. We use an extended version[1] of Cloudsim[34] framework, which is a well-known simulator for single and multi-Cloud scenarios. However, it lacks handling and evaluating data migration issues, as well as providing the execution time of the tasks. For this reason, our extension (1) proposes a module to simulate data migration; (2) implements and tests its own data migration strategy; (3) describes data movements across geo-distributed data centers; and (4) estimates the overall time to transfer datasets.

TABLE 3 Default setting for our experiment

Components	Values
# of datasets	[5, 10, 25, 50, 75, 100]
Dataset size	[1TB - 100TB]
# of data centers	[5, 10, 15, 20, 25]
data center capacity	[1PB - 25PB]
Storage cost	$0.1 per GB
Transfer cost	$0.05 per GB
Penalty cost	$0.01 per violation

We would like to mention that the proposed approaches (including the greedy algorithm) can run on a real Cloud by construction, but this will require extra efforts and the results might be different from that of the simulations. Running on a real cloud, especially with applications managing a huge volume of data, might be costly and source of several technical issues that are not relevant for our consideration in this work. However, we consider this transition in our perspectives.

5.2 | Simulation results

In our experiments, we evaluate our proposed two-steps strategy and compare the performances with the build-time phase (without replication) and Kouidri's approach. We mainly consider the workflow communication cost (WCC) as described in section 3.3.

Using our extended simulation tool[1], we simulate each workflow 100 times and consider the average value of the WCC.

In the first experiment, we set the number of geographically distributed data centers to 5, 10, 15, 20 and 25, and for each case we vary the number of datasets from 5 to 200 while estimating WCC as shown in Figure 3. We can notice that in all cases, WCC continuously increases with the growth of the number of datasets. Despite this, our results show that our combined strategy outperforms the build-time approach (data placement without replication) and the kouidri's work (i.e. lower global transfer time). For example, in the case of 20 data centers, owe get a reduction of 75.31% and 48.02% respectively. Moreover, for the same case, our strategy yields a variance of 3.67 hours for the timings with 5 to 200 datasets, while the no-replication and Kouidri give 13.92 hours and 6.8 hours respectively. Thereby, we can say that our proposed algorithm is always better regarding the reduction of communication cost and running time fluctuation.

