Efficient placement design and storage cost saving for big data workflow in cloud datacenters

Sonia Ikken

THESE DE DOCTORAT CONJOINTE TELECOM SUDPARIS et L'UNIVERSITE PIERRE ET MARIE CURIE

Ecole doctorale : Informatique, Télécommunications et Electronique de Paris

Présentée par

SONIA IKKEN

Pour obtenir le grade de
DOCTEUR DE TELECOM SUDPARIS

EFFICIENT PLACEMENT DESIGN AND STORAGE COST SAVING FOR BIG DATA WORKFLOW IN CLOUD DATACENTERS

Soutenue le 14 décembre 2017 devant le jury composé de :

Prof. Hamamache Kheddouci	Rapporteur	Université Lyon 1
Prof. Lynda Mokdad	Rapporteur	Université Paris 12
Prof. Pierre Sens	Examinateur	Université Paris 6
Prof. Véronique Vèque	Examinateur	Université Paris-Sud 11
Prof. Tahar Kechadi	Encadreur	Université Collège Dublin
Dr. Éric Renault	Directeur de thèse	Télécom Sud Paris

N° NNT : 2017TELE0020

Abstract

Cloud services provide users with highly reliable, scalable and flexible storage, computing and network resources in a pay-as-you-go model. Data storage services are gaining increasing popularity and many organization types such as industrial and scientific communities, are considering moving data to the cloud datacenters. Cloud computing is one factor which accelerated the evolution of big data that is emerged alongside with it. The multi geo-distributed oriented infrastructure of cloud which enables collocating computation and data, and on-demand scaling provides an interesting option for supporting big data system. The typical cloud big data systems are the workflow-based including MapReduce which has emerged as the paradigm of choice for developing large scale data intensive applications. These systems are carried out in collaboration with researchers around the geo-distributed sites to exploit existing cloud infrastructures and perform experiments at massive scale. Data generated by such experiments are huge, valuable and stored at multiple geographical locations for reuse. Indeed, workflow systems, composed of jobs using collaborative task-based models, present new dependency and intermediate data exchange needs. The task input needs to be shared across the workflow instances which requires their partial (or all) intermediate results among each other make them available to the users. This gives rise to new issues when selecting distributed data and storage resources so that the execution of tasks or job is on time, and resource usage-cost-efficient. Furthermore, the performance of the tasks processing is governed by the efficiency of the intermediate data management.

In this thesis we tackle the problem of intermediate data management in cloud multi-datacenters by considering the requirements of the workflow applications generating them. For this aim, we design and develop models and algorithms for big data placement problem while considering the characteristic and requirements of workflow and data-intensive applications running in the underlying geo-distributed cloud infrastructure so that the data management cost of these applications is minimized. More specifically, this thesis deals with the intermediate data placement problem as a first-class citizen by considering its multiple facets and levels to provide not only a specific solution, but also a generic and complete approach.

The first problem that we address in this thesis is the study of the intermediate data access behavior of tasks running in MapReduce-Hadoop cluster. MapReduce-Hadoop serves as a means of execution of micro benchmarks which is a reference system for big data processing. Our approach develops and explores Markov model that uses spatial locality of intermediate data blocks and analyzes spill file sequentiality. We also propose a prediction algorithm

based on a Markov model to predict future intermediate data request. The whole model and algorithm were evaluated and demonstrated high prediction accuracy.

The second problem that we address in this thesis deals with storage cost minimization of intermediate data placement in federated cloud storage. Through a federation mechanism, we propose an exact algorithm named ExactFed_BDWP to assist multiple cloud datacenters hosting the generated intermediate data dependency. Under the constraints of the problem, the ExactFed_BDWP algorithm minimizes the intermediate data placement cost over the federated cloud datacenters, taking into account scientific user requirements, data dependency and data size. Experimental results show the cost-efficiency of the proposed cloud storage cost model for the intermediate data dependency placement. Finally, this thesis proposes two algorithms that involves two variants of the placement problem: splittable and unsplittable intermediate data dependency. The proposed algorithms place intermediate data by considering not only their sources locations within the different datacenters hosting them but also their dependencies using a model based on a directed acyclic graph. The main goal is to minimize the total storage cost, including effort for transferring, storing, and moving them according to their needs. For this purpose, we first develop an exact algorithm named SPL_LP which takes the needs of intra-job dependencies and shows that the optimal fractional intermediate data placement problem is NP-hard. To solve the unsplittable intermediate data placement problem from inter-job dependencies, we propose a greedy heuristic algorithm named UNS_GREED_HEUR based on network flow optimization framework. The results of performance evaluation demonstrate the promise of our intermediate data placement algorithms compared to other strategies in term of total storage cost. Additionally, by showing that even with divergent conditions, the cost ratio of the UNS_GREED_HEUR algorithm is close to the optimal solution and it reduces convergence times by several orders of magnitude comparing with both the SPL_LP and Exact_Fed_BDWP algorithms.

Acknowledgement

I would like to express my deepest gratitude to my supervisors, Dr. Eric Renault and Prof. Tahar Kechadi. This dissertation owes much to the readings and proofs of Dr. Eric, who must have spent almost as much time as me on this manuscript. Thanks to Prof. Kechadi for making me discover and love Big Data universe, a discovery without which my professional and personal background would have been quite different. Thank you very much for your kindness, complete availability, rigor, and all wise advice during my PhD study years. I hope that we can continue our collaboration.

I would like to thank the two thesis referees (rapporteurs) Prof. Lynda Mokdad and Hamamache Kheddouci who have agreed to devote a significant part of their time to reading and reviewing my dissertation. I also thank Prof. Véronique Vèque and Prof. Pierre Sens for accepting being my thesis examiners and for having accepted the chairmanship of the thesis jury.

Thank you to all the others (colleagues and friends inside and outside Telecom SudParis) who have taken the trouble to read my work helping me improve it and / or contribute to my needs in order to realize this thesis work: Amine whose criticisms and collaboration have been for me a constant source of enrichment and reflection, Djamila who provided me with a completely different but necessary look on some editorial, Houria for its technicality in practice and my office colleagues, particularly Oussama, whose remarks were very useful to me.

I can not forget to thank all members of my family for their support and encouragement and especially my dear parents and husband.

