Exact and heuristic data workflow placement algorithms for big data computing in cloud datacenters

Sonia Ikken, Eric Renault, Abdelkamel Tari, Tahar Kechadi

Exact and Heuristic Data Workflow Placement Algorithms for Big Data Computing in Cloud Datacenters

Article *in* Scalable Computing Practice and Experience · September 2018

DOI: 10.12694/scpe.v19i3.1365

4 authors:

Sonia Ikken
United Lutheran Seminary
6 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Éric Renault
Ecole Supérieure d'Ingénieur en Electronique et Electrotechnique - Paris
157 PUBLICATIONS **701** CITATIONS
SEE PROFILE

Abdelkamel A Kamel Tari
Université de Béjaïa
125 PUBLICATIONS **1,508** CITATIONS
SEE PROFILE

Tahar Kechadi
University College Dublin
451 PUBLICATIONS **5,568** CITATIONS
SEE PROFILE

DOI 10.12694/scpe.v19i3.1365
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 19, Number 3, pp. 223–244. http://www.scpe.org

EXACT AND HEURISTIC DATA WORKFLOW PLACEMENT ALGORITHMS FOR BIG DATA COMPUTING IN CLOUD DATACENTERS

SONIA IKKEN[*], ERIC RENAULT[†], ABDELKAMEL TARI[‡], AND M. TAHAR KECHADI[§]

Abstract. Several big data-driven applications are currently carried out in collaboration using distributed infrastructure. These data-driven applications usually deal with experiments at massive scale. Data generated by such experiments are huge and stored at multiple geographic locations for reuse. Workflow systems, composed of jobs using collaborative task-based models, present new dependency and data exchange needs. This gives rise to new issues when selecting distributed data and storage resources so that the execution of applications is on time, and resource usage-cost-efficient. In this paper, we present an efficient data placement approach to improve the performance of workflow processing in distributed datacenters. The proposed approach involves two types of data: splittable and unsplittable intermediate data. Moreover, we place intermediate data by considering not only their source location but also their dependencies. The main objective is to minimise the total storage cost, including the effort for transferring, storing, and moving that data according to the applications needs. We first propose an exact algorithm which takes into account the intra-job dependencies, and we show that the optimal fractional intermediate data placement problem is NP-hard. To solve the problem of unsplittable intermediate data placement, we propose a greedy heuristic algorithm based on a network flow optimization framework. The experimental results show that the performance of our approach is very promising.

Key words: Big data placement, Data workflow management, Dataflow model, Distributed datacenters, Storage cost minimization, Scalable storage and computing

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction. This study addresses the problem of intermediate data placement in big data-driven applications. These data are usually stored in multiple cloud datacentres. The main goal is to be able to efficiently process and share them between a set of tasks (jobs or services) according to their needs, dependencies, and their resource requirements. In other words, the problem is how to efficiently store and access the intermediate data taking into account the inter- and intra-job (process) dependencies. These jobs can be any traditional process at the level of the operating system or services at the application level. Because of the popularity of the service delivery model, the cloud consists of a set of services that should be provided to the clients. At a very large scale (cloud scale), a huge number of activities coexist generating or consuming huge amount of intermediate data, which are stored in the form of files, called temporary files, in datacentres. From the point of view of data usage, this follows a workflow depending on the dynamic nature the execution of these services. These dependencies are of very high importance for the correct execution of the services that were provided to the clients. Each workflow has different requirements not only in the dependencies of its intermediate data files but also their sizes and types. Moreover, these services (jobs or set of tasks) can be initiated remotely from different geographic locations, which adds a level of complexity of how these data can be accessed without putting a significant stress on the cloud resources (bottlenecks, long waiting queues, etc.). Therefore, the way that these intermediate data files should be manipulated and managed must take into account their near future usage (frequency of use, life cycle, etc.). Accordingly, their management (storage, movement, processing, etc.) should be derived from the workflow of the jobs and services that are using them, which is called "Data Workflow".

This paper proposes a new approach that takes into account the types of dependencies and accesses to the intermediate data, as these are the key factors for improving big data-driven applications while leveraging cloud resources among the clients in an efficient and scalable manner. We start by formulating the intermediate data placement dependencies derived from multiple workflows running on distributed cloud datacenters. Then we model the whole system as a constrained optimization problem, where constraints represent the dependencies derived from the running workflows. The derived constrained optimization problem that includes storage cost is very complex. Two types of data are considered; the intermediate data that are used by the same job with

[*]Faculty of Exact Sciences, University of Bejaia, 06000 Bejaia, Algeria, and Telecom SudParis, Samovar-UMR 5157 CNRS, University of Paris-Saclay, France (sonia.ikken@telecom-sudparis.eu, sonia.ikken@gmail.com).
[†]Telecom SudParis, Samovar-UMR 5157 CNRS, University of Paris-Saclay, France (eric.renault@telecom-sudparis.eu).
[‡]Faculty of Exact Sciences, University of Bejaia, 06000 Bejaia, Algeria (tarikamel59@gmail.com).
[§]UCD School of Computer Science and Informatics, Dublin, Ireland (tahar.kechadi@ucd.ie).

their intra-dependencies from multiple tasks and intermediate data that are used by different jobs with their inter-job dependencies. Since intra-job dependencies can be split partially and placed on different locations (as in MapReduce), the problem is called minimum cost multiple-sources multicommodity flow problem (MCMF). Intermediate data dependencies that are used by different tasks (of a single job) can be split and kept in the same datacenter and preferably at the location of the task. We formulate the problem with these data as splittable demands which can be solved with an exact algorithm to obtain an optimal fractional solution. However, as most of these problems are NP-hard, it is difficult to obtain an optimal solution using exact methods for the variant that deals with unsplittable intermediate data from inter-job dependencies. Greedy approaches implement simple algorithms but effective for unsplittable flow problem [1, 2, 3, 4, 5], and they scale linearly with the number of instances. Their resolution techniques are adaptable to our intermediate data placement problem that the present paper deals with. Experimental results show that the proposed techniques are very promising for storage cost minimisation.

The rest of the paper is organized as follows: Section 2 summaries the related work. Section 3 introduces the system model and problem definition according to the cloud computing environment and data models. The proposed techniques for the optimal intermediate data dependencies placement are presented in Section 4. Section 5 discusses the performance evaluation and the simulation results. We conclude and give some future directions in Section 6.

2. Related works. We have thoroughly investigated recent research works on cloud ressource scheduling in the literature [31, 32, 33]. These works focus primarily on resource sharing and provisioning problems in order to either save energy consumption or reduce its costs by providing efficient application processing. Authors in [31], addressed the power consumption problem and network performance degradation by relying on an optimization model that is based on sliding-scheduled tenant request. The latter allows to manage application execution time as well as their resources for efficient placement and routing. Authors in [32], proposed a genetic algorithmic based heuristic methods to schedule tasks across limited resources, but restricted to use one global cost and time for multiple tasks execution. More recently, authors in [33], addressed the problem of resource scheduling of scientific workflow applications in cloud. They focus on reducing the cost of the communications and the information exchange time across a management framework of multiple-site awareness data administration. Nevertheless, these works did not address the problem of data workflow placement and the dependencies between resources.

Workflow scheduling problems [35, 36] in cloud environments are considered to be very challenging. Many strategies based-heuristic were proposed to solve the tasks scheduling problem without considering the data that are generated by these tasks. Authors in [35], proposed a meta-heuristic approach, called Hybrid GA-PSO (Genetic Algorithm-Particle Swarm Optimization), to solve the workflow tasks scheduling problem. The Hybrid GA-PSO algorithm returns a balanced solution for tasks distribution among different virtual machines in a cloud environment by considering both the total monetary cost and the execution makespan. While the PSO-based algorithm converges quickly to a local optimal solution, this can be far from the global optimal solution. In the same context, authors in [36] proposed a metaheuristic-based algorithm, called Hybrid Bio-inspired Metaheuristic for Multi-objective Optimization (HBMMO), to solve the multiple conflicting objectives optimization problem. The authors considered in their optimization some important requirements of the users or the providers, such as makespan, cost, and load balancing among virtual machines. The proposed HBMMO method optimizes the scheduling of tasks workflow in the cloud environment by considering a non-dominant sorting strategy which is a hybridization of the list-based heuristic algorithm PEFT (Predict Earliest Finish Time) [37] and the discrete version of the metaheuristic algorithm SOS (Symbiotic Organisms Search) [38].

Many researchers [34, 6, 7, 8, 9] have focused on the big data placement optimization problem in distributed system environments. However, most of these studies did not include the dependencies between data workflows. In addition, these solutions did not take into consideration the dependency type constraint for making intermediate data placement decision. Authors in [34], defined an optimization problem based on a greedy heuristic for simultaneous placement of virtual machines and data blocks. A greedy heuristic allows to place on-demand application components by localising network traffic in interconnected datacenters, and therefore, reducing packet transmission delays, increasing network performance, and minimizing the energy consumption of datacenter network infrastructure. Nevertheless, the efficiency of multi-tier application processing through the data com-

munication and correlation is not considered. Authors in [6], proposed a strategy placement for large-volume user's data while minimizing their operational costs of accommodating various social networks. The relation between the user community and dynamic maintenance of the placed user data in an evolving social network are considered, but the use of the data dependency aspects is not explored. In [7], the big data placement problem from a collaborative-aware environment that continuously generates data from different geographical locations has been studied. The authors developed a solution to save the high cost incurring when managing the distributed big data, and they proposed an approximation algorithm by reducing the data placement problem to the minimum cost multicommodity flow problem. Their solution addressed a data placement respecting a fair usage of the cloud services like the quality of service (QoS) requirement of cloud provider while savings computation and bandwidth costs. The solution is closely similar to our context but differs mainly on the conditions and characteristics of data workflow aspects and by no means disclosing intermediate data dependencies. They focused more on maximizing the system throughput in terms of data volume to be placed while saving the computing/storage and communication costs in the distributed datacenters. Sharing intermediate data from computation produced between different workflow MapReduce jobs is studied in [8]. The authors presented a scheduling technique for data-driven jobs sharing opportunities that involves the scan of the input file with the goal of maximizing the likelihood of sharing scans. A similar optimization approach is presented in [9]. A cost model is presented saving processing time and money for MapReduce jobs in order to define an optimization problem that finds an optimal grouping of set of queries and solves it using a dynamic programming approach. These works present a data-driven job scheduling issue which is not exactly the same as the data placement problem. These do not focus on the intermediate data scheduling optimization as well as the incurred storage cost.

Workflow scheduling problems [35, 36] in cloud environments are considered to be very challenging. Many strategies based-heuristic were proposed to solve the tasks scheduling problem without considering the data that are generated by these tasks. Authors in [35], proposed a meta-heuristic approach, called Hybrid GA-PSO (Genetic Algorithm-Particle Swarm Optimization), to solve the workflow tasks scheduling problem. The Hybrid GA-PSO algorithm returns a balanced solution for tasks distribution among different virtual machines in a cloud environment by considering both the total monetary cost and the execution makespan. While the PSO-based algorithm converges quickly to a local optimal solution, this can be far from the global optimal solution. In the same context, authors in [36] proposed a metaheuristic-based algorithm, called Hybrid Bio-inspired Metaheuristic for Multi-objective Optimization (HBMMO), to solve the multiple conflicting objectives optimization problem. The authors considered in their optimization some important requirements of the users or the providers, such as makespan, cost, and load balancing among virtual machines. The proposed HBMMO method optimizes the scheduling of tasks workflow in the cloud environment by considering a non-dominant sorting strategy which is a hybridization of the list-based heuristic algorithm PEFT (Predict Earliest Finish Time) [37] and the discrete version of the metaheuristic algorithm SOS (Symbiotic Organisms Search) [38].