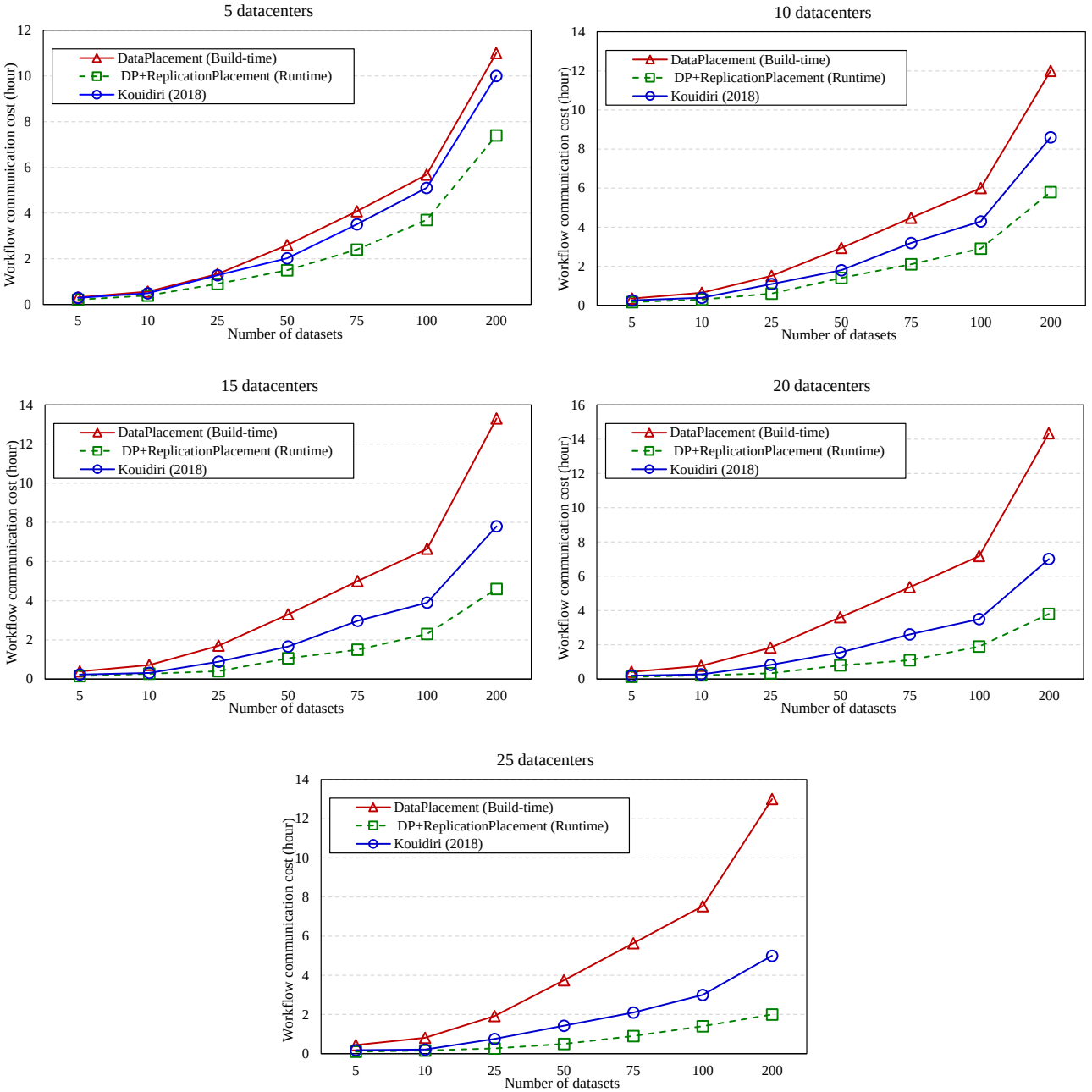

FIGURE 3 Communication cost with different number of datasets and a fixed number of data centers