Contents

List of Figures

List of Tables

Acronyms

IoT	*Internet of Things*
IDC	*International Data Corporation*
NIST	*National Institute of Technology*
ROI	*Return on Investment*
CRM	*Customer Relationship Management*
IaaS	*Infrastructure as a Service*
SaaS	*Service as a Service*
PaaS	*Platform as a Service*
SaaS	*Storage as a Service*
WaaS	*Workflow as a Service*
DaaS	*Database as a Service*
HDFS	*Hadoop Distributed File Sytem*
HITS	*Hyperlink-Induced To- pic Search*
ILP	*Integer linear programming*
LP	*Linear programming*
DAG	*Directed Acyclic Graph*
BDA	*Big Data Application*
SDDF	*Self Defining Data Format*
LBA	*Logical block addressing*
MAPE	*Mean Absolute Percent Error*
MCMF	*Minimum Cost Multiple-source Multicommodity Flow*

Chapter 1

Introduction

Sommaire

1.1 Introduction

To date, the digital universe is facing the aftermath of data explosion, data deluge is becoming a reality accordingly. The data deluge, which is a phrase used to describe the excessively huge volume of data captured by organizations, such as the rise of social media, Internet of Things (IoT) and multimedia, at a regularly increasing basis in the world, have come into existence. In this landscape, [FB13] considered that the world wide web alone was estimated to contain 512 exabytes of data in 2009. This amount of data is available from more than one trillion web pages currently accessible on the web. As reported by the International Data Corporation (IDC) [VON+15], the amount of all digital data generated, created and consumed in a single year will rise from about 3,000 exabytes in 2012 to 40,000 exabytes in 2020. Currently, about 90% of the digital data available was created in the last 2 years [Gob13]. Acquiring, storing, curating and processing these exponentially growing the recently created digital data stands as a difficult challenge, and are often reffered to as big data. Indeed, big data describes the unprecedent growth of heterogeneous, structured or unstructured data generated and collected from all kinds of data sources mentioned above. Managing big data with diverse data formats is the main basis for competition in business and management in itself. Therefore, big data poses a challenge to industrial organizations as well as scientific researchers presenting them with a complex range of valuable-use, storage and analysis issues.

Addressing the need of big data management highly requires fundamental changes in the architecture of data management systems. Among them, the highly distributed workflow processing systems that are at the core of the management of massive volumes of complex big data. These data can be an input to an application or an intermediate output which needs to be stored and managed. Some applications of this type include high-performance scientific data processing techniques, data-intensive science and real-time streaming applications [WCAL14]. These applications are subject to a series of computation phases. Workflow frameworks integrate and coordinate multiple jobs which may contain several collaborative tasks [HPL13, KJH+14, CBHTE10]. Some of these tasks are executed sequentially but others can be executed in parallel on a distributed platform. For instance, scientific organizations such as the Telescience research Project [LDU+] execute a parallel scientific task across a distributed and heterogeneous pool of shared resources. Each task not only generates data about microscopes and bio-medical images but also needs intermediate output from its collaborative tasks of bio-medical image analysis for correlation studies. Another scientific organization concerns the Climate Corporation research that is based on data-task workflow system. They adopted a component sensor located on several locations in order to capture and generate a massive amount of data from including high-resolution agronomic, environmental and weather fields [1]. Large amount of data are generated per day from those workflow processing systems which are extremely valuable with great diversity of types. However, it becomes difficult to process and store them. Equally, other applications deal with a massive data-task workflow using the MapReduce paradigm adopted and integrated by major companies like Google, Facebook, Amazon and LinkedIn. Such an application ecosystem requires a flexible composition of workflow tasks supporting different processing phases.

Meanwhile, the emergence of cloud services offers a new key knowledge for outsourcing organizations IT infrastructures which can be required and returned on demand with its flexible pricing model [HSS+10]. Cloud primarily provides data storage and processing services, which are optimized for high availability and durability. Thus, by embracing the cloud storage and processing models through distributed datacenters, the moving of the workflow collaborative tasks to the cloud can directly perform massive-scale and complex big data storage and processing, at the expense of performance which is not the primary goal. Despite the fast transition towards cloud services use, some critical issues are raised and remain not fixed. A challenging problem, both for business and scientific researchers, is how to execute such an application in a cost-efficient manner to obtain the desired level of performance. Furthermore , some big data workflow features, such as data sharing or intermediate result reuse and geographical replications are the primary options, while many others are not supported [Tud14]: geographically distributed transfers, cost optimization, differentiated quality of service, cus-

1. http://www.concurrentinc.com/customer/the-climate-corporation/

tomizable trade-offs between cost and performance. This all brings to mind that big data workflow applications are often costly (time- and money-wise) or hard to structure because of difficulties and inefficiencies in cloud data management. In view of this, providing diversified and efficient cloud data management services are key milestones for the performance of workflow-based applications.

Consequently, this thesis focuses on the problem of big data workflow management in cloud while ensuring their cost-efficient storing and processing. However, adopting big data workflow features in a distributed cloud datacenters, is a very challenging issue. For this aim, we propose new efficient big data placement strategies while considering the characteristics of workflow and data-intensive applications running in the underlying geo-distributed cloud infrastructure.

1.2 Research Context

1.2.1 Cloud Computing & Storage Basics

IT leaders in various organisations and consulting companies such as Gartner [2] or IDC [3] consider the cloud paradigm as a highly attractive proposition in this economic landscape, offering the promise of both immediate ROI and longer-term strategic benefits. Given the potential for innovation of the cloud, this thesis is in the context of this promising technology. Indeed, cloud technologies are next-generation data-storage and distributed computing systems that enable access to virtualized resources including computation power, storage capacity and network bandwidth. These resources are dynamically provisioned on demand as a personalized inventory to meet a specific service-level agreement [BYV08]. Cloud computing solutions used for accessing geographical data first and allow users to focus on extracting value, renting and scaling different services and applications for an optimal resource utilization [169]. Furthermore, resources can be rapidly scaled up and down to meet the user's needs, thus creating the illusion of infinite resources available at any time.

As cloud computing evolved considerably over these years, many cloud service providers have an IT operation outsourcing service, as shown in Fig. 1.1 such as Amazon Web Services, Google Cloud Platform, Microsoft Azure, Rackspace and IBM Cloud. They provide many popular cloud services and applications which are very useful for our daily life. These services are deployed on multiple, large datacenters over the resources, which are geographically-distributed around the globe [BYV08]. In fact, a datacenter is a centralized repository for

2. http://www.gartner.com/technology/home.jsp
3. https://www.idc.com/about/about.jsp

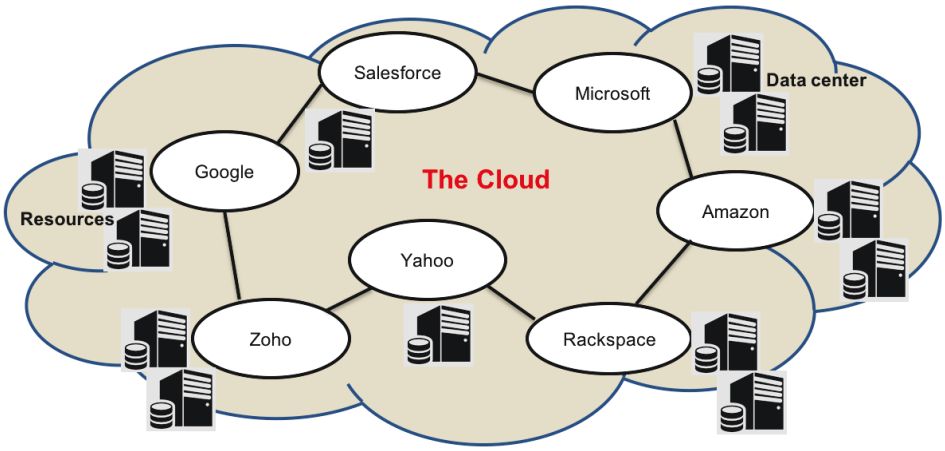

Figure 1.1 – Main cloud service providers.

the storage, management, and dissemination of data and information [Str10]. The concept of cloud computing involves a datacenter somewhere in the world, or even multiple datacenters in various countries. Furthermore, the National Institute of Technology (NIST) [HLST11] listed the five essential characteristics for cloud datacenters involving on-demand self-service, broad network access, resource pooling, rapid elasticity or expansion, and measured service.