The research works that considered data workflow features are presented in [10, 11, 12, 13, 14]. Nevertheless, the dynamic variation of inter and intra-jobs dependencies from the generated intermediate data was not addressed with the same focus. In [10], an adaptive data-task placement approach is proposed that reflects asynchronous coupling among tasks in order to reduce execution time and data movement overhead. The authors in [11] have dealt with improving the data workflow's execution by clustering the interdependent datasets and distribute them intelligently onto the same datacenters to reduce data transfers. In [14], the authors established a data placement algorithm based on data dependency clustering and recursive partitioning. The aims of the algorithm are to reduce the amount of transmitted data and the time consumption during data-intensive application execution. The pursued strategy is extended with a heuristic to make frequent data movements occuring on high-bandwidth channels of the entire cloud system.

3. System model.

3.1. Cloud storage infrastructure and assumptions. For the intra- and inter-job data workflow placement problem depicted in Fig. 3.1, the objective is to route and store a set of intermediate data considering their dependencies generated by a collaborative tasks[1] from multiple physical sites while saving their opera-

[1]Tasks are launched and executed from an environment where scientific users collaborate and conduct their research together.

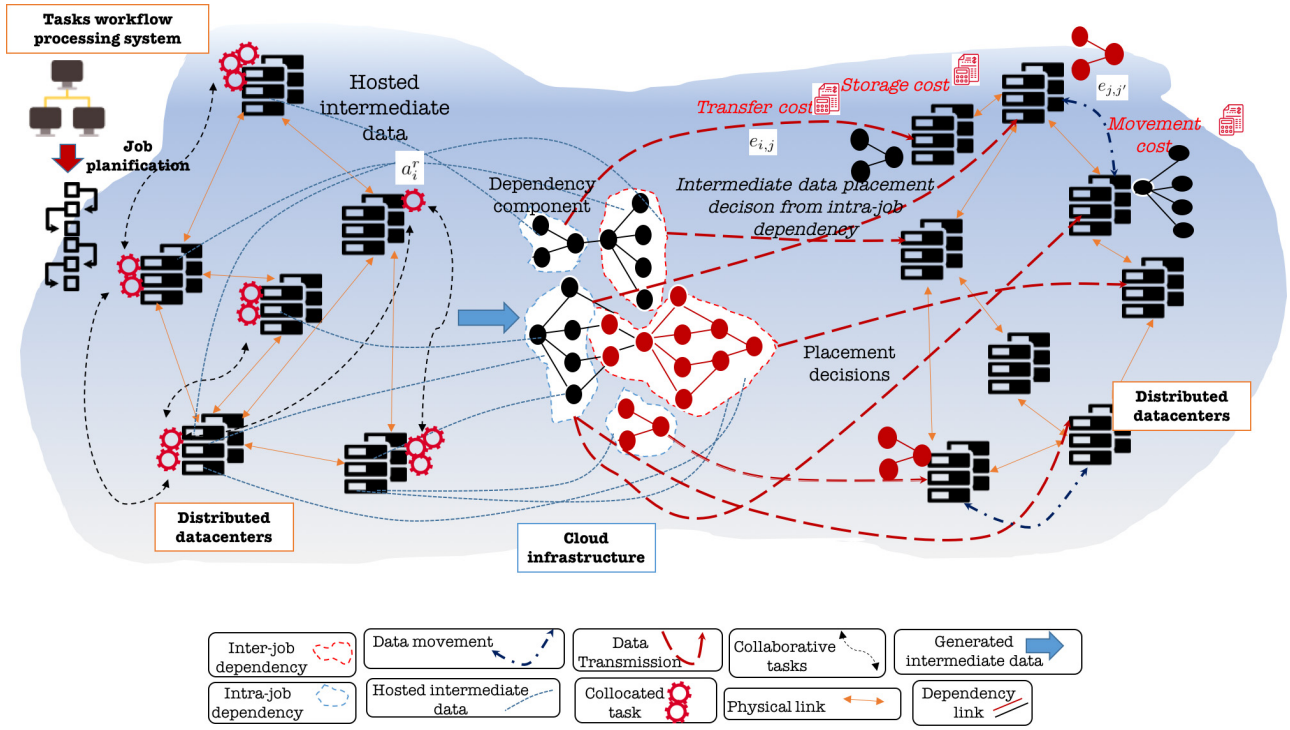

FIG. 3.1. *The system architecture.*

tional costs. Without loss of generality, we assume that the collaborative tasks, which process and generate new intermediate data files, are previously assigned to the cloud infrastructure (model task assignment offered by a cloud infrastructure). Since the intermediate data dependency placements are our most significant concern, we assume that the tasks were assigned to computing nodes following some simple model. The problem of placing the intermediate data files is close to the well-known MCMF problem; an optimization problem described in [15] that involves simultaneously shipping multiple commodities through a single graph, so the total flow obeys the arc capacity constraints by optimizing the cost.

The modeling starts by considering a set of geographically distributed datacenters[2] as a directed graph-based model $G = (DC \cup A, E)$. It forms a cloud infrastructure and constructs a shared computation and storage limited to a set of resources for processing and storing the data workflow. Users, such as enterprises, institutions or researchers, that own and share a cloud infrastructure issued from providers, have an access to the distributed datacenters (DC) to process multiple collaborative tasks into multiple processing phases. The distributed datacenters known as storage containers cohabit with collaborative task A through one or multiple jobs r running in parallel [16]. A set of tasks are collocated on multiple source datacenters, and each task $a_i^r \in A$ from job r is assigned to source datacenter dc_i. Let $\{e_{i,j}, e_{j,j'}\} \in E$ be the intermediate data transfer and movement (initial and dynamic intermediate data routing respectively) links between source datacenter dc_i and destination datacenter dc_j and between destination datacenters dc_j and another destination datacenter $dc_{j'}$, which are geographically interconnected via the Internet. The placement of the intermediate data dependencies to the set of datacenter destinations $dc_j \in DC$ is considered at the beginning of each phase.

3.2. Intermediate data dependency model. Placing intermediate data with the same correlation to a single destination datacenter can significantly decrease the amount of data dependency movements [17]. This leads to consider a vector of all intermediate data files denoted by Φ^M and $|\Phi^M|$ its size, representing the

[2]The security and communication management aspects in a collaborative processing are supposed to be covered by the cloud SLA policy in a cloud environment.

correlations among them that are generated during the workflow phases divided into equal period of time t. These correlations, which reflect the intra- and inter-job dependencies from a set of intermediate data files, are recovered into dependency component $m \in M$. M contains all the different components of dependency that are modeled by a Directed Acyclic Graph (DAG) which takes the advantage of a topology ordering, thus defining relations among nodes [18, 19]. The DAG represents a set of intermediate data files $\phi^m_{a^r_i}(t)$. Let $|\phi^m_{a^r_i}(t)|$ be their respective sizes. These data files have unavoidable complex dependencies that are generated by single task $a^r_i \in A$ from job r at source datacenter dc_i. In DAG, set of files $\phi^m_{a^r_i}(t)$, from the inter-job dependencies, are atomic and must be synchronized for their processing. By contrast, for the intra-job dependencies, these files are deduced from a partial correlation with an asynchronous processing [10]. Let $\phi^m(t)$ and $\phi^m_i(t)$ be the intermediate data of a single dependency component m generated at multiple datacenters and the single home datacenter dc_i respectively and, $|\phi^m(t)|$ and $|\phi^m_i(t)|$ be their respective sizes. At the end of each workflow phase, generated intermediate data file $\phi^m_i(t) \in \Phi^m$ must be outsourced and placed through data transfer link $e_{i,j} \in E$ from datacenter dc_i to dc_j for persistent storing or future reuse [20, 21]. It is important to note that the set of dependency components M and the related type are a predetermined value given by scientific user that can be obtained through the data analysis clustering method [22]. We assume that the intermediate data dependencies clustering is given a priori and is beyond the scope of the present work.

3.3. Capacity and cost model. To come up with an intermediate data dependency placement from the collaborative task workflow execution in cloud datacenters, we take into consideration the fact that all datacenters and network resources are limited [23, 24]. Thus, let S_j be the storage capacity of datacenter destination $dc_j \in DC$, and $W_{i,j}$, $W_{j,j'}$ be the bandwidth capacities of the data file transfer and movement links $e_{i,j}, e_{j,j'} \in E$ respectively. In order to manage and transfer these files, a data bandwidth denoted w_ϕ is assigned for one unit of intermediate data file. During a run-time phase, the available amount of storage capacity in datacenter dc_j, when transferring an amount of intermediate data files $\phi^m_i(t)$, is denoted by $s^{avail}_{i,j}(t)$. Let $w^{avail}_{i,j}(t)$, $w^{avail}_{j,j'}(t)$ be the available capacities of a data transfer and movement of links $e_{i,j}, e_{j,j'} \in E$. In addition, transferring and storing the intermediate data dependencies from source datacenter dc_i into destination datacenter dc_j are facing in both storage resource cost and scale. However, they usually consume high costs in a cloud infrastructure due to an inefficient utilization of the resources [25]. In practice, these resource demands are leading to operational cost specifically for data transfers and storage costs (measured per one unit in GB) that embrace the usage-based pricing policy [9]. Moreover, reused intermediate data dependencies that are not locally stored but remotely served on data demands are led to an additional cost, as movement cost, which is deducted from their migration among datacenter destinations [14]. In fact, these operational storage costs are related to the size of the intermediate data files that are transferred, stored and moved among distributed datacenters according to their correlation during each run-time phase. Moreover, each datacenter destination $dc_j \in DC$ is preserved to the geographical area where it is located [26], thus holding a storage cost noted c_{s_j}. The proportion of intermediate data dependencies ϕ^m of a single dependency component generated from multiple source datacenters and placed separately into different locations dc_j and $dc_{j'}$ are led to a potential dependency movement cost. For clear differentiation from the transfer cost, we assume that the cost of intermediate data movement is proportional to the number of intermediate data dependency files transmitted between datacenter destinations. Therefore, the movement cost is defined as the amount of data moved among two or multiple destination datacenters. Hence, each link $e_{i,j}, e_{j,j'} \in E$ entry faces data bandwidth cost c_{w_ϕ}.

For the sake of easier reading, Table 3.1 summarizes the notations used in the present work.

4. Placement algorithms. From the intermediate data dependency placement issue that reduced to an MCMF problem in G, two variants are materialized. In fact, in the case of intra-job dependency type, the routing of intermediate data dependencies can be performed using multiple links. When this assumption is omitted, i.e. when splittable flow routing can be used, variable of the optimization problem becomes continuous and as a consequence the considered problem becomes easier to solve. In contrast, in the case of inter-job dependency type, the routing of intermediate data dependencies in G cannot be fractionated. Thus, the MCMF problem seems to be hard to solve. For this aim, we formulate the intra-job splittable dependency placement based on a Linear Program (LP) approach, and a heuristic approach is proposed in this section as an approximation algorithm for the intra-job unsplittable dependency placement.