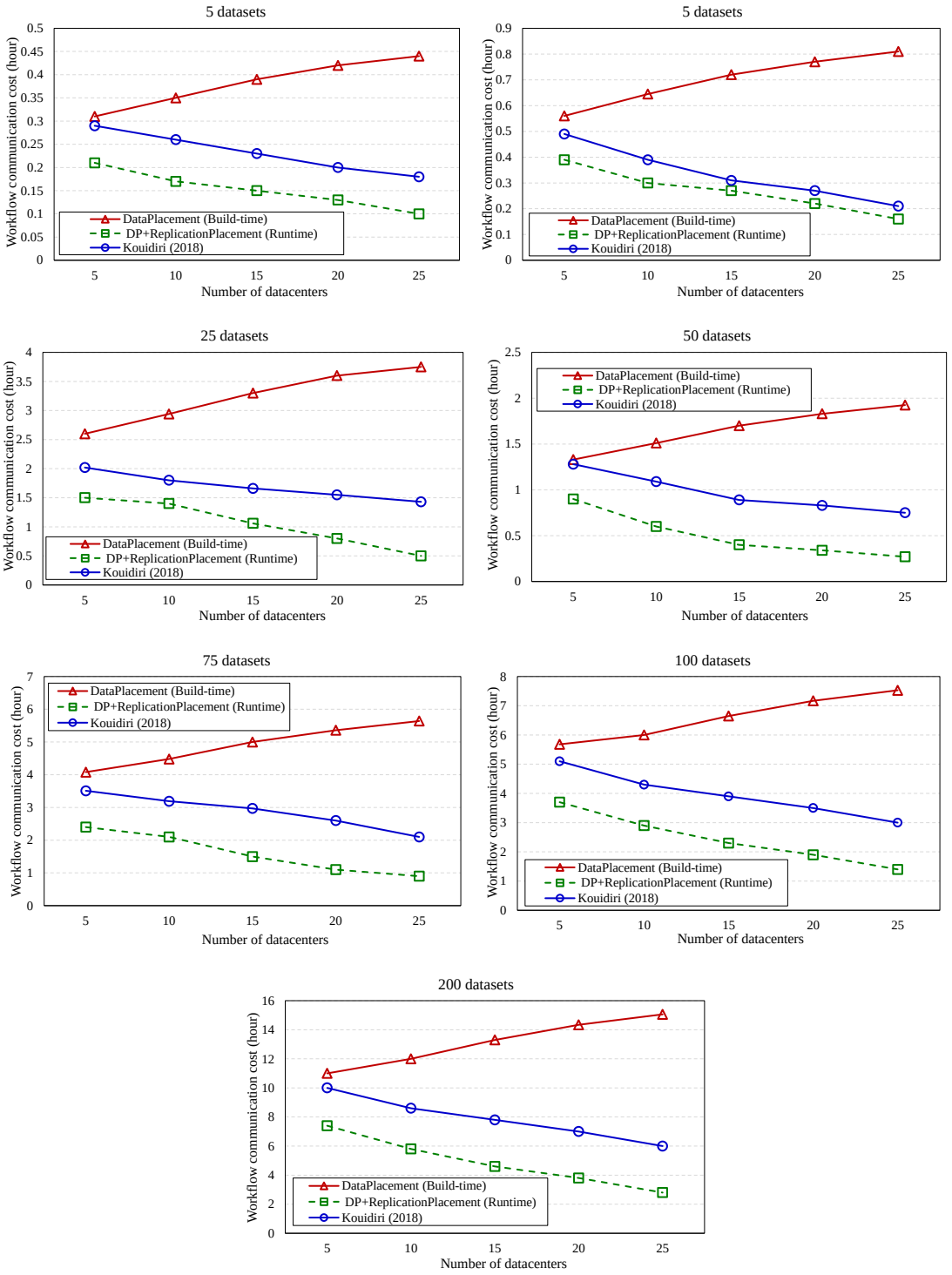

FIGURE 4 Communication cost with different number of data centers and a fixed number of datasets

In the second experiment, as shown in Figure 4, we set the number of datasets to 5, 10, 25, 50, 75, 100 and 200 and vary the number of geographically distributed data centers within the range [5, 25]. We observe that by increasing the number of data centers, the WCC for the no-replication appraoch increases. In contrast, WCC seems to decrease with our approach as well as with Kouidri.

In the case with 50 datasets, our approach gives 1.05 hours on average, next to 3.23 and 1.7 hours on average for the no-replication and Koudiri's algorithm respectively. Thus, our proposed approach reduces effectively the cost of data movement by 67.51% and 37.82%, respectively. The results show an increasing improvment with more and more data centers.