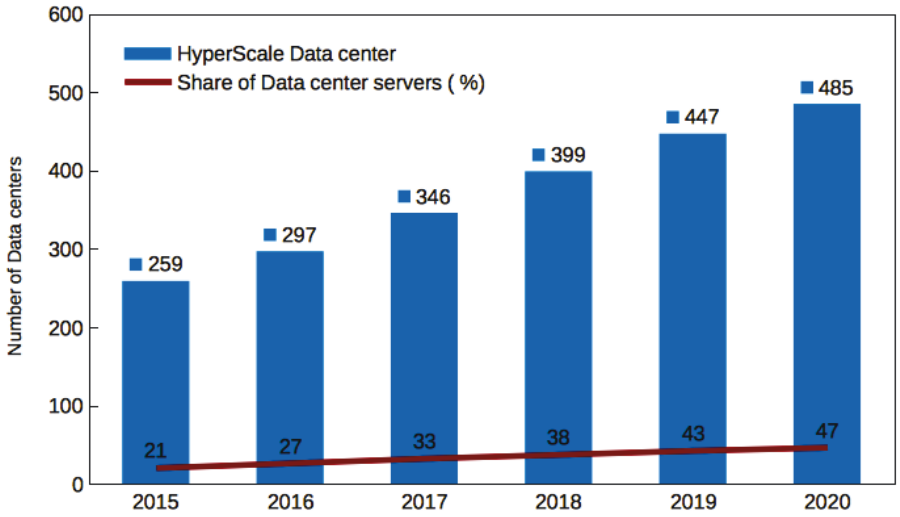

Figure 1.2 – Analysis of datacenters growth [Net15].

Meanwhile, the increased focus on business agility and cost optimization has led to the rise and growth of cloud datacenters. With the growth of data analytic as a result of big data exponential growth, there is a need to run more efficient physical servers inside datacenters. As an example of outsourcing of a company's CRM (user relationship management) software, Salesforce.com delivers their complex CRM solution to 97,000 users using 3,000 servers, a ratio of 0.031 servers per user [Rat11]. In the same context, the study conducted by CISCO exhibits

that the increasing need for datacenter and cloud resources from both the business and user service perspective has led to the development of large-scale public cloud datacenters called hyperscale datacenters. Fig. 1.2 describes the general growth of the number of datacenters since 2015 and a forecast until 2020. As depicted by this latter, the growth of the hyperscale datacenters reaches 485 which is equivalent of 47% of sharing datacenter servers from 2015 to 2020.

1.2.1.1 Cloud Deployment Models

Organizations moving towards cloud technologies have to choose between public cloud services, such as: Amazon Web Services, Microsoft Cloud and Google Cloud services, or private self-built clouds. While the firsts are offered with affordable fees, the others provide more privacy and control. Thus, with respect to the providers and their accessibility, the cloud community [4] has introduced four deployment models to define an access level for cloud deployment: private cloud, public cloud, community and hybrid clouds. Fig. 1.3 depicts a taxonomy based on usage levels that can be determined, depending on what falls under the responsibility of the provider to administrate. The resulting deployment categories providing the cloud deployment model functionalities are as follow:

— **Private Cloud:** the cloud infrastructure is deployed, maintained and operated for a specific organization. The operation may be in-house or with a third party on the premises.
— **Community Cloud:** the cloud infrastructure is shared among a number of organizations with similar interests and requirements. This may help limit the capital expenditure costs for its establishment as the costs are shared among the organizations. The operation may be in-house or with a third party on the premises.
— **Public Cloud:** the cloud infrastructure is available to the public on a commercial basis by a cloud service provider. This enables a user to develop and deploy a service in the cloud with very little financial outlay as compared to the capital expenditure requirements normally associated with other deployment options.
— **Hybrid Cloud:** the cloud infrastructure consists of a number of multi-cloud of any type. However they have the ability through their interfaces to allow data and/or applications to be moved from one cloud to another. This can be a combination of private and public clouds that support the requirement to retain some data in an organization, and also the need to offer services in the cloud.

4. https://www.dialogic.com/ /media/products/docs/whitepapers/12023-cloud-computing-wp.pdf

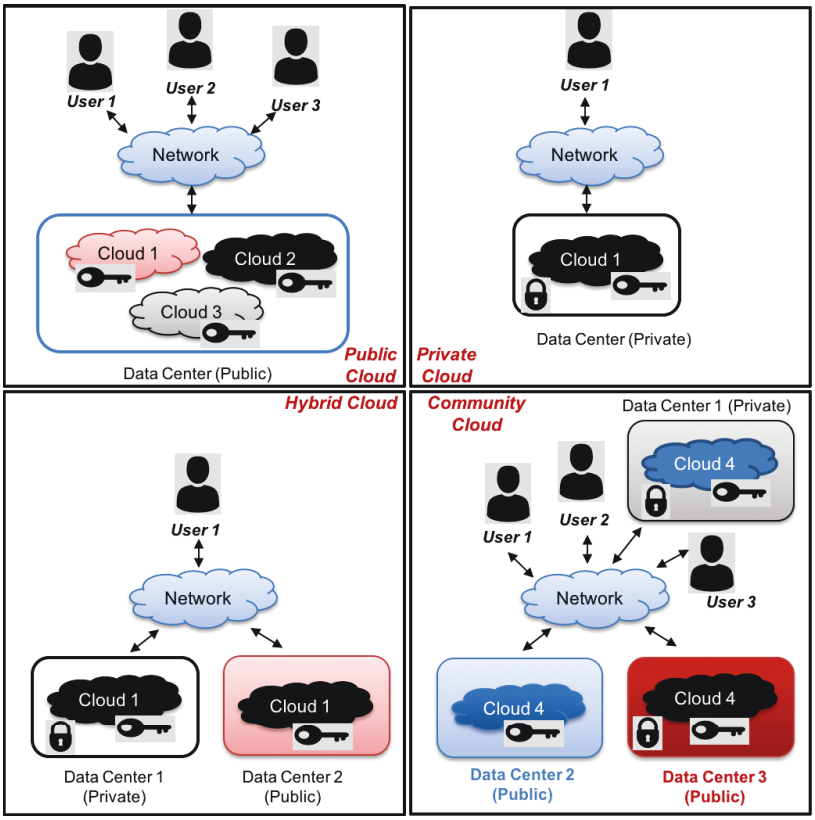

Figure 1.3 – Public, Private, Community and Hybrid cloud deployment examples.