TABLE 3.1
Symbols for the model

Notation	Description
G	The cloud infrastructure (provider)
DC	A set of distributed datacenters in cloud infrastructure G
t	The run-time window which represents the homogeneous discrete time slot from generated collaborative data-tasks workflow processing
A	The set of collaborative tasks in distributed datacenter DC
E	The set of links among distributed datacenter DC
i, j, j'	Indices used to designate distributed datacenters. i belongs to source datacenter dc_i, while j and j' belong to different destination datacenters (dc_j and $dc_{j'}$)
$e_{i,j}$	Data transfer link between source datacenter dc_i and destination datacenter dc_j
r	Workflow job in the system
a_i^r	The collocated task in source datacenter dc_i
dc_i	A source datacenter temporarily storing generated intermediate data from collocated task a_i^r of job $r \in R$
dc_j	A datacenter destination where the intermediate data files are to be placed
M	The set of dependency components in the system including correlation among generated intermediate data
$\phi^m(t)$	The intermediate data files of a single dependency component m generated in multiple datacenters at time slot t, and $\|\phi^m(t)\|$ its size
$\phi_i^m(t)$	The intermediate data files generated in datacenter dc_i from dependency component $m \in M$ at time slot t, and $\|\phi_i^m(t)\|$ its size
$\phi_{a_i^r}^m(t)$	The intermediate data files generated by task a_{i^r} of dependency component m at time slot t, and $\|\phi_{a_i^r}^m(t)\|$ its size
Φ^M	All generated intermediate data files in the system, and $\|\Phi^M\|$ its size
L_ϕ	The vector list of intermediate data of all dependency components $m \in M, m = 1, ..., k$
w_ϕ	The data bandwidth assigned to one unit of intermediate data file $\phi_{a_i^r}^m(t)$
$W_{i,j}$	The data bandwidth capacity of movement link $e_{i,j} \in E$
$w_{i,j}^{avail}(t)$	The available amount of data transfer link $e_{i,j} \in E$ at time slot t
$W_{j,j'}$	A data bandwidth capacity of movement link $e_{j,j'} \in E$
$w_{j,j'}^{avail}(t)$	The available amount of data transfer link $e_{j,j'} \in E$ at time slot t
$s_{i,j}^{avail}(t)$	The available amount of storage space when transferring an amount of intermediate data files from source datacenter dc_i to destination datacenter dc_j at time slot t.
S_j	The data storage capacity of destination datacenter $dc_j \in DC$
$x_{i,j}^m(t)$	A decision variable reflecting the amount of intermediate data flow moving from source datacenter dc_i of dependency component m to destination datacenter $dc_j \in DC$ at time slot t.
$x_{j,j'}^m(t)$	A decision variable reflecting the amount of intermediate data dependency component m moving between destination datacenters $dc_j, dc_{j'} \in DC$ at time slot t
c_{s_j}	The storage cost of one unit of intermediate data in datacenter destination $dc_j \in DC$
c_{w_ϕ}	The data bandwidth cost of one unit of intermediate data
$f(\phi_{a_i^r}^m)$	A dependency component flows in graph G_p
$f(\phi^m)$	All flows from a single dependency component in graph G_p
ShP_ϕ	The shortest path from s_{source} to s_{sink} in G_p

4.1. Exact algorithm. This section presents an exact analytical algorithm for splittable variant of the intermediate data dependency placement problem from multiple datacenters in cloud infrastructure G. The exact algorithm is an LP model with the inclusion of valid conditions expressed in the form of constraints or inequalities. Through the constraints of the problem, the intermediate data placement in a directed graph $G = (DC \cup A, E)$ at time slot t is to route and place intermediate data dependencies $\phi^m(t) \in \Phi^M$ that are considered as continuous commodity flows of dependency component m from multiple source datacenters to one or multiple destination datacenters while saving their transfer, storage and movement costs. A number of decision variables and valid inequalities (as listed for convenience in Table 3.1) are thus defined as follows:

1) Decision variables: Let $x^m_{i,j}(t) \in \mathbb{R}$ be the intermediate data of one dependency component m standing for the amount of intermediate data dependency flows transferring from source datacenter dc_i at time slot t to destination datacenter dc_j at time slot $t+1$ on link $e_{i,j} \in G$. In order to take into account the amount of intermediate data dependencies that are moved among different destination datacenters dc_j, $dc_{j'}$, we add variable $x^m_{j,j'}(t) \in \mathbb{R}$.

2) Flow conservation constraint: One typical constraint or requirement is to ensure that at all time, every flow through directed graph G is physically possible. First, we enforce flow continuity by making sure that the sum of intermediate data dependency flows leaving from source datacenter dc_i at time slot $t-1$ is equal to $\phi^m_i(t)$ which is the sum of flows arriving from the same datacenter dc_i that are considering the same dependency component m at time slot t. Formally:

$$\sum_{j \in DC} x^m_{i,j}(t) - \sum_{j \in DC} x^m_{j,i}(t-1) = \phi^m_i(t) \quad \forall m, t, i. \tag{4.1}$$

3) Capacity constraint of intermediate data flows: Each intermediate data dependency flow $x^m_{i,j}(t)$ may have its own individual capacity constraint which represents a lower bound on dependency component commodity m through link $e_{i,j}$. This ensures the atomicity of lower bound $\phi^m_{a^r_i}(t)$ on $x^m_{i,j}(t)$ of which all these flows take a same link $e_{i,j}$, hence:

$$0 \leq \phi^m_{a^r_i}(t) \leq x^m_{i,j}(t) \quad \forall i, j, a^r_i, m, t. \tag{4.2}$$

4) Capacity constraint of data transfer links: In G, each link $e_{i,j}$ may have a capacity constraint like the data bandwidth routing constraint. Equation (4.3) ensures that the routing of aggregate intermediate data dependencies is limited by the available amount of data bandwidth allocated on link $e_{i,j}$ at time slot t:

$$\sum_{m \in M} w_\phi \cdot |\phi^m_i(t)| \cdot x^m_{i,j}(t) \leq w^{avail}_{i,j}(t) \quad \forall i, j, t. \tag{4.3}$$

Additionally, link $e_{i,j}$ is bounded by the data bandwidth capacity at all system execution time, hence:

$$\sum_{m \in M} \sum_{t \in T} w_\phi \cdot |\phi^m_i(t)| \cdot x^m_{i,j}(t) \leq W_{i,j} \quad \forall i, j. \tag{4.4}$$

5) Capacity constraint of data movement links: In G, each link $e_{j,j'}$ may have a capacity constraint like the data bandwidth routing constraint. Equation (4.5) ensures that moving intermediate data dependencies is limited by the available amount of data bandwidth allocated on link $e_{j,j'}$ at time slot t:

$$\sum_{m \in M} \sum_{a^r_i \in A} w_\phi \cdot |\phi^m(t) - \phi^m_{a^r_i}(t)| \cdot x^m_{j,j'}(t) \leq w^{avail}_{j,j'}(t) \quad \forall j, j', t. \tag{4.5}$$

Additionally, the link $e_{j,j'}$ is bounded by the data bandwidth capacity at all system execution time, hence:

$$\sum_{m \in M} \sum_{a^r_i \in A} \sum_{t \in T} w_\phi \cdot |\phi^m(t) - \phi^m_{a^r_i}(t)| \cdot x^m_{j,j'}(t) \leq W_{j,j'} \quad \forall j, j'. \tag{4.6}$$

6) Capacity constraint of dependency component: A uniqueness constraint is used to ensure that the routed intermediate data dependency flows do not exceed the dependency corresponding component capacity. Formally:

$$\sum_{i \in DC} x_{i,j}^{m}(t) \leq \phi_{i}^{m}(t) \quad \forall j, m, t. \tag{4.7}$$

7) Storage capacity constraint: Each destination datacenter has a limited amount of storage space available to share across all the intermediate data placement demands. This allows to host only a limited amount of intermediate data dependencies from source datacenter dc_i to destination datacenter dc_j. Formally:

$$\sum_{m \in M} |\phi_{i}^{m}(t)| \cdot x_{i,j}^{m}(t) \leq s_{i,j}^{avail}(t) \quad \forall i, j, t. \tag{4.8}$$

For any intermediate data placement demands, the data routing must not exceed the total storage capacity at all system execution time. Formally:

$$\sum_{m \in M} \sum_{t \in T} |\phi_{i}^{m}(t)| \cdot x_{i,j}^{m}(t) \leq S_{j} \quad \forall i, j. \tag{4.9}$$

8) Balancing constraint: Since the collaborative tasks in the workflow processing generate the intermediate data dependencies in multiple phases, these latter may vary over time in the distributed datacenter environment. In other words, the flow sequence of generated intermediate data dependencies changes as commodity changes. Thus, the flows among the distributed datacenters must be balanced. Hence, source and sink nodes s_{source} and s_{sink} are respectively introduced in graph G. Source node s_{source} is connected to every source datacenter dc_i, and sink node s_{sink} is connected to every destination datacenter dc_j. Source and sink nodes are also subject to a constraint that enforces all the intermediate data dependency flows starting on s_{source} to ending at s_{sink}. Formally:

$$\sum_{i \in DC} x_{s_{source},i}^{m} = \sum_{j \in DC} x_{j,s_{sink}}^{m} \quad \forall m \in M \tag{4.10}$$

9) Data transfer cost: Equation (4.11) denotes the data transfer cost on link $e_{i,j}$ which intermediate data dependency flows are routed.

$$C(w_{i,j}) = \sum_{i \in DC} \sum_{j \in DC} \sum_{m \in M} \sum_{t \in T} |\phi_{i}^{m}(t)| \cdot x_{i,j}^{m}(t) \cdot w_{\phi} \cdot c_{w_{\phi}} \tag{4.11}$$

10) Storage cost: Equation (4.12) denotes the storage cost of destination datacenter dc_j which intermediate data dependency flows are routed. Formally:

$$C(s_{j}) = \sum_{i \in DC} \sum_{j \in DC} \sum_{m \in M} \sum_{t \in T} |\phi_{i}^{m}(t)| \cdot x_{i,j}^{m}(t) \cdot c_{s_{j}} \tag{4.12}$$

11) Data movement cost: The proportions of intermediate data ϕ^m from one dependency component that are stored separately into different locations dc_j and $dc_{j'}$ are led to potential intermediate data dependency movement cost. With no loss of generality, it is assumed here that the amount of intermediate data that moves from dc_j to $dc_{j'}$ is defined as the set of intermediate data of a single dependency component m that is fractionated from the set of atomic $\phi_{a_i^r}^{m}(t)$. Formally:

$$C(w_{j,j'}) = \sum_{i \in DC} \sum_{j,j' \in DC}^{j \neq j'} \sum_{m \in M} \sum_{t \in T} |\phi^{m}(t) - \phi_{a_i^r}^{m}(t)| \cdot x_{j,j'}^{m}(t) \cdot w_{\phi} \cdot c_{w_{\phi}} \tag{4.13}$$

12) Objective function: The objective of the intermediate data placement problem is to find, for a given set of dependency flows $x^m_{i,j}(t)$, a set of destination datacenters that can place them to minimize the aggregate cost of transferring, storing and moving intermediate data dependencies. This can be expressed using the following expression:

$$Minimize \ \ (C(w_{i,j}) + C(s_j) + C(w_{j,j'})) \tag{4.14}$$

Under the formulation listed above, the LP model is polynomial. However, the optimization is carried out with respect to flows $x^m_{i,j}(t)$ that are bounded and constrained as a result of the amount of intermediate data dependencies $\phi^m_{a^r_i}(t)$ generated by a single task a^r_i. This converges the exact algorithm into a non-polynomial time regarding to size $|\phi^m_{a^r_i}(t)|$ on very large instances when the splitting of flows $x^m_{i,j}(t)$ becomes marginal. Since a dependency component cannot start before the intermediate data dependencies of their predecessors are materialized, the unsplittable version of the intermediate data placement problem considering all flows for each dependency component from inter-job must be sent along a single link, making the problem NP-hard [15]. Due to the intractability of the problem, a heuristic is presented to address larger scale instances in a reasonable time.

4.2. Heuristic approach. The intra-job dependency placement solution is compared to the solution of the exact approach and requires the placement of the amount of intermediate data dependencies into a single destination datacenter. Thus, a naive greedy solution considers an integer commodity of dependency component m from different sources as a single source flow unlike the exact approach that tolerates multiple source of dependency component m independently when solving the problem. Under the unsplittable solution, a commodity is never split along multiple paths during the placement decision. Furthermore, the greedy approach applies a routine procedure in specific graph G_p, and assume that the minimum demands are less than or equal to the maximum capacity of the nodes in graph G_p [15]. The latter involving less connection, the local search of the optimum on a specific optimized graph that reduces the search space accelerates the execution time of greedy solutions.