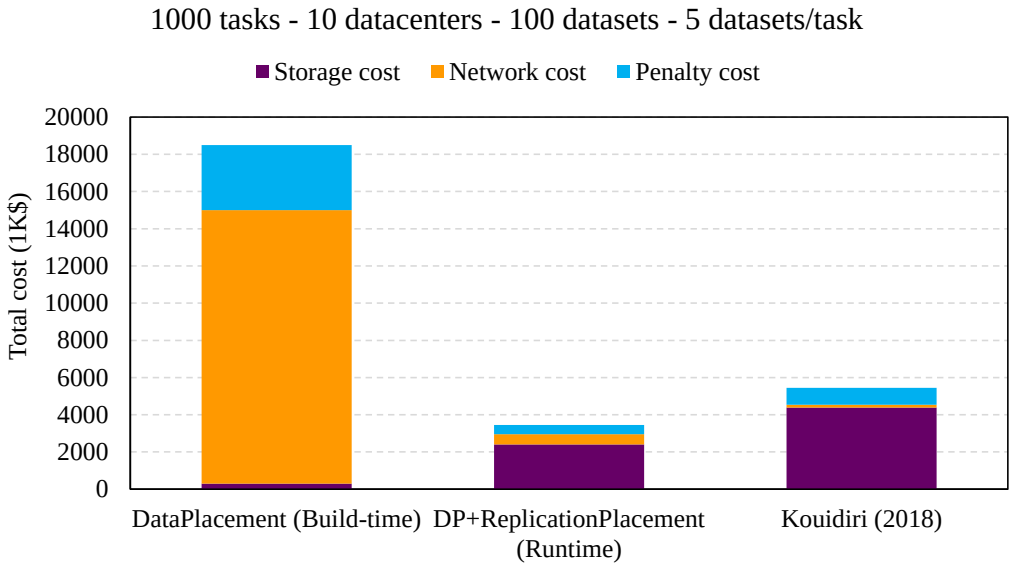

FIGURE 5 Total monetary cost

Figure 5 shows the total monetary cost of each strategy over the simulated billing period. All strategies are evaluated from the case of 15 data centers and varying datasets within the range [5, 200]. About the storage cost, the value looks huge with the no-replication approach. With the other two strategies, a high number replicated datasets is produced. However, our proposed startegy reduces the storage cost by 45% compared to the Kouidri's strategy. In terms of network cost, the build-time strategy generates a very high cost. Our proposed approach does have a little more network cost compared to Kouidri's approach, but this can be considered as a compromise to save on the storage cost. In addition, with no-replication appraoch, the high number of SLA violations results in a high penalty. Whereas apprshow a low number of SLA violations. Based on total costs, data placement and replication strategy reduce the total financial costs by 81.33% and 36.5% compared to the build-time and Kouidri's strategy respectively. This cost reduction shows the efficiency of our work in improving the performance of the Cloud system.

6 | CONCLUSION AND FUTURE WORK

This paper proposes a combination of data placement and data replication to address the problem of data movements across geographically distributed data centers in the Cloud Computing environment. Our aim is to reduce the total transfer cost of moving the datasets between data centers when executing a given workflow. Our data placement approach is a static algorithm that begins with creating a matrix of data migration costs followed by a greedy algorithm to get the final placement of the datasets over the data centers. Our data replication approach is a dynamic procedure that is considered during the runtime phase. While tasks are running, our algorithm searches for the best sources from where to get required datasets. A conjunction of both strategies lead to noticeable improvment compare to a naive management and a selected contribution from the litterature. Indeed, using the extension of Cloudsim framework, we compared our work with other studies in the literature and see that we perform better regarding the minimization of time consumption for moving datasets as well as in reducing the monetary cost

by re-using the replicas. As a future work, we intend to simulate our exprimental study considering Peagsus generator for more consistency in the measurements and to implement our solution in a real platform for realistic results. Another possible future direction to reduce the overall execution time is to propose (1) a migration strategy that considers a routing procedure for data migration and (2) a dynamic tasks allocation in conjunction with our data management strategies.

References

1. Bouhouch L, Zbakh M, Tadonki C. Data Migration: Cloudsim Extension. In: Proceedings of the 2019 3rd International Conference on Big Data Research (ICBDR 2019); ACM International Symposium; DOI:https://doi.org/10.1145/3372454.3372472.

2. Bouhouch L, Zbakh M, Tadonki C. A Big Data Placement Strategy in Geographically Distributed data centers. In: IEEE 5th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech20); Marrakesh, Morocco; 2020.

3. I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid com- puting 360-degree compared," in 2008 Grid Computing Environments Workshop. IEEE, Nov. 2008, pp. 1–10.

4. M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A view of Cloud computing," Commun. ACM, vol. 53, no. 4, p. 50, 2010.