1.2.1.2 Cloud Service Models

Different types of cloud services target separate user groups, and are basically delivered under three well-discussed layers namely the Infrastructure as a Service (IaaS), the Platform as a Service (PaaS) and the Software as a Service (AaaS). IaaS and PaaS services are usually purchased by enterprise users, while SaaS services, likewise Web services, are aimed at both corporate and individual users [HK10]. Building upon hardware facilities, these service models are offered in various forms that we present below:

— **Saas:** users are able to access and use an application or service that is hosted in the infrastructure of the provider or platform through an interface. They are not responsible of managing or maintaining the used cloud resources.

— **Paas:** users are able to access the platforms, enabling them to deploy their own software and applications in the cloud. The operating systems and network accesses are not managed by the user, and there might be constraints as to which applications can be deployed.

— **Iaas:** users control and manage the system in terms of operating systems, virtual

machines, applications, storage, and network connectivity, however they do not themselves control the cloud infrastructure.

1.2.1.3 Cloud Pricing Model

Unlike traditional Web services, each cloud service has his scheme for calculating the price for the cloud services offered to users, such as the fixed fee-paying and the dynamic pricing models. The goal of the provider is to have a greater benefit, while each user's goal is to have the maximum service for lower cost. When using dynamic or variable pricing, the price is established as a result of dynamic supply and demand, for example, as the means of auctions or negotiations [MSS16] (see Fig. 1.4). A few cloud services are free of certain level of resource charge, such as Google Docs and the Google App Engine. As compared to fixed prices, users of Amazon EC2 are billed monthly for the resources used based on the pay-per-use model, or purchase a fixed amount based on subscription to benefit from a service for a long period at any convenient time.

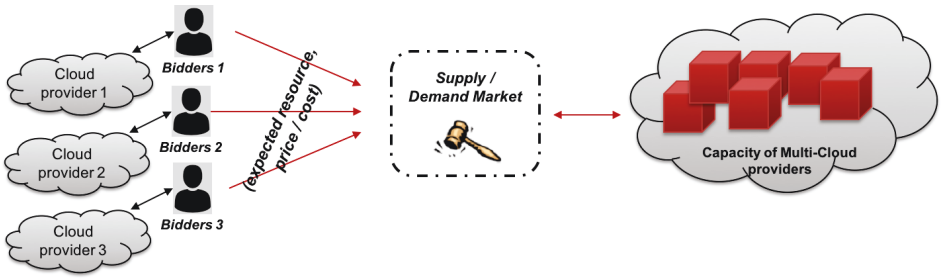

Figure 1.4 – Supply/demand market and auction mechanism.

Currently, the pricing model of cloud storage products is generally related to the amount of stored data and additional parameters of cloud data storage, which are based on the basic billing services, different payment models mean different service modes and priorities. As for the users, they can buy the corresponding storage resources according to the demand. In Google storage cloud, the price is based on a flat rate for storage and a usage rate for network bandwidth, and resources storage and bandwidth usage are calculated in gigabytes (GB). However, these mechanism does not make revenue maximization for cloud storage provider, and cannot be distinguish the pricing in relation to the requirement and need of the data user's. With the dynamic pricing model of cloud storage services, one can solve the needs of users, service requirements, and the resource allocation problem.

Besides, some dynamic pricing models are established by authors and are based on the federal cloud. Research study in [MT10] analyzed the impact of multiple types of resources and market conditions for pricing, and evaluate its performance by the use of simulation

methods, and enhance the utilization of the user. Other studies focus on the use of the federation by controlling the idle resource capacity and the problem of peaks in demand. As the average demand of the system is several times smaller than the peak demand [RHZ15], providers are able to lease part of their resources to others, in order to avoid wasting their unused resources. Hence, the following section exposes the motivation and use of the concept of federation in a cloud storage architectures.

1.2.1.4 Cloud Storage Architectures

Cloud storage provider, as one of a typical cloud infrastructure technology (i.e. IaaS), provides enterprise with an economical, feasible, flexible and convenient storage mode. Cloud storage represents a model of networked online storage where data are hosted by third parties in a single or multiple datacenters. A key benefit of datacenters is that physical hard drive storage resources are aggregated into storage pools, from which a *logical storage* is created. Big data is a key driver of the overall growth in stored data within datacenters.

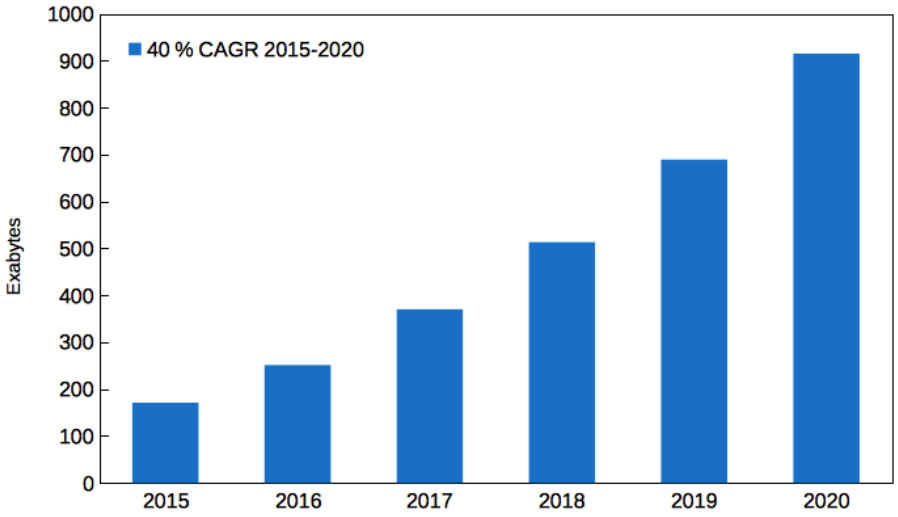

Figure 1.5 – Actual data stored in cloud datacenters [Net15].

This significant driver of traffic prepares organizations for ongoing data growth by ensuring ample capacity for new data user's and application's. The Cisco Global Cloud Index study [Net15] has estimated the total amount of actual data stored within datacenters from cloud storage provider. As depicted by Fig. 1.5, the data stored in datacenters are continuously taking extent and will reach 915 EB by 2020, up 5.3-fold (a Compound Annual Growth Rate 'CAGR' of 40%) from 171 EB in 2015. This huge data revolutionizes both enterprise, which now capitalizes the value searched in large data collections, and the scientific research, which moves towards a new paradigm: Data Science. Consequently, the applications are distributing

and scaling their processing over datacenters in order to handle overwhelming volumes, high acquisition velocities or great varieties of data.

This obviously leads to a growth of big data as shown in the Fig. 1.6. Big data volumes will reach 247 EB by 2020, up to almost 10-fold from 25 EB in 2015. Big data alone will represent 27% of data stored in datacenters by 2020, up from 15% in 2015. In fact, storage servers and datacenters remain even more essential to managing and processing the massive amount of big data created by industrial or scientific applications around the world each day.