4.2.1. The greedy optimization framework. The basic idea behind the proposed framework is to reduce the problem to a minimum cost unsplittable multicommodity flow problem with multiple dependency component sources in specific directed flow network graph $G_p = (DC_p \cup A_p; E_p; u; c)$, and deals with a cost function c: $E \rightarrow R$ and capacity function u: $E \rightarrow R$

The first part of the construction of the network flow graph G_p concerns the assignment of the input flows from multiple sources. For each collocated task $a^r_i \in A$ that generates intermediate data $\phi^m_{a^r_i}(t)$ in the same source datacenter dc_i, there is a virtual source datacenter node $dc_i(\phi^m_{a^r_i})_p$ in DC_p. For all generated intermediate data $\phi^m(t)$ from multiple collocated tasks belonging to the same dependency component $m \in M$, there is a virtual dependency source datacenter node $dc_i(\phi^m)_p$ representing those intermediate data dependencies for different tasks. For all generated intermediate data dependency components $\phi^m(t)$ hosted in a multiple source datacenter in G, there is a virtual dependency component node $dc(\phi^m)_p$ which corresponds to a virtual location of distributed source datacenter $dc_i(\phi^m)_p$ hosting intermediate data of dependency component $\phi^m(t)$. The $dc_i(\phi^m_{a^r_i})_p$, $dc_i(\phi^m)_p$ and $dc(\phi^m)_p$ are added in graph G_p.

In network flow graph G_p, a virtual source node s_{source} is added and represents the source of all intermediate data dependencies $\sum_{m \in M} \sum_{a^r_i \in A} \phi^m_{a^r_i}(t)$ hosted in the different virtual source datacenter nodes $dc_i(\phi^m_{a^r_i})_p$. Source node s_{source} is connected with a link $(s_{source}, dc_i(\phi^m_{a^r_i})_p)$ in E_p to each $dc_i(\phi^m_{a^r_i})_p$. Also, from this latter to $dc_i(\phi^m)_p$ represented by link $(dc_i(\phi^m_{a^r_i}))_p, dc_i(\phi^m)_p)$, involving cost $c(s_{source}, dc_i(\phi^m_{a^r_i})_p) = 0$, as well as a link capacity demand that is assigned as the set of intermediate data dependencies $\phi^m_{a^r_i}(t)$ generated from each collocated task in the source datacenter at time slot t, i.e:

$$u(s_{source}, dc_i(\phi^m_{a^r_i})_p) = u(dc_i(\phi^m_{a^r_i})_p, dc_i(\phi^m)_p) = |\phi^m_{a^r_i}(t)|. \tag{4.15}$$

A link $(dc_i(\phi^m)_p, dc(\phi^m)_p)$ is added to G_p from each virtual dependency source datacenter node $dc_i(\phi^m)_p$ to the corresponding virtual dependency component node $dc(\phi^m)_p$ within the same dependency component

$m \in M$. The corresponding cost is $c(dc_i(\phi^m)_p, dc(\phi^m)_p) = 0$, and the capacity is an amount of dependency component from all source datacenters that temporarily store them, i.e:

$$u(dc_i(\phi^m)_p, dc(\phi^m)_p) = \sum_{a_i^r \in A} |\phi_{a_i^r}^m(t)| = |\phi_i^m(t)|. \tag{4.16}$$

The second part of the optimization framework deals with the identification of potential links for routing intermediate data dependencies to the destination datacenter. For each destination datacenter dc_j in G, there is a virtual destination datacenter node dc_{j_p} which hosts all intermediate data dependencies for one or multiple dependency components ϕ^m. Each virtual destination datacenter node dc_{j_p} is added to G_p. The obvious no-bottleneck assumption which was made throughout an unsplittable version of the greedy optimization framework is that a virtual destination datacenter node dc_{j_p} in a network flow graph G_p has enough capacity to satisfy all dependency components ϕ^m individually, but not necessarily all commodities. Thus, in graph G, destination datacenters that do not have available storage capacity to accommodate each dependency component are excluded from G_p. Hence, from each virtual dependency component node $dc(\phi^m)_p$ there is a link $(dc(\phi^m)_p, dc_{j_p})$ to each destination datacenter dc_{j_p} that is added to graph G_p. All these links are connected to each virtual destination datacenter node dc_{j_p} that satisfies the placement of an integer dependency component ϕ^m. A positive cost $c(dc(\phi^m)_p, dc_{j_p})$ is assigned along a link $(dc(\phi^m)_p, dc_{j_p})$ from the virtual dependency component to the destination datacenter node. The corresponding total storage cost represents the sum of the data transfer cost $c_{w_\phi}(dc(\phi^m)_p, dc_{j_p})$ and the storage cost $c_{s_j}(dc(\phi^m)_p, dc_{j_p})$ to host one unit of intermediate data dependency $\phi_{a_i^r}^m$, i.e:

$$c(dc(\phi^m)_p, dc_{j_p}) = c_{w_\phi}(dc(\phi^m)_p, dc_{j_p}) + c_{s_j}(dc(\phi^m)_p, dc_{j_p}). \tag{4.17}$$

In addition to the cost of a virtual link $(dc(\phi^m)_p, dc_{j_p})$, a capacity $u(dc(\phi^m)_p, dc_{j_p})$ is assigned, which is the amount of intermediate data $\phi_{a_i^r}^m(t)$ that can be routed along a virtual link with an available bandwidth capacity upon routing integer dependency component ϕ^m. The capacity of the bandwidth is shared between each routing unit of a dependency component at time slot t. Since, the storage capacity constraint is raised when a link $(dc(\phi^m)_p, dc_{j_p})$ is created in graph G_p, the routing of the intermediate data dependency component $\phi^m(t)$ considers only the available amount of a data bandwidth $c_t(dc(\phi^m)_p, dc_{j_p})$ on each corresponding link $(dc(\phi^m)_p, dc_{j_p})$ to different virtual destination datacenters dc_{j_p} at time slot t , i.e:

$$u(dc(\phi^m)_p, dc_{j_p}) = \frac{w^{avail}_{dc(\phi^m)_p, j}(t)}{w_\phi \cdot |\phi_{a_i^r}^m(t)|}. \tag{4.18}$$

A virtual destination node s_{sink} is finally added to a flow graph G_p from each virtual destination datacenter node dc_{j_p}. A virtual link (dc_{j_p}, s_{sink}) is added between them. A zero cost is assigned to each virtual movement link (dc_{j_p}, s_{sink}). A capacity $u(dc_{j_p}, s_{sink})$ for each link (dc_{j_p}, s_{sink}) is the available amount of storage space in each one upon storing an integer dependency component ϕ^m at time slot t, i.e:

$$u(dc_{j_p}, s_{sink}) = s^{avail}_{dc_{j_p}}(t) - |\phi^m(t)| \tag{4.19}$$

Figure 4.1 shows the representation of the generated directed flow graph $G_p = (DC_p \cup A_p; E_p; u; c)$

4.2.2. Greedy heuristic algorithm. A greedy heuristic algorithm has been developed for the minimum cost inter-job intermediate data dependency placement problem through the reduction to the minimum cost of unsplittable multicommodity flow with multiple dependency component sources in flow graph G_p.

Let $S_{dc_j,min}$ be the minimum storage capacity of a destination datacenter dc_{j_p} on a network flow graph G_p, and ϕ^m_{max} the largest dependency component generated from virtual source datacenter node $dc(\phi^m)_p$. As storage resources are scalable in a flow graph G_p acting as a cloud environment, it is realistic to assume that $|\phi^m_{max}| \leq S_{dc_j,min}$ from the construction of the flow graph G_p. Since the splittable exact algorithm is a relaxation of the unsplittable heuristic algorithm, a feasible solution is assumed for the splittable exact algorithm which is fractional feasible flow f_0 that satisfies all demands of dependency component ϕ^m. Since all dependency

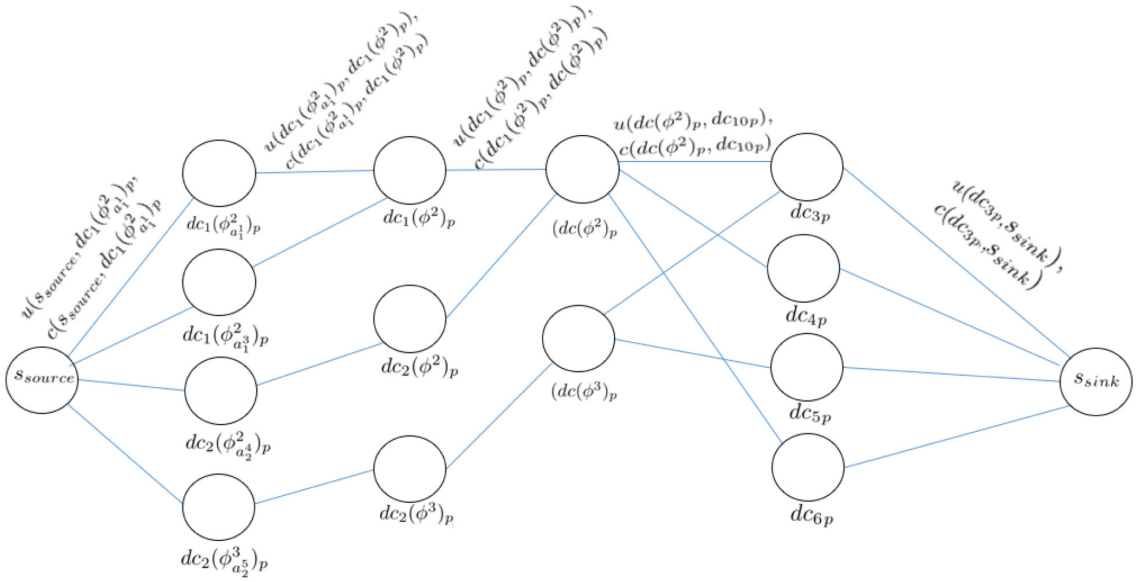

FIG. 4.1. *The generated directed flow graph* $G_p = (DC_p \cup A_p; E_p; u; c)$.

components are known a priori, so is their generation order. Hence, the greedy heuristic algorithm adopts an orderly greedy method and starts with the initial placement and works in steps. At the end of each step, it outputs a set of destination datacenters and transfers intermediate data dependencies to that destination datacenters, considering a minimum transfer and storage cost. As the greedy algorithm gives sequential placement solutions, there is no congestion problem on the different dependency components sharing links. Therefore, the greedy algorithm just takes care of the integer dependency component placement (bandwidth capacity is shared between the flows of a single dependency component at time slot t) to their destination. The greedy heuristic algorithm execution on a network flow graph G_p is provided in the following steps:

Step 1. Let $f(\phi^m)$ be the dependency component flow for all dependency intermediate data-task flows $\sum_{a^r_i \in A} f(\phi^m_{a^r_i})$ with the minimum total storage cost from s_{source} to s_{sink}. Flows $f(\phi^m)$ route dependency component commodities $\phi^m_{a^r_i}$ from different virtual source datacenter nodes connected from source node s_{source} to their destination datacenter nodes $dc_j(\phi^m)_p$, the latter being connected with destination node s_{sink}. A set of dependency component commodities $\sum_{m \in M} \phi^m$ are routed to s_{sink} in graph G_p according to the ascending order of their respective size as dependency component demands: $|\phi^1|$, $|\phi^2|$, ..., $|\phi^k|$, with $\phi^1 \geq \phi^2 \geq \phi^3 \geq ... \geq \phi^k$. Let $L_\phi = (\phi^1, \phi^2, ..., \phi^k)$ be the dependency component list.

Step 2. Start with the first dependency component by selecting it from list L_ϕ. The algorithm scans each dependency component value $\phi^m_{a^r_i}(t)$ in G_p to find the possible path which routes the selected dependency component flow $f(\phi^m)$ along each link $(dc(\phi^m)_p, dc_{j_p})$ in G_p that satisfies the flow conservation in G_p i.e from any nodes dc_p, $dc(0)_p \in DC_p \setminus \{s_{source}, s_{sink}\}$, there is $\sum_{dc(0)_p \in DC_p} f(dc_p, dc(0)_p) = \sum_{dc(0)_p \in DC_p} f(dc(0)_p, dc_p)$.