5. Berriman G. B, Juve G, Deelman E, Regelson M, Plavchan P. The Application of Cloud Computing to Astronomy: A Study of Cost and Performance. In: 6th IEEE International Conference on e-Science Workshops; Brisbane, QLD; 2010:1-7; doi: 10.1109/eScienceW.2010.10.

6. Coutinho E, Rego P, Gomes D, Souza J N. Physics and microeconomics-based metrics for evaluating Cloud Computing elasticity. In: Journal of Network and Computer Applications; 2016:Volume 63; 159-172; ISSN 1084-8045; https://doi.org/10.1016/j.jnca.2016.01.015.

7. Navale V, Bourne PE. Cloud Computing applications for biomedical science: A perspective. In: PLoS Comput Biol 14(6): e1006144. 2018; https://doi.org/10.1371/journal.pcbi.1006144.

8. Sadiku M, Musa S, Momoh O. Cloud Computing: Opportunities and Challenges. In: IEEE Potentials; Volume 33, no. 1, 34-36, Jan.-Feb; 2014; doi: 10.1109/MPOT.2013.2279684.

9. Furht B, Villanustre F. Introduction to Big Data. In: Big Data Technologies and Applications. Springer, Cham; 2016; https://doi.org/10.1007/978-3-319-44550-2_1.

10. Xiao W, Bao W, Zhu X, Liu L. Cost-Aware Big Data Processing Across Geo-Distributed data centers. In: IEEE Transactions on Parallel and Distributed Systems; 2017:Volume 28, no. 11, 3114-3127, 1 Nov; doi: 10.1109/TPDS.2017.2708120.

11. Chaowei Y, Qunying H, Zhenlong L, Kai L Fei H. Big Data and Cloud Computing: innovation opportunities and challenges. In: International Journal of Digital Earth; 2017; 10:1, 13-53; DOI: 10.1080/17538947.2016.1239771.

12. Long S, Zhao Y, Chen W. MORM: A Multi-objective Optimized Replication Management strategy for Cloud storage cluster. In: J Syst Arch; 2014; 60 (2):234–44.

13. Shvachko K, Hairong K, Radia S, Chansler R. The Hadoop distributed file system. In: Proceedings of the 26th Symposium on Mass Storage Systems and Technologies; 2010:1–10.

14. Zeng Z, Veeravalli B. Optimal metadata replications and request balancing strategy on Cloud data centers. In : J Parallel Distrib Comput; 2014; 74 (10):2934–40.

15. Rahman RM, Barker K, Alhajj R. Replica placement design with static optimality and dynamic maintainability. In: Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid; 2006:434–437.

16. Subia S, Samar W. Performance Analysis of Big Data and Cloud Computing Techniques: A Survey. In: Procedia Computer Science; 2018: Volume 132, 118-127; ISSN 1877-0509, https://doi.org/10.1016/j.procs.2018.05.172.

17. Nayak J, Naik B, Jena A, Barik R, Das H. Nature Inspired Optimizations in Cloud Computing: Applications and Challenges. In: Cloud Computing for Optimization: Foundations, Applications, and Challenges. Studies in Big Data; 2018:Volume 39. Springer, Cham. https://doi.org/10.1007/978-3-319-73676-1_1.

18. Arunarani AR, Manjula D, Sugumaran V. Task scheduling techniques in Cloud Computing: A literature survey. In: Future Generation Computer Systems; 2019:Volume 91, 407-415; ISSN 0167-739X; https://doi.org/10.1016/j.future.2018.09.014.

19. Mazumdar S, Seybold D, Kritikos K. et al. A survey on data storage and placement methodologies for Cloud-Big Data ecosystem. In: J Big Data 6, 15; 2019; https://doi.org/10.1186/s40537-019-0178-3.

20. Bahareh A M, Navimipour N J. A Systematic Literature Review of the Data Replication Techniques in the Cloud Environments. In: Big Data Research; 2017:Volume 10, 1-7; ISSN 2214-5796; https://doi.org/10.1016/j.bdr.2017.06.003.