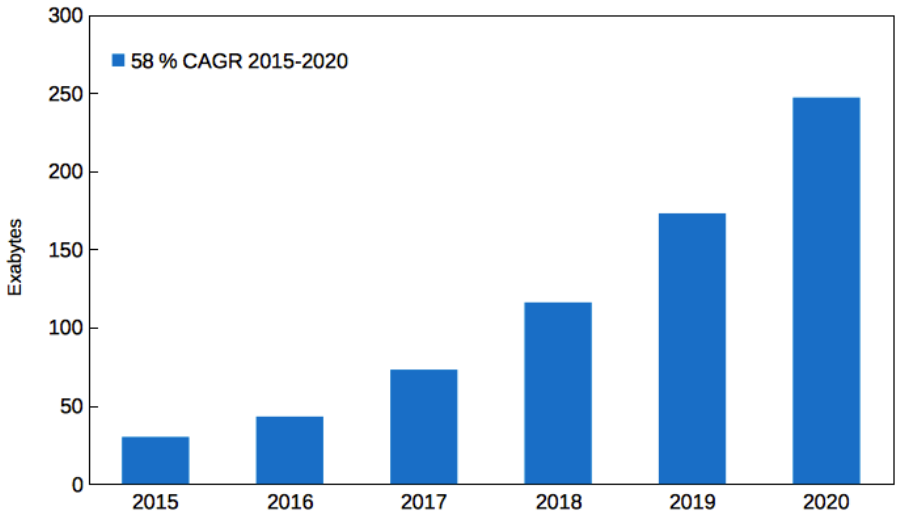

Figure 1.6 – Big Data volumes [Net15].

Despite the growth and improvement in services offered by cloud mega-providers, an enlarging number of cloud-oriented applications and global services will require provisioning for cloud-based infrastructure services involving multi-provider and multi-domain resources [MND+13]. To respond to variable and increasing data applications ensuring their continuity, multiple cloud storage providers and federation mechanisms have been proposed as a key solution to random bursts in big data application's demands. By interconnecting datacenters, cloud storage providers having complementary resource requirements over time can collaborate by sharing their respective resources and dynamically adjust their hosting capacities in response to their workloads [RHZ15].

In this context, Fig. 1.7 depicts several scenarios that can be considered to accomplish cloud federations, e.g., hybrid cloud, multi-cloud, and aggregated clouds [RHZ15, MVML12, GB14, Pet14] for the dynamic cooperation and balancing of workload among a set of cloud datacenters and providers. Cloud federation is not mature yet mainly because of the diversity of the approaches to the concept implementation.

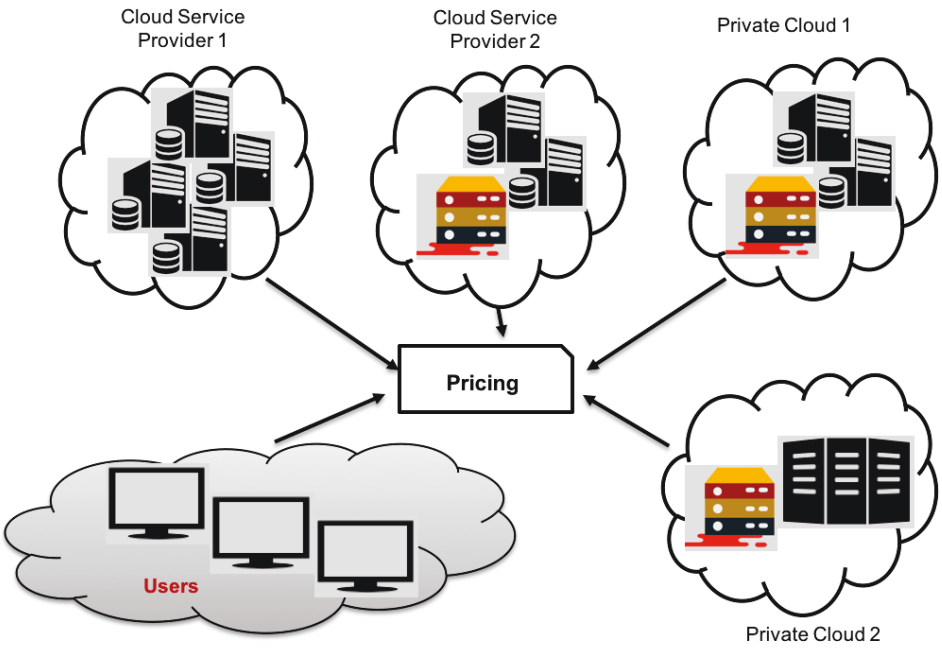

Figure 1.7 – Federated clouds [DJL+13].

In the federated cloud environment, a cloud provider acts as both infrastructure provider and user. The egocentric and rational behavior of federation members focuses on maximizing their revenue and resource utilization by serving as many users as possible. Data users thus benefit from the federation, as using more than one cloud allows to mitigate the risk of storage failures and prevents data lock-in [DJL+13].

1.2.2 The Era of Big Data

1.2.2.1 Definition of Big Data

Accordingly, Gartner Inc.[5] defines the term "big data" as a high-volume, high-velocity and/or high-variety information asset that demands cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation. This definition reflects that big data could be defined as a fast growing amount of input data and/or intermediate results from various sources or from data-driven processing that increasingly poses a challenge to industrial organizations or research communities and also presents them with a complex range of valuable-use, storage and analysis issues.

5. http://jtc1bigdatasg.nist.gov/_uploadfiles/N0095_Final_SGBD_Report_to_JTC1.docx.

1.2.2.2 Data processing in Hadoop-MapReduce Framework

Data-intensive processing included applications that generate, manipulate, and analyze large amount of data in a distributed and parallel environment such as Hadoop-MapReduce platform. With the emergence of cloud storage services, data generated from these applications are typically stored in a single data server or geo-replicated data servers. Data is organized as files (data sets) which are created and accessed by users or application providers.

Definition: Hadoop [6], an open-source project, is a framework which supports running data-intensive applications on a large cluster of commodity machines. Hadoop was adopted with hosted services and full stack distributions provided by companies such as Microsoft [7], Google [8], Greenplum [9], and Amazon [10]. Data files in Hadoop-MapReduce are stored in Hadoop Distributed File Sytem (HDFS) in data blocks which can be processed immediately by MapReduce programs.

The MapReduce programming model was first described in [DG08] as a method employed at Google for processing large-scale data-intensive applications. MapReduce paradigm is originated from the Lisp functional language, its data-parallel programing model enabling efficient implementations on different platforms and architectures. The expressiveness and the extension of MapReduce programs enabled to support iterations and incremental computations as workflow phases [TLA+16, HGC+13], higher-level data management abstractions parallel database concepts, and relational operators. Typically, MapReduce programs represent workflows of MapReduce jobs. In the current Hadoop-MapReduce framework, the client sends a MapReduce job to the master server (NameNode) as Hadoop cluster administrator, which is also the master of the cluster. Before sending a job to the NameNode, data source files should be divided into block of 64 or 128 MB, and uploaded to HDFS. Data blocks are distributed among different DataNodes within the cluster.