Step 3. For each solution of dependency component flow $f(\phi^m)$, find the shortest path noted ShP_ϕ from s_{source} to s_{sink} in G_p according to the total minimum storage cost, i.e., $c(ShP_\phi) = \min_{(dc(\phi^m)_p, dc_{j_p}) \in E_p} c(dc(\phi^m)_p, dc_{j_p})$. Once the shortest path ShP_ϕ is found, set $f(ShP_\phi) = \phi^m$ and delete iteratively its flow value $f(\phi^m_{a^r_i})$. Define residual capacity $u_{res}(s_{source}, s_{sink})$ from s_{source} to s_{sink} in order to decrease the routed flows in graph G_p, i.e., $u_{res}(s_{source}, s_{sink}) = u(s_{source}, s_{sink})$ - $f(ShP_\phi)$. Delete the routed dependency component ϕ^m from list L_ϕ and repeat the sub-procedure of step 2 until all flow values $f(\phi^m_{a^r_i})$ are scanned.

Step 4. Repeat the sub-procedure of step 3 until $L_\phi \longleftarrow \varnothing$ and carry the largest flow values iteratively. Then, restore these shortest paths including the optimal cost and denote for each ShP_ϕ the pair $< \phi^m, dc_j >$

corresponding to graph G.

4.2.3. Time complexity. To build a network flow graph G_p for the greedy framework optimization, two steps are needed. The first one consists in assigning each source datacenter $dc_j \in DC$ hosting intermediate data to its dependency component node $m \in M$, and the second one to each destination datacenter $dc_j \in DC$ which is capable of accommodating. The construction of G_p takes $O(M + |DC|)$ for the first step and $O(M^2 + |DC|^2)$ for the second one. Finally, we analyze the time complexity of the greedy solution which considers mostly the shortest path computing step and the sorting of the list of dependency components. The worst complexity of the sorting computation has a fundamental requirement of $O(M^2)$. The shortest path computation step is more complex and requires computing the distance between all intermediate data dependency components and datacenter destinations. This leads to $O(M^2 \times |DC|^2)$ time complexity to consider all combinations or couples. In summary, the average computational complexity of the proposed greedy heuristic algorithm is $O(2M^2 + |DC|^2 \times (M^2 + 1) + M + |DC|)$ in the worst case.

5. Performance evaluation. This section gives an overview of the simulation, evaluation conditions and settings of the proposed algorithms. A dedicated simulation program has been developed to conduct the performance assessments of the heuristic and compare it with the exact algorithm, random and uniform strategies, named random heuristic and uniform heuristic respectively. The random heuristic strategy randomly selects a datacenter to host the intermediate data until its capacity is exhausted and then selects another one as in default Hadoop scheduler [30] (random capacities and random costs). The uniform heuristic strategy is based on the uniform storage capacity of the distributed datacenter upon intermediate data dependency placement decision (balanced capacities and variable costs). This data placement strategy excludes the storage requirements as in [27, 28, 29, 11]. Subsequently, the performance evaluation overall intends to present relevant comparisons between the solutions found by the greedy heuristic algorithm with the optimal ones found by the exact algorithm in terms of performance metrics like optimality, scalability and convergence time.

5.1. Simulation environment. The heuristic is evaluated through a C++ language implementation. The exact algorithm is implemented with IBM ILOG AMPL and solved optimally using CPLEX. The objective of a numerical evaluation is to quantify the amount of total storage cost saving (objective function) that can be expected when routing intermediate data dependencies through cloud storage infrastructures using the greedy heuristic and exact algorithms. The evaluation also reflects particularly the influence of the number of datacenters, the amount of the routed intermediate data and the dependency parameters on the performance metrics.

The assessment scenarios correspond to a cloud infrastructure consisting of 50 distributed datacenters including source and destination datacenters which are connected to each other randomly. We run the simulation program for 20 random tasks, each one including an amount of a random intermediate data generated per one hour time slot in random adjacency matrix-based DAG, each one having a size ranging from 10 GB to 100 GB [12], including their dependencies that are generated randomly as correlation links in DAG from input to output intermediate data. The latter are assigned randomly to the set of source datacenters in charge of temporarily storing them.

The intra-job dependency is described by a dependency parameter value α generated randomly from range [0, 1] and belonging to each intermediate data-task in the DAG. Value 1 corresponds to a splitting rate of an intermediate data file (a fraction of 1 GB splitting for each file from partial correlation), and the opposite case is represented by value 0. A dependency parameter value β is given also that is generated randomly from range [1, 20] which represents the number of clusters randomly grouping intermediate data-tasks. For the inter-job dependency, we set the value of α to 0 from full correlation coupled with dependency parameter value β (the case of inter-job dependency is intrinsically related to the intra-job dependency case when the α value of the latter converges to 0 and has the same dependency value β). On all the carried out experiments, the case when $\alpha = 1$ and $\beta = 20$ are excluded which means that the intermediate data are completely independent. The same dependency parameter value β is assigned to both intra- and inter-job dependencies according to each experiment.

The storage space capacity is considered for the datacenters as randomly set from range [10 GB, 1000 GB] [12]. The transfer link capacity of one unit of intermediate data transmission between distributed datacenters

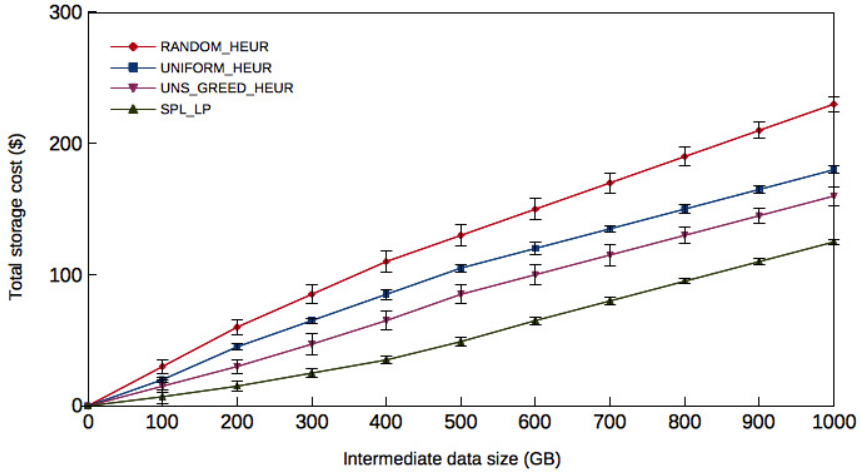

FIG. 5.1. *The total storage cost of algorithms exact, greedy, random and uniform heuristics while varying the intermediate data size when the number of datacenters is set to 50.*

are randomly drawn from range [1, 10] Gbps [7] with a random transfer cost (in $) ranging from 0 to 0.09. Both storage and transaction costs (in $) of one unit of intermediate data dependency are set within [0.02, 0.04] and [0, 0.09] respectively, in relation to the typical charges in Amazon S3 [3].

5.2. Simulation results. To present the performance of the proposed algorithms regarding their effectiveness against comparison strategies solutions, we study the optimality of the exact and greedy heuristic algorithms in terms of the total storage cost ratio, and the results of the scalability and the convergence are reported below.

5.2.1. Impact of the amount of routed intermediate data on algorithms performance. For the specific needs of the simulation, a variation of the amount of intermediate data must be placed from 100 to 1000 GB with an increment of 100 while the number of datacenters DC is set to 50.

To continue to appropriately analyze the simulation, we reflect the concerns of dependency parameter values on the algorithms performance. In this case, each solution found from the execution algorithms is a mean of the results obtained by varying dependency parameters α and β from range [0, 1] and [1, 20] respectively.

Figure 5.1 depicts the curves of total storage cost delivered by the proposed algorithms and the two other strategies. The figure shows that both greedy heuristic and exact algorithms outperform random heuristic and uniform heuristic strategies in terms of cost. The optimal result obtained by the exact solution reaches a cost of $125 when the amount of placed intermediate data achieves 1000 GB, and the greedy heuristic algorithm achieves a nearly optimal storage cost of $160, which is lower than the costs of the random heuristic and uniform heuristic algorithms (43% and 12% respectively). Clearly, the gap between greedy heuristic and uniform heuristic algorithms is very small since the uniform heuristic is independent of the capacity of the cloud infrastructure, so the cost within the datacenters contrast on the placement decision.

Figure 5.2 depicts the curves of the total storage costs of the algorithms by increasing the simulation time. In this instance, the obtained result of the total storage cost is the aggregation of the previously calculated costs during the same simulation (continuous placement). In addition, the simulation test is conducted for 48h in order to validate the need of the greedy heuristic algorithm and to estimate the probability to have good solutions. The lengthening of simulation at time slot 48 while the number of datacenters DC is set to 50 makes the total storage cost of algorithms greedy heuristic, exact, random heuristic and uniform heuristic to $4900, $3300, $7000 and $5400 respectively. Typically, this means that the cost of greedy heuristic is 10% and 42% less than those of uniform heuristic and random heuristic algorithms, while the result of the exact algorithm

[3] https://aws.amazon.com/fr/s3/pricing/

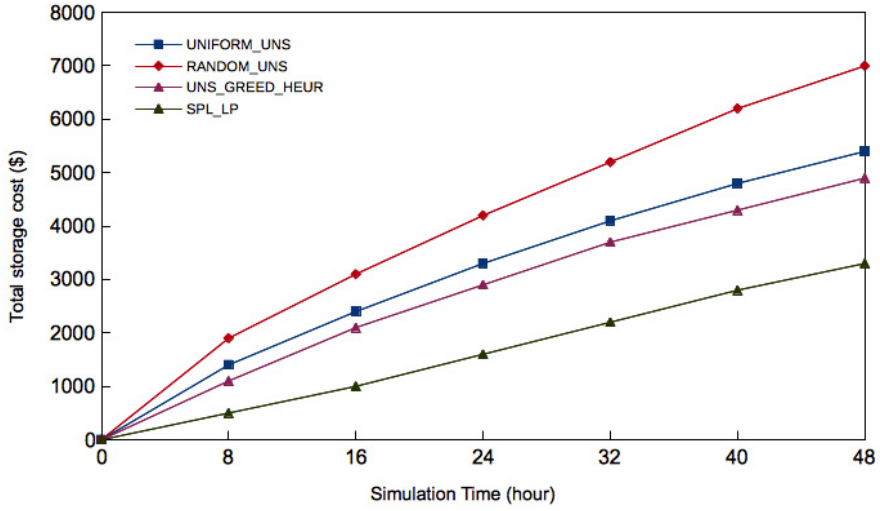

FIG. 5.2. *Total storage cost of algorithm exact, greedy, random and uniform heuristics when the simulation time is extended to 48h, while the number of datacenters is set to 50.*

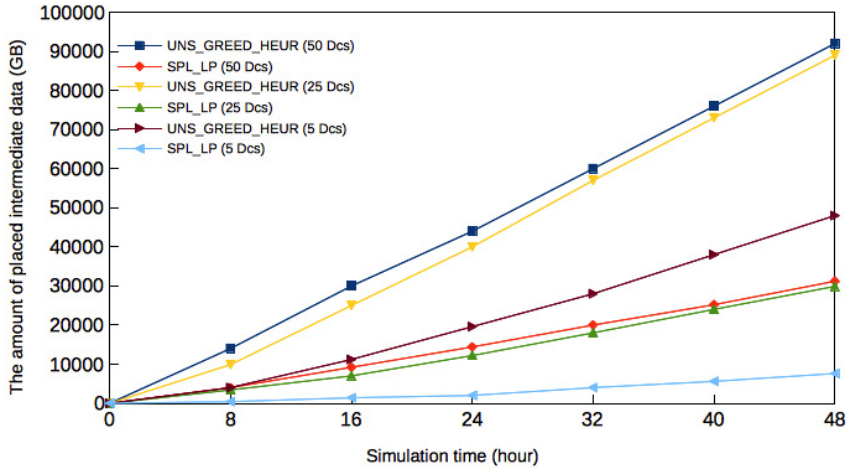

FIG. 5.3. *The amount of intermediate data accumulated per time slot for the proposed algorithms while the number of datacenter ranges from 5 to 50.*

as expected remains the best total storage cost. These results show that the uniform capacity constraint on the intermediate data growth directly affects their placement cost as in the case of the uniform heuristic algorithm. In the case of random heuristic algorithm, some datacenters offering the lowest cost are not involved unintentionally (random selection) or of the causes of inability to host data.