21. Yuan D, Yang Y, Liu X, Chen J. A data placement strategy in scientific Cloud workflows. In: Future Generation Computer Systems; 2010:Volume 26, Issue 8, 1200–1214, https://doi.org/10.1016/j.future.2010.02.004.

22. Qiang X, Zhengquan X, Tao W. A Data-Placement Strategy Based on Genetic Algorithm in Cloud Computing. In: International Journal of Intelligence Science; 2015:05. 145-157; https://doi.org/10.4236/ijis.2015.53013.

23. Er-Dun Z, Yong-Qiang Q, Xing-Xing X, Yi C. A Data Placement Strategy Based on Genetic Algorithm for Scientific Workflows. In; 8th International Conference on Computational Intelligence and Security, Guangzhou; 2012:146–149, https://doi.org/10.1109/CIS.2012.40.

24. Zhao Q, Xiong C, Zhao X, Yu C, Xiao J. A Data Placement Strategy for Data-Intensive Scientific Workflows in Cloud. In: 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing; Shenzhen; 2015:928–934; https://doi.org/10.1109/CCGrid.2015.72

25. Ebrahimi M, Mohan A, Kashlev A, Lu S. BDAP: A Big Data Placement Strategy for Cloud-Based Scientific Workflows. In: IEEE 1st International Conference on Big Data Computing Service and Applications; Redwood City, CA; 2015:105–114, https://doi.org/10.1109/BigDataService.2015.70

26. Tos U, Mokadem R, Hameurlain A, Ayav T, Bora S. A Performance and Profit Oriented Data Replication Strategy for Cloud System. In: 2nd IEEE International Conference on Cloud and Big Data Computing (CBDCom 2016); Toulouse, France; 2016:780-787. hal-01690142

27. Lihui L, Junping S, Haibo W, Pin L. BRPS: A Big Data Placement Strategy for Data Intensive Applications; 2016:813-820. 10.1109/ICDMW.2016.0120.

28. Najme M. Adaptive data replication strategy in Cloud Computing for performance improvement. In: Frontiers of Computer Science. 10; 2016; 10.1007/s11704-016-5182-6.

29. Xie F, Yan J, Shen J. Towards Cost Reduction in Cloud-Based Workflow Management through Data Replication. In : 5th International Conference on Advanced Cloud and Big Data (CBD); Shanghai; 2017:94-99, doi: 10.1109/CBD.2017.24.

30. Kouidri S, Yagoubi B. Dynamic Data Replication Based on Tasks scheduling for Cloud Computing Environment. In: International Journal of Strategic Information Technology and Applications. 8. 2017:40-51. 10.4018/IJSITA.2017100104.

31. Amadeo A. Optimising Data Access with an Adaptive Geo–Replication Strategy. In: International Journal of Swarm Intelligence and Evolutionary Computation. 07; 2018; 10.4172/2090-4908.1000171.

32. Badshah, Afzal; Ghani, Anwar; Shamshirband, Shahaboddin; Aceto, Giuseppe; Pescapè, Antonio: 'Performance-based service-level agreement in cloud computing to optimise penalties and revenue', IET Communications, 2020, 14, (7), p. 1102-1112, DOI: 10.1049/iet-com.2019.0855 IET Digital Library, https://digital-library.theiet.org/content/journals/10.1049/iet-com.2019.0855.

33. Tos, Uras & Mokadem, Riad & Hameurlain, Abdelkader & Ayav, Tolga. (2021). Achieving query performance in the cloud via a cost-effective data replication strategy. Soft Computing. 25. 10.1007/s00500-020-05544-w.

34. Calheiros R N, Ranjan R, Beloglazov A, Buyya, R. CloudSim: a toolkit for modeling and simulation of Cloud Computing environments and evaluation of resource provisioning algorithms. In: Softw: Pract. Exper.; 2011; 41: 23-50. https://doi.org/10.1002/spe.995.