Intermediate data management in Hadoop-MapReduce: A data placement strategy during the execution workflow of MapReduce in Hadoop consists of two functions: map and reduce. Jobs are divided into map and reduce tasks to be executed by the mapper and the reducer respectively. First, the input block stored at DataNodes is handled by the mapper for data processing. As an intermediate output, $< key, value >$ pairs are generated which are given to the reducer which merge them to generate a single output.

6. http://hadoop.apache.org/docs/r1.0.4/index.html.
7. https://azure.microsoft.com/fr-fr/solutions/hadoop/.
8. https://cloud.google.com/hadoop/
9. http://greenplum.org
10. https://aws.amazon.com/fr/emr/

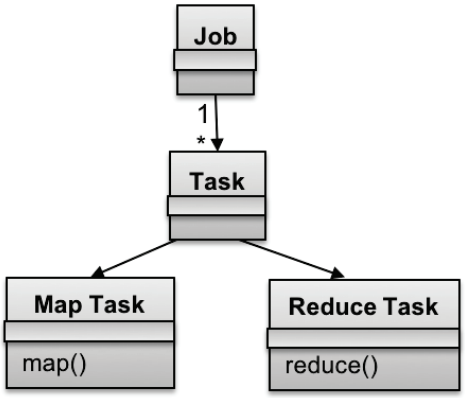

Figure 1.8 – MapReduce job hierarchy.

As shown in Fig. 1.8, a job contains multiple tasks, and each task should either be a map task or a reduce task. Functions map() and reduce() are user-defined functions and are the most important functions to achieve a user's goal. Map function transforms a $< key, value >$ pair into a list of a $< key, value >$ pairs (intermediate outputs); while reduce function aggregates all the pairs from intermediate outputs from map function sharing the same key and processes their values. The details of partition function and reduce phase are explained using the Wordcount example [11] see Fig. 1.9.

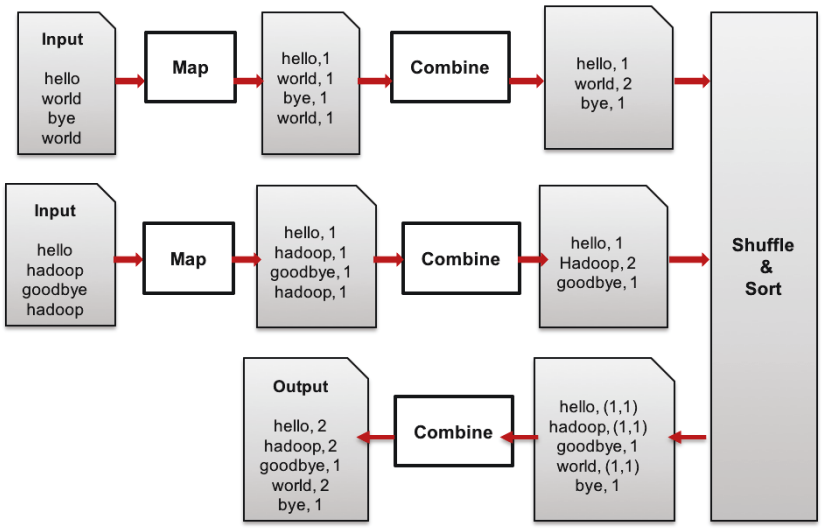

Figure 1.9 – An overview of the Wordcount MapReduce job.

The input files are first partitioned by the framework according to input split settings from the NameNode. then, a map function is executed for each input split, each returning a set of $< word, count >$ pairs. According to the map function, each output pair indicates

11. https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

a single word occurrence only. Using the combine function that summarizes the map output records with the same key, these same keyed pairs are merged locally, producing a single pair indicating the total count for a word in the input split. Following the execution of the map stage, shuffling and sorting occurs, merging the pairs returned from the separate map functions by key into $< word, list(count) >$ pairs. Reduce functions are then called, each executing on one of these pairs to compute the final data; a total count for each individual word. Map nodes periodically write on disks their intermediate data into n regions by applying the partitioning function and indicating the region locations to the master. Reduce nodes are assigned by the master to work on one or more partitions. Each reduce node first reads the partitions from the corresponding regions on the map nodes, disks, and groups the values by intermediate key, using sorting. Then, for each unique key and group of values, it calls the user reduce operation to compute a final result that is written in the output data set.

1.2.2.3 Workflow Big Data Processing in Cloud Environment

Dataflow-style workflows provide a simple, high-level programming model for modeling and implementing data scientific applications due to their natural capability for problem decomposition and coordination of dataflow and produce iterative computations [Zha12, JDB14]. A large number of workflow engines were developed due to their provided distinctiveness, such as Taverna [12], or more generic workflow models like Pegasus [13] and Kepler [14]. They facilitate the expression of multi-phase task workloads in a manner which is easy to understand, debug, and maintain. By using scientific workflows, multiple researchers can collaborate on designing a single distributed application. Meanwhile, migrating workflow management systems over a cloud infrastructure may support data parallelism over big data sets by providing scalable, distributed deployment and execution of complex data analytic applications. By exploring workflow concepts one can support data processing pipelines, and MapReduce-based constructs. Indeed, many data scientific applications cannot fit into the initial MapReduce model. Nevertheless, the tendency is to mix the expressivity provided by workflows with the simplicity of MapReduce. The goal is not only to allow users to create more diversified dependencies between tasks, but also to simplify the description of the inter-dependencies and the specifications of the big data flow. Hence, providing efficient data management services capable of interacting with the processing of these engines and of enableing high-performance data exchanges between the compute nodes is a key milestone for big data processing on cloud environment.

12. http://www.taverna.org.uk
13. https://pegasus.isi.edu
14. https://kepler-project.org

Data workflow management: The iterative computations between dataflow-tasks represent an important class of workflow applications. They are at the core of several data analysis applications [WKQ+08] such as: k-means clustering, PageRank, Hyperlink-Induced Topic Search (HITS), and numerous other machine learning and graph mining algorithms. These applications process data iteratively for a pre-specified number of iterations/phases or until a termination condition is satisfied. The intermediate output of one phase from dataflow-tasks is used as an intermediate input to subsequent phases. Some of these phases are executed sequentially but others can be executed concurrently in a distributed and parallel environment. Given the massive sizes of intermediate data and the importance of iterative computations, there is a need for scalable management solutions to efficiently apply iterative computations on large scale data sets in a cloud environment. On one hand, the MapReduce framework scales to thousands of machines on a cluster or a cloud. On the other hand, it was not designed to run iterative data processing efficiently. However, the MapReduce framework coupled with a simple workflow phase allows a parallel iterative phase of the submitted data files and applications over distributed resource nodes in cloud. These applications represent a sequence of jobs and each job constitute one or multiple tasks. Hence, jobs before they are split, are labeled as tasks after the split. As depicted by Fig. 1.10, each job in step (A) corresponds to several phases of execution function in step (B), where each phase comprises multiple tasks. Specifically, job_i has map tasks denoted $m_{i,x}$ and reduce tasks denoted $r_{i,y}$, where in this case x and y represent unique identifiers for each task in a single map or reduce phase, respectively.