The aim of the performed simulation as depicted in Fig. 5.3 is to quantify the amount of intermediate data placed continuously by varying the number of datacenters from 5 to 50 according to capacities. The accumulated intermediate data placement increases with the increase of the number of datacenters for both algorithms, and begins to be stable when cloud infrastructure handles 25 datacenters. Moreover, the amount of intermediate data accumulated by greedy heuristic algorithm is very important over all variations of datacenter instances, more importantly, from 25 to 50, due to the abundance of the cloud infrastructure capacity, i.e. more intermediate data can be placed in the cloud infrastructure. The decline of intermediate data placement from 25 to 50 is due to the fact that the intermediate data routing is limited by data bandwidth $w_{i,j}^{avail}(t)$ and $w_{j,j'}^{avail}(t)$

since the bandwidth capacity is shared by all dependency components in the exact algorithm. In contrast, in the greedy heuristic algorithm, the data bandwidth capacity is shared by a single dependency component. However, in the exact algorithm, the amount of placed intermediate data is less in comparison with the greedy solution, because it expands the search space for the exact solution since it is based on the simplex method. Thus, the greedy heuristic algorithm responds well to large datacenter instances for which the exact algorithm has more difficulty to find solutions.

5.2.2. Impact of dependency parameters on the algorithms performance. This section studies the impact of the dependency parameters α and β on the algorithms performance in terms of optimal cost. Since the behavior of the proposed execution algorithms regarding dependency type that are processed are different, a need of a variation of quantitative values is experienced for achieving a useful analysis and allowing optimal cost to the unsplittable placement solutions to be more efficiently identified. On the one hand, interval values are considered to align the types of dependency. On the other hand, values are considered when dependency types diverge. In the following, the assessment scenarios correspond to varying dependency parameters (α, β) pair values. Then, simulation results correspond to pair ranges from (α, β)= (0.1, 18), (0.3, 14), (0.5, 10), (0.7, 6), (0.9, 2). These results are reported on figures below keeping the value of α to 0 and with the same dependency values β for the greedy heuristic algorithm while the number of datacenters is set to 50.

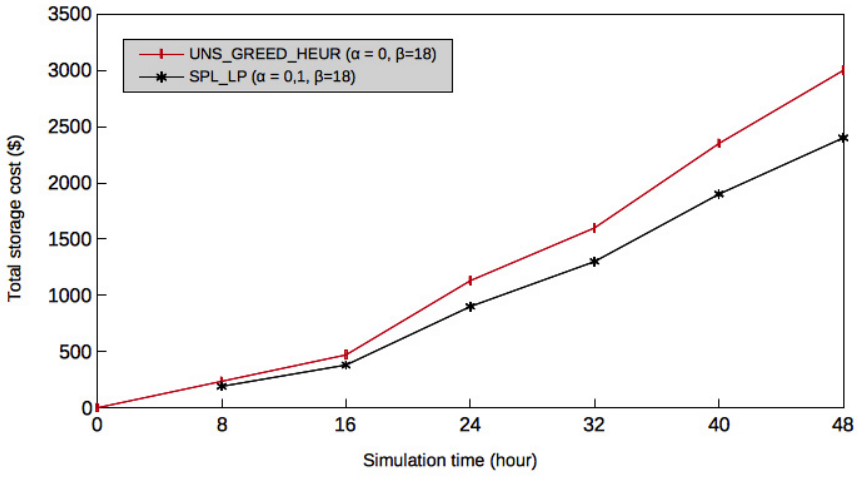

FIG. 5.4. *Greedy heuristic versus exact solutions for the total storage cost when $\alpha = 0.1$ and $\beta = 18$.*

Figure 5.4 depicts the best optimal cost achieved by the objective function for exact and heuristic solutions. The greedy heuristic algorithm performs very well close to the optimal one and achieves a cost of \$3000 at time slot 48 that is near to the optimal result of \$2400 for the exact algorithm. This is due to the fact that the behavior of the two algorithms have to deal with the aligned correlation $\alpha = 0.1$ (practically, no amount of intermediate data is splitted with exact, and 0 defaults to the heuristic) with $\beta = 18$. Therefore, this has a direct impact on the reduction of the transfer, storage and movement costs for both algorithms.

Figures 5.5 and 5.6 depict the second best-case results for the total storage cost which does not exceed \$2800, \$3000 for the exact algorithm, and \$4000, \$4500 for the greedy heuristic algorithm at time slot 48. Since, the amount of the dependency movements are marginal to half ($\alpha = (0.3, 0.5)$) in the exact algorithm, the movement cost is reduced, which reflects the total storage cost. In addition, the heuristic algorithm processes less intermediate data dependencies (10 to 14 clusters). Therefore, it has more chance to find datacenters that have the capacity to allocate those clusters and at the same time offer a better cost.

Fig. 5.7 shows the case when the amount of dependency movements increase more than half ($\alpha = 0.7$) with the growth of the intermediate data dependency volume ($\beta = 6$). This gives a total storage cost of \$3800 and \$5800 respectively for exact and greedy heuristic algorithms. It can be seen that the total storage cost of the

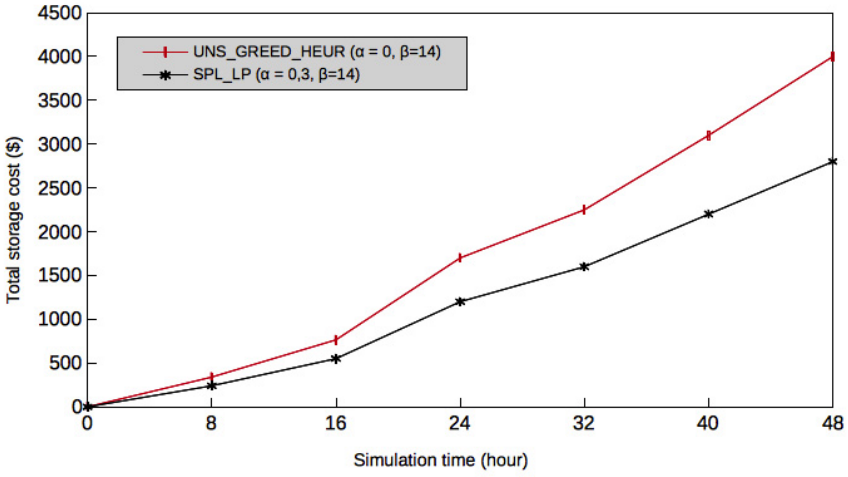

FIG. 5.5. *Greedy heuristic versus exact solutions for the total storage cost when $\alpha = 0.3$ and $\beta = 14$.*

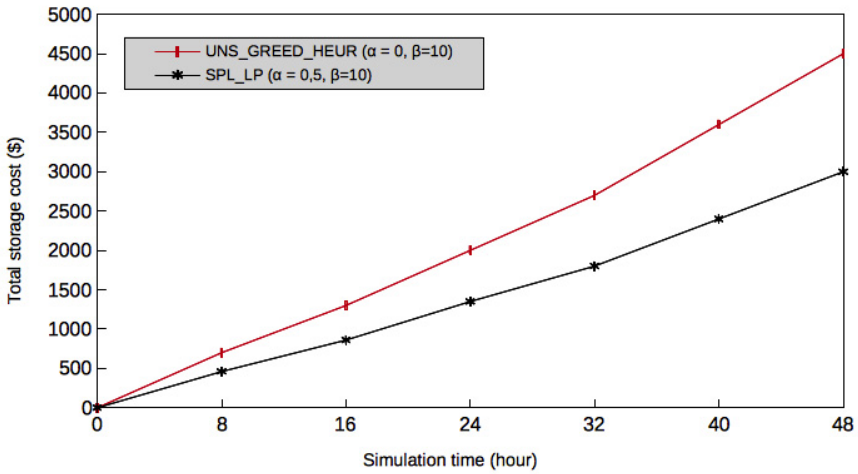

FIG. 5.6. *Greedy heuristic versus exact solutions for the total storage cost when $\alpha = 0.5$ and $\beta = 10$.*

two algorithms dependents on the amount of intermediate data dependencies. This validates our analysis for the results discussed above (Fig. 5.4, 5.5 and 5.6).

Fig. 5.8 shows the worst case when the highest total storage cost is found for both algorithms. The total storage cost reaches \$4200 and \$7200 for exact and greedy heuristic algorithms respectively at time slot 48 since the amount of intermediate data dependencies that transit between destination datacenters are significant ($\alpha = 0.9$) for the exact algorithm (defined by variable $x^m_{j,j'}(t)$). In contrast, the heuristic processes more dependencies grouped onto two clusters. In addition, the same capacity values were considered for each solution reached with the different values of α and β. Therefore, it has less opportunity to find datacenters than the capacity to allocate those large clusters, and at the same time it offers a better cost. This influences considerably the search for the optimal result which is a real compromise for both algorithms.

The performance of the greedy heuristic algorithm as compared to the exact fractional optimal solutions in terms of total storage cost are represented as a cost ratio between the cost delivered by the heuristic algorithm $HEUR$ which is a greedy approximation approach for the unsplittable intermediate data dependency placement problem, and the fractional optimal solution $FRAC\_OPT$ provided by the simplex method to the problem of

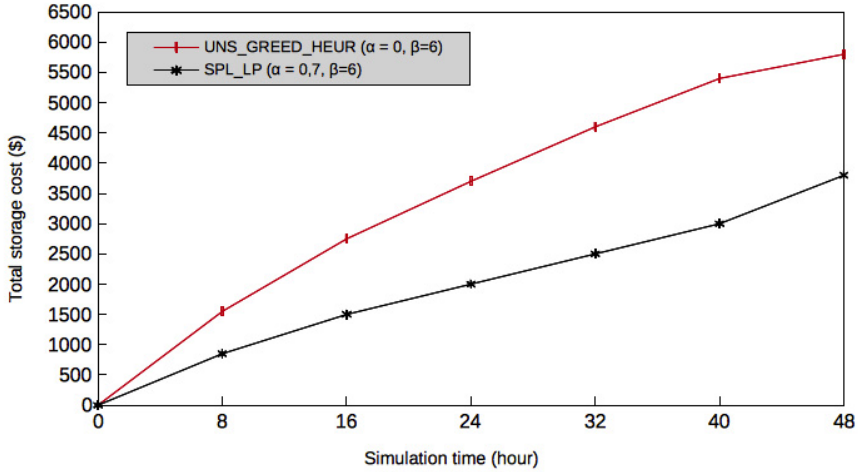

FIG. 5.7. *Greedy heuristic versus exact solutions for the total storage cost when $\alpha = 0.7$ and $\beta = 6$.*

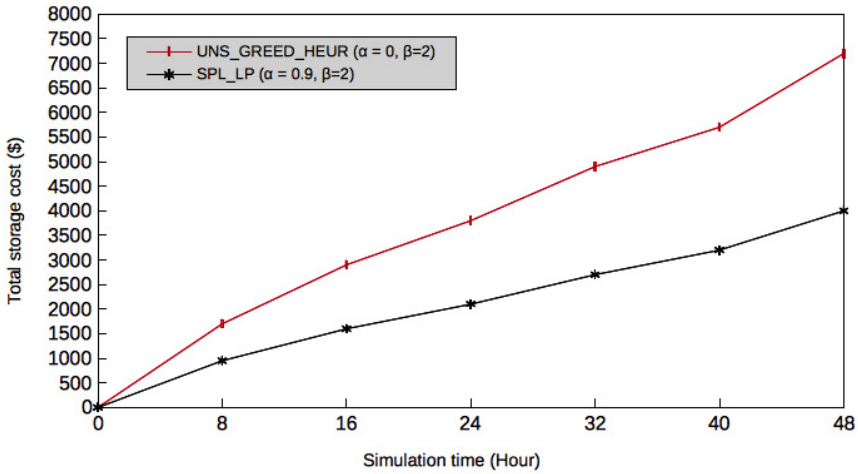

FIG. 5.8. *Greedy heuristic versus exact solutions for the total storage cost when $\alpha = 0.9$ and $\beta = 2$.*

splittable variant of the placement. The cost ratio of the heuristic $HEUR$ is $\epsilon = \frac{HEUR}{FRAC\_OPT}$. The cost ratio of the different curves above (Fig. 5.4 to 5.8) is reported in Table 5.2 when the number of datacenters varies from 5 to 50.