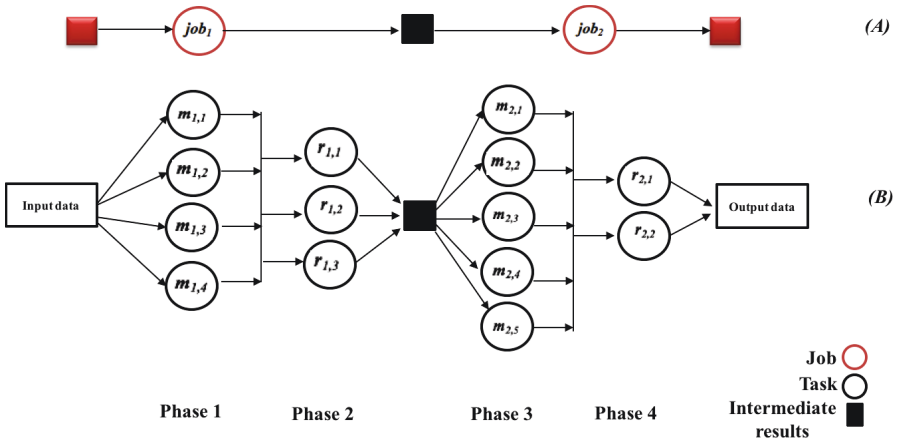

Figure 1.10 – An example of workflow phases of MapReduce jobs.

Data workflow dependency types: Moreover, during the workflow task execution, a large volume of intermediate data is generated and processed by other collaborative tasks. This reveals the type of intermediate data correlation, including inter-and intra-job dependencies [WKQ+08, DJN+10, Dai05, BF05]. Inter-job dependencies are correlations among

a set of tasks that share an amount of intermediate data that must be retrieved from their respective storage locations before their processing. Intermediate data are then forwarded to successor collaborative tasks for later reuse or reanalysis. This type of correlation is much more distinguished in batch-mode analytics with one typical application, i.e. TeraSort. TeraSort includes multiple jobs to behave in a synchronous fashion: an initial step for intermediate data production, second step for their marginal sorting and then write back to validation in a later step. Another example is the airline reservation application capturing the data tendency traveller's to keep planes fully booked. During the analysis, it is found that the result of one query is required by other queries; here is a situation where there are multiple dependent queries [NPM+10a]. In contrast, intra-job dependencies emerge in concurrent workers/reducers of MapReduce framework coupled with workflow-based processing [HGC+13]. These data correlations exploit the parallelism of MapReduce job by fractionating it among multiple tasks from synchronous iterations between map and reduce tasks as well as from asynchronous iterations between map tasks.

1.3 Research Problems & Objectives

Currently, more and more workflow-based big data processing systems are relying on the distributed cloud resources for processing and storing very large input and intermediate data results [KL14, WC16, RB15, WLN+13]. Indeed, organizations or scientific researchers who operate intensively on these systems, naturally share intermediate data gathered from one or multiple physical locations. Indeed, keeping the data storage locally and/or distributing them in multiple datacenters lead to additional costs that can be very important with the increase of the amount of stored data. This converges towards a massive growth of large amounts of intermediate data to be shared, processed and stored through cloud infrastructures. However, the geo-distributed nature of such system informally arises the intermediate data placement issue, which significantly impacts such generated data movement among the distributed datacenters and makes dependency requirements more complex for handling regarding the processing performance of these systems.

As depicted by Fig. 1.11, a naive placement strategy for such intermediate data deluge may incur huge costs on data transfer and data storage across one or distributed datacenter locations. However, the question of what is the optimal intermediate data placement required for workflow-based processing systems according to their dependency requirements to get the most efficient performance optimization while saving a storage cost still has no clear-definite answer. To avoid the cost and complexity of the intermediate data placement, previous research on those optimization problems mainly focuses on big data placement ap-

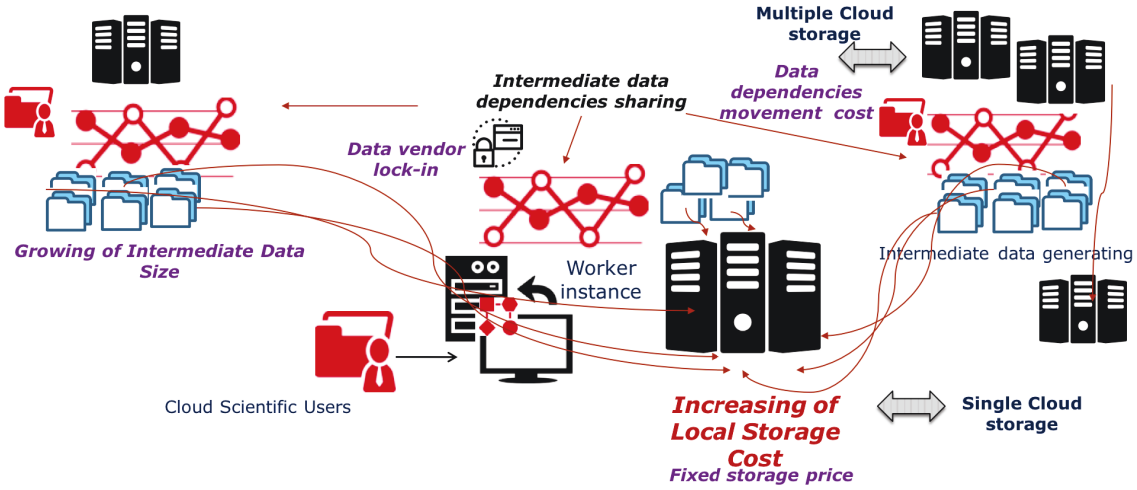

Figure 1.11 – Data workflow placement issues in cloud datacenter.

proaches [XXLZ16, GLJ17, CPL16, ZLWC14, SHW+16, EMKL15, STLM07, DJ12, BR01] where heuristic and approximation algorithms are used. However, none of them assumed the requirement and type of correlation, whereas it is a key needs in a workflow-based big-data processing application.

The problem addressed by the present thesis corresponds to the initial and dynamic data placement for workflow-based processing applications in a multiple-cloud datacenter according to intermediate data storage, transfer and movement cost. The initial data placement involves scheduling intermediate data from source location to one or multiple destinations, and from multiple datacenter destinations regarding to the dynamic data placement. Hence, our focus is to propose, design and evaluate optimization algorithms of such a data placement problem to reach the objective of minimizing the total storage cost. Indeed, we deal with the cloud infrastructure environment (IaaS) within the cloud storage provider as well as taking into consideration the data application characteristics and requirements. The basic data placement problem without considering the data application requirements is NP-hard [CPL16, ZXZ+15, ZCLS16, RLZW16, DJ12]. However, supporting workflow-based big-data applications in cloud, at large-scale, raises the need to address the detailed challenges. Below, we list the considered research issues and objectives that a cloud workflow-based data placement may encounter that are tracked in this thesis.