One can be see that the cost ratio of the greedy heuristic algorithm is no more than 1.85. Indeed, for simulated instances in the range from 5 to 50 datacenters when dependency parameter pairs $(\alpha, \beta) = \{(0.1, 18); (0.3, 14), (0.5, 10)\}$, the cost ratio of the greedy heuristic algorithm performs closer to the optimal solution and does not exceed 1.25, 1.42 and 1.52 respectively for each pair in fairly adverse conditions.

However, in the range from 5 to 50 datacenters when dependency parameter pairs $(\alpha, \beta) = (0.7, 6)$, the greedy heuristic encounters some difficulties in finding an optimal solution. Thus, the cost ratio of greedy algorithm reaches 1.81. Note that, in the greedy algorithm, the feasibility of the solution is assumed by scaling datacenter capacities. Thus, there is a solution to the problem when $\beta = 2$. The cost ratio of the greedy algorithm in this case reaches 1.85, which diverges considerably from the optimal solution, as a condition for finding any solutions that matches the optimal ones when $\alpha \leq 0.5$ and $\beta \geq 10$. Even as well, if dependency types are well identified, it is more difficult in these cases to find the best cost ratio meeting the dependency

TABLE 5.1
Gaps between the greedy heuristic and the exact algorithms in terms of cost ratio.

$(\alpha; \beta)$ / DC	(0-0.1, 18)	(0-0.3, 14)	(0-0.5, 10)	(0-0.7, 6)	(0-0.9, 2)
5	1.255	1.410	1.510	1.819	1.858
10	1.253	1.422	1.509	1.820	1.859
15	1.249	1.413	1.511	1.809	1.857
20	1.248	1.401	1.512	1.809	1.856
25	1.245	1.402	1.509	1.808	1.856
30	1.239	1.402	1.513	1.810	1.851
35	1.241	1.411	1.520	1.810	1.851
40	1.241	1.411	1.520	1.819	1.850
45	1.250	1.420	1.519	1.819	1.849
50	1.249	1.419	1.519	1.819	1.850

restrictions. Indeed, each proposed algorithm responds differently to the dependency requirements as well.

A special cases are also considered which are not reported on Table 5.2, when dependency parameter pairs are set from a range of $(\alpha, \beta) = (0.1, 1), (0.1, 2), (0.9, 19), (0.9, 18)$. These parameter values are the most extreme and contradictory cases, in the sense that for dependency pairs (0.1, 1) and (0.1, 2), the exact algorithm finds a solution with an adjustment of time (beyond the days) but could not find an optimal solution, and for the latter cases (0.9, 19) and (0.9, 18), this does not reflect the correlation-type of intra-job dependency.

We conclude that the cost ratio of the greedy algorithm depends on the value of the dependency parameters and the amount of intermediate data that increase at each time slot. In the two cases, where the dependency parameters nearly correlate $(\alpha, \beta) = \{(0.1, 18); (0.3, 14), (0.5, 10)\}$, the cost ratio is more profitable. This means that the two proposed algorithms reacted well to these dependency value requirements. However, the cost ratio of the greedy algorithm that is reported in Table 5.2 increases as dependency parameters deviate $(\alpha, \beta) = \{(0.7, 6); (0.9, 2)\}$.

5.2.3. Convergence time of the proposed algorithms. To pursue the extensive experiments, we evaluate the effectiveness of the proposed algorithms and compare them in terms of scalability and convergence time from input parameters. For the comparison, we extend the simulation by varying the number of datacenters from 10 to 100 and by setting the amount of routed intermediate data from 100 GB to 1000 GB. Obviously, the values of the dependency parameters must also vary in order to better understand the behavior of the execution time of the proposed algorithms as regard to the dependencies. Thus, the value of dependency parameters is set as specified in Sec. 5.2.2. Algorithm running times are recorded as follows.

First, the execution time of the greedy algorithm solves the placement problem one to four orders of magnitude faster than the exact solution. However, the exact algorithm solves the NP-hard problem in exponential time for large instances since a part to solve the simplex-based LP method takes much time, particularly for α values between 0.1 and 0.5 because the intermediate data splitting parameters are less tolerated throughout their placement. Furthermore, the values of dependency parameters correlate with the continuous amount of intermediate data bounded by a discrete quantity. Not surprisingly, greedy heuristic is much easier to solve than the exact algorithm.

Indeed, Fig. 5.9 shows the best convergence time for each of the proposed algorithms.

The time needed to find an optimal solution when the amount of intermediate data to be hosted is 100 GB remains very satisfactory for datacenter sizes below 10, with less than 0.075 and 0.7 seconds for greedy heuristic and exact algorithms respectively. For datacenter sizes below 50, the convergence time remains fairly reasonable too, with less than 0.15 and 1.05 seconds for greedy heuristic and exact algorithms respectively. For the latter, it slightly increases when the number of datacenters is beyond 100 (about 5 seconds). In fact, the exact algorithm performance gradually degrades with input network topology and exponentially grows for wide range (not shown in Fig. 5.4). The following figures (Fig. 5.10 and 5.11) show these behaviors.

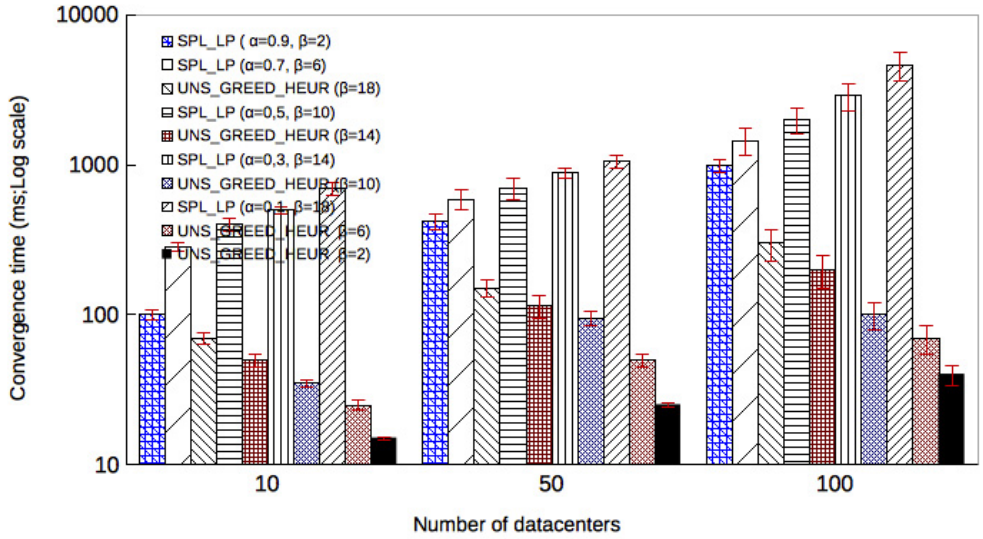

FIG. 5.9. *Time execution comparison between greedy heuristic and exact algorithms for different datacenter size when varying the amount of generated intermediate data ($|\Phi^M| = 100$ GB).*

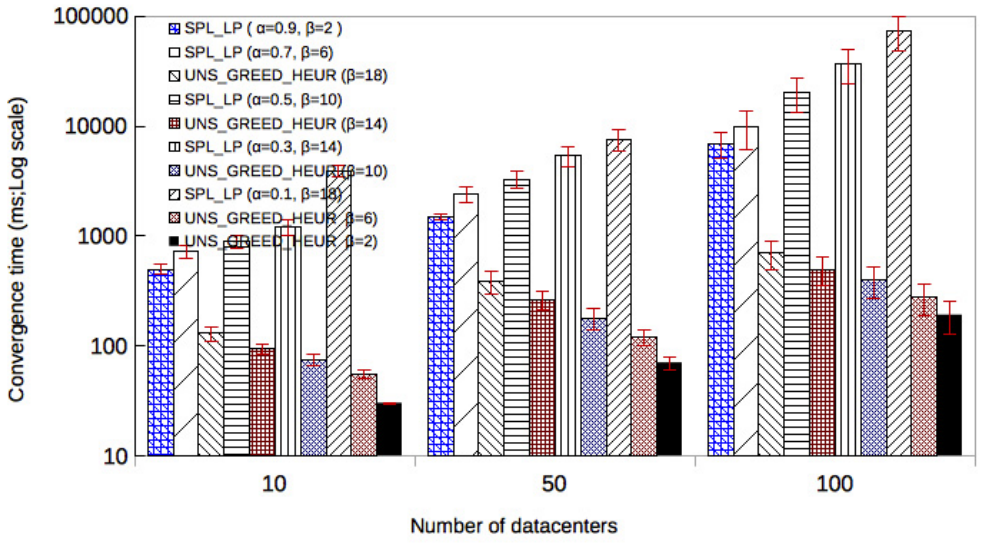

FIG. 5.10. *Time execution comparison between greedy heuristic and exact algorithms for different datacenter size when varying the amount of generated intermediate data ($|\Phi^M| = 500$ GB).*

Figures 5.10 and 5.11 show the worst cases for the exact algorithm. Then, the time needed for convergence grows mainly for an amount of placed intermediate data. These amount varies between 500 GB and 1000 GB for the simulated scenarios from 50 to 100 datacenters while the time running the greedy heuristic remains very fast to find solutions with a convergence time improvement ratio in range $[10^1,10^3]$ as compared to the exact algorithm. Although dependency parameter values vary, the number of routing β from 2 to 18 commodities, the heuristic algorithm scales better already for these large instances and it is more robust in ensuing scenarios and simulations. By contrast, the exact algorithm performance reacts poorly to dependency parameter variations.

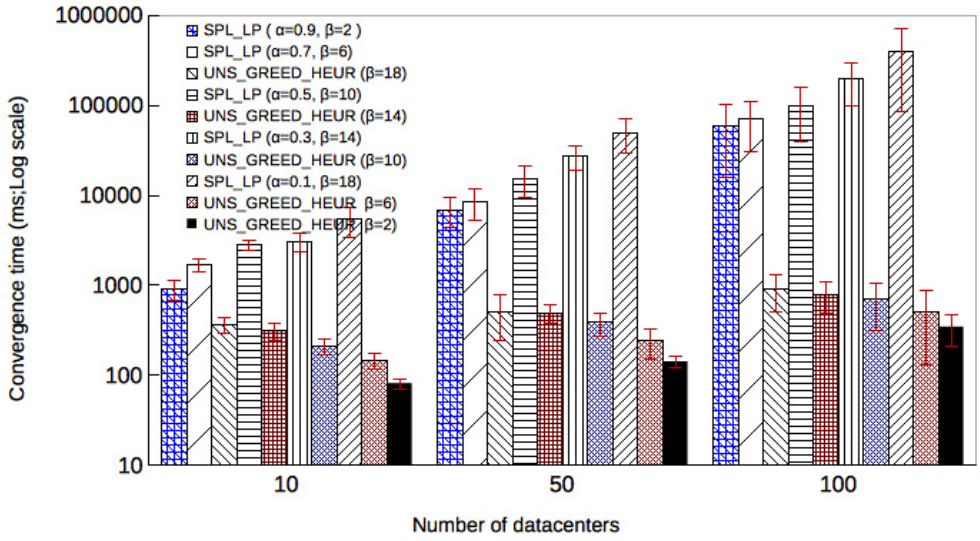

FIG. 5.11. *Time execution comparison between greedy heuristic and exact algorithms for different datacenter size when varying the amount of generated intermediate data ($|\Phi^M| = 1000$ GB).*

Particularly, this corresponds to values of α that vary from 0.1 to 0.5, where the exact algorithm exceeds a running time of one minute as shown in Fig. 5.10.

The convergence time increases with the number of datacenters as well as slightly less with the amount of intermediate data (see Fig. 5.11). As a matter of fact, the capacity of bandwidth is limited to 10 GB for data transfer, but with more transfer links through the growth of the number of datacenters.

The gap between the algorithms is in range $[10^2,10^4]$ with an improvement factor in favor of the greedy heuristic.

As expected for the exact algorithm that was built upon the simplex method (which is based on the number of intermediate data to be fractionated and this is done for each iteration as the scale of the datacenter and links between them, meaning that the separation procedure is generally not polynomial) and even with the use of a set of dependency constraint values to limit the convex hull problem to find the optimal solution faster that goes beyond 500 GB for100 datacenters, the convergence time remains widely slow at about 7 minutes.

In conclusion, the execution time of the proposed algorithms depends mostly on the cloud infrastructure topology, and slightly less on the amount of intermediate data dependencies for the exact algorithm. Besides, the change in dependency parameter values influences largely the exact algorithm performance and much less the greedy heuristic. This validates the motivation for the use of a heuristic approach to find solutions faster even if there are bound to be approximated (as reported in Table 5.2).

6. Conclusion. In this work, we have studied the problem of intermediate data dependency placement. We presented and evaluated an exact model, as well as a greedy heuristic. Our proposed solutions try to save the total storage cost for an economical and efficient task workflow processing across distributed datacenters. The presented solutions take into consideration both intra- and inter-job dependencies including fractional and atomic demands respectively. The exact algorithm based on the LP model introduces new locality constraints on the optimal placement of intermediate data dependencies. The latter can be fractionated and routed in the same physical datacenter or assigned to different destinations. In addition, the exact model is generic enough to optimize the data placement for task workflow processing in cloud environment thanks to the use of a generic objective function that combines multiple criteria such as data bandwidth and storage capability, as well as data movement optimization with an approved scalability for medium instances. Despite our formulation for the LP model, the number of datacenters and the variation of intermediate data dependency parameters makes it only solvable for medium instances. In order to ensure the placement of inter-job dependency-based intermediate

data for larger instances, we developed a heuristic based on a greedy optimization framework, this, solves the problem in very fast time, making an assumption of an optimal fractional solution. The evaluation tests show that the greedy heuristic algorithm performs closer to the exact formulation solution (in the case of converged correlations), and boots higher performance as compare to other state of the art strategies. We evaluated also the convergence time of the proposed algorithms. It is improved by several orders of magnitude for the greedy heuristic algorithm compared to the exact algorithm, while making possible to solve large cloud infrastructures in a reasonable time.

REFERENCES

[1] P. Kolman, *A note on the greedy algorithm for the unsplittable flow problem*, in Information Processing Letters Elsevier, vol. 88, no 3, (2003), pp. 101–105.

[2] P. Krysta, *Greedy approximation via duality for packing, combinatorial auctions and routing*, in International Symposium on Mathematical Foundations of Computer Science (Springer 2005), pp. 615–627.

[3] Belaidouni, Meriema and Ben-Ameur, Walid, *On the minimum cost multiple-source unsplittable flow problem*, RAIRO-Operations Research , 41(3), (2007), pp. 253-273

[4] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar, *Approximation algorithms for the unsplittable flow problem*, Algorithmica, (2007), vol. 47, no 1, pp. 53-78.

[5] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes, *Globally-optimal greedy algorithms for tracking a variable number of objects*, Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on. IEEE, (2011). pp. 1201-1208.

[6] Q. Xia, W. Liang, and Z. Xu, *The operational cost minimization in distributed clouds via community-aware user data placements of social networks*, Computer Networks (2017), vol. 112, pp. 263-278.

[7] Q. Xia, Z. Xu, W. Liang, and A. Y. Zomaya, *Collaboration-and fairness-aware big data management in distributed clouds*, in IEEE Transactions on Parallel and Distributed Systems (2016), 27(7), pp. 1941-1953.

[8] P. Agrawal, D. Kifer, and C. Olston, *Scheduling shared scans of large data files*, Proceedings of the VLDB Endowment (2008), vol. 1, no 1, p. 958-969.

[9] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas, *MRShare: sharing across multiple queries in MapReduce*, Proceedings of the VLDB Endowment 3.1-2 (2010), pp. 494-505.

[10] Q. Sun, M. Romanus, T. Jin, H. Yu, P. T. Bremer, S. Petruzza, ... and M. Parashar, *In-staging data placement for asynchronous coupling of task-based scientific workflows*, in Extreme Scale Programming Models and Middlewar (ESPM2), International Workshop on (2016), pp. 2-9. IEEE.

[11] Q. Zhao, C. Xiong, X. Zhao, C. Yu, and J. Xiao, *A data placement strategy for data-intensive scientific workflows in cloud*. In Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on (2015), pp. 928-934. IEEE.

[12] M. Ebrahimi, A. Mohan, A. Kashlev, S. Lu, *BDAP: a big data placement strategy for cloud-Based scientific workflows*. In Big Data Computing Service and Applications (BigDataService), IEEE First International Conference on (2015), pp. 105-114. IEEE

[13] R. Vilaa, R. Oliveira, J. Pereira, *A correlation-aware data placement strategy for key-value stores*, in IFIP International Conference on Distributed Applications and Interoperable Systems (2011), pp. 214-227. Springer Berlin Heidelberg.

[14] Q. Zhao, C. Xiong, P. Wang, *Heuristic Data Placement for Data-Intensive Applications in Heterogeneous Cloud*, in Journal of Electrical and Computer Engineering, 2016.

[15] Asano, Y., *Experimental Evaluation of Approximation Algorithms for the Minimum Cost Multiple-source Unsplittable Flow Problem*, In ICALP Satellite Workshops (2000), pp. 111-122

[16] D. Warneke, O. Kao, *Nephele: efficient parallel data processing in the cloud*, in Proceedings of the 2nd workshop on many-task computing on grids and supercomputers. ACM (2009), pp. 8.

[17] X. Liu, A. Datta, *Towards intelligent data placement for scientific workflows in collaborative cloud environment*, in Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on. IEEE (2011), pp. 1052-1061.

[18] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, *A market-oriented hierarchical scheduling strategy in cloud workflow systems*, The Journal of Supercomputing (2013), vol. 63, no 1, pp. 256-293.

[19] X. Liu, Z. Ni, Z. Wu, D. Yuan, J. Chen, and Y. Yang, *A novel general framework for automatic and cost-effective handling of recoverable temporal violations in scientific workflow systems*, in Journal of Systems and Software (2011), vol. 84, no 3, pp. 492-509.

[20] D. Wang, and J. Liu, *Optimizing big data processing performance in the public cloud: opportunities and approaches*, IEEE network (2015), vol. 29, no 5, pp. 31-35.

[21] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, *Bar: An efficient data locality driven task scheduling algorithm for cloud computing*, in Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Computer Society (2011), pp. 295-304.

[22] L. Kaufman, and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*, vol. 344, (2009) John Wiley & Sons.

[23] B. Rajasekar, and S. K. Manigandan, *An Efficient Resource Allocation Strategies in Cloud Computing*, in International Journal of Innovative Research in Computer and Communication Engineering (2015), vol. 3, no 2, pp. 1239-1244.

[24] V. P. Anuradha, and D. Sumathi , *A survey on resource allocation strategies in cloud computing*, in Information Communication and Embedded Systems (ICICES), 2014 International Conference on (2014), IEEE, 2014. pp. 1-7.
[25] D. Ardagna, G Casale, M. Ciavotta, J. F. Prez, and W. Wang, *Quality-of-service in cloud computing: modeling techniques and their applications*, Journal of Internet Services and Applications (2014), vol. 5, no 1, pp. 11.
[26] Z. Xu, and W. Liang, *Operational cost minimization of distributed data centers through the provision of fair request rate allocations while meeting different user SLAs*, Computer Networks (2015), vol. 83, pp. 59-75.
[27] S. Agarwala, D. Jadav, and L. A. Bathen, *iCostale: adaptive cost optimization for storage clouds*, in Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE (2011). pp. 436-443.
[28] D. Yuan, Y. Yang, X. Liu, G. Zhang, and J. Chen, *A data dependency based strategy for intermediate data storage in scientific cloud workflow systems*, in Concurrency and Computation: Practice and Experience (2012), vol. 24, no 9, pp. 956-976.
[29] X. Ren, P. London, J. Ziani, and A. Wierman, *Joint data purchasing and data placement in a geo-distributed data market*, ACM SIGMETRICS Performance Evaluation Review. ACM (2016). pp. 383-384.
[30] T. da Silva Morais, *Joint data purchasing and data placement in a geo-distributed data marketSurvey on frameworks for distributed computing: Hadoop, Spark and Storm*, in Proceedings of the 10th Doctoral Symposium in Informatics Engineering-DSIE (2015). Vol. 15.
[31] A. Dalvandi, M. Gurusamy, and K. C. Chua, *Application scheduling, placement, and routing for power efficiency in cloud data centers*, in IEEE Transactions on Parallel and Distributed Systems (2017). 28, no. 4 pp. 947-960.
[32] A. Gawanmeh, S. Parvin and A. Alwadi, *A Genetic Algorithmic method for scheduling optimization in cloud computing services*, in Arabian Journal for Science and Engineering (2017), pp. 1-10.
[33] B. A. Rao and L. V. Vahini, *Efficient scheduling of scientific workflows using multiple site awareness big data management in cloud*, in International Journal of Scientific Research in Computer Science, Engineering and Information Technology (2018). Volume 3 — Issue 1 — ISSN : 2456-3307.
[34] M. H. Ferdausa, M. Murshedb, N Rodrigo Calheirosc and R. Buyyac, *An algorithm for network and data-aware placement of multi-tier applications in cloud data centers*, in Journal of Network and Computer Applications (2017), vol. 98, pp. 65-83.
[35] A. M Manasrah and H. Ba Ali, *Workflow Scheduling Using Hybrid GA-PSO Algorithm in Cloud Computing*, in Wireless Communications and Mobile Computing (2018), vol. 2018.
[36] N. Anwar and H. Deng, *A Hybrid Metaheuristic for Multi-Objective Scientific Workflow Scheduling in a Cloud Environment*, in Applied Sciences (2018), vol. 8, no 4, pp. 538.
[37] H. Arabnejad and J. G. Barbosa, *List scheduling algorithm for heterogeneous systems by an optimistic cost table*, in IEEE Transactions on Parallel and Distributed Systems (2014), vol. 25, no 3, pp. 682-694.
[38] M. Abdullahi and M. A. Ngadi, *Symbiotic Organism Search optimization based task scheduling in cloud computing environment*, in Future Generation Computer Systems (2016), vol. 56, p. 640-650.

Edited by: Sasko Ristov
Received: Feb 14, 2018
Accepted: Aug 24, 2018