Data-Intensive I/O Scheduling Performance: A major challenge in big-data management issue is to first identify the required input parameters (intermediate data) to study I/O performance of data-intensive applications such as workflow and MapReduce scenarios [WuRILP17]. As an important part of these applications, short-lived data are difficult to analyze and are becoming increasingly complex because of the rapid data arriving speed and

huge size of data set in the dataflow model. Besides, the huge volume of intermediate data (logs) generated from electronic commerce and scientific computation can exceed hundreds of terabytes every day. The challenge is managing large amounts of unstructured data. For example, in organizations such as A9.com, AOL, Facebook, The New York Times, Yahoo!, the generated logs exceeds 30TB of data every day. Meanwhile, these intermediate data share some proprieties with the intermediate data from traditional file systems and local disks (e.g., temporary/logs, .o files). These intermediate data are short-lived, used immediately, written once and read once [BHK+91, KHCG09]. Furthermore , in the dataflow model, there are new specificities related to the intermediate data like blocks which are distributed, large in number, large in aggregated size, and the fact that a computation phase cannot start until all its input intermediate data have been generated by the previous phases. Despite these issues, one can observe that the intermediate data management problem is not widely disclosed in current workflow and MapReduce scenarios-based programming models. The most popular approach to intermediate data management is to rely on the local disk and file system. Data are written locally on the generating node, and read remotely by the next node that needs them. Hence, the understanding of such behavior and its implications for I/O data scheduling in such workflow and MapReduce scenarios are primary concerns of this thesis.

Sharing & Reuse Intermediate Results: Accordingly, scientific workflow applications are typically very complex. They usually have a large number of tasks and need a long time for execution. During the execution, a large volume of new intermediate results are generated. On another note, intermediate data sharing is the most difficult characteristic from big data management. Whether the scientific data workflow processing is performed within datacenters or across multiple datacenters, the task intermediate results need to be shared across the compute tasks, which in turn need to share their full or partial results among each other, with other applications within a workflow job or make them available to users. They could be even larger than the input data and contain some important intermediate results. Each job in a workflow-based processing produces output that is stored in HDFS nodes used by the MapReduce system in the case of a Hadoop cluster, or in datacenter in the case of a distributed cloud storage. These intermediate results are used as input by subsequent jobs in the workflow. As depicted by Fig. 1.12, job 1 and job 2 produce intermediate outputs that are used as intermediate inputs by job 3 and so on.

In the current practice, some intermediate results may need to be stored for future use after finishing the execution of the previous workflow jobs. Indeed, scientist users may require to re-process the results or apply new processing or analyses on the intermediate data. Specifically, in a collaborative environment, the intermediate results are shared among scientist users from different institutions and the intermediate data can be reused. Managing effectively valuable

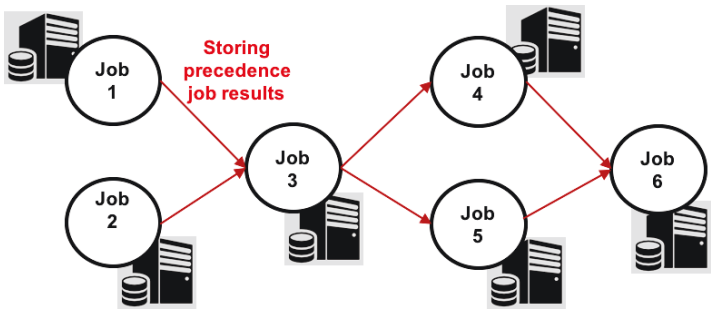

Figure 1.12 – An example of workflow of jobs processing phases.

intermediate data can save their regeneration cost and communication cost when they are reused, besides the time constraint or their regeneration. Since the cost for task workflow execution and large data size is going to be more expensive with the growth of the number of datacenters hosting them, our research must take into account the considerable cost savings that can be obtained by intermediate data placement optimization in a massive storage.

Ensuring Characteristics & Requirements: Designing, tuning, and handling workflow -based data processing application all rely on a good understanding of these applications, and in particular on the characteristics and requirements of the generated data. Indeed, these applications have a set of computation and data intensive tasks that generate many intermediate data dependencies of considerable size that exist among them. The dependency depicts the derivation relationship among the generated intermediate data. The authors in [HGC+13] describe three main types of data dependency namely, inter-query, intra-and inter-job dependencies. The first one occurs in the query operations, when the intermediate output of a query can be reused as intermediate input by another. A concrete example of this type is the most popular K items sold by a chain of stores, that can be collected from the output of a query that computes the most popular K items for each store. In the second one, dependencies among jobs are translated into workflows of jobs such that jobs cannot start before the intermediate output of their predecessors is materialized. A typical example of inter-job data dependency is given by the TeraSort application which was disclosed earlier in the Sec. 2.2.2. Regarding the last type, it was referenced in MapReduce workflow-based processing. In these applications, intra-job dependencies occur in concurrent workers/reducers of MapReduce framework (see Sec. 2.2.2). Consequently, the amount of generated intermediate data from these types of correlation is non-uniform, i.e., some intermediate data dependencies are used more than others. Thus, their storage and prior location should be considered seriously to meet the needs of their dependency requirements.

Cost-effectiveness of Data Storage in Multiple datacenters: The development of efficient solutions to the mentioned workflow big data placement problem requires a pre-

allocation of the data-task computation that must be acquired beforehand to achieve a high degree of parallelism. Additionally, the performance of the data workflow processing is governed by the efficiency of the intermediate data placement. Hence, estimating the optimal amount of storage and bandwidth resources needed to run a specific complex distributed data workflow application given an input intermediate data result is a challenging problem. This influences both, scientific users who aim at minimizing their cost while meeting the application requirements, and cloud infrastructure providers who aim at controlling their resources allocation and account for resources usage saving. To achieve the best cost-effectiveness for intermediate data management, one need to investigate the following issues:

— **Intermediate Data Cost Model**: With data volume growing and cluster scaling, managing intermediate data from different geographical locations incurs operational cost for both users and cloud storage providers. Moreover, it is inefficient and costly to store the shared intermediate data anywhere within multiple datacenters. This costly data placement lies in several aspects. First, users or collaborative researchers need a new I/O cost model that can represent the amount of shared data to be managed and placed in the cloud infrastructure taking into account their requirements. Second, being oblivious to I/O intermediate data may result in substantial unnecessary data movement cost from multiple datacenters [LSWL16, ADJ$^+$10]. This large-scale distributed short-lived I/O data-intensives nature of intermediate data manifests bandwidth bottlenecks because it has to be transferred in-between workflow phases. Hence, the new I/O cost model should be able to represent the total storage cost of moving and placing intermediate data in cloud datacenters, which is the amount of intermediate data to store and their complex correlation to be managed.

— **Intermediate Data Placement Strategy**: Since at different task execution phases the intermediate data management has different requirements for data movement, the placement strategy should be aware of these data requirements before allocating the intermediate data among distributed datacenters. Even though knowing the estimated overall storage cost of intermediate data for workflow-based processing application is essential, a further step towards finding the optimal placement of these data in the cloud datacenters is required. Moreover, scientific users may have some preferences concerning the storage of some valuable intermediate data (e.g. tolerance of intermediate data dependencies). In addition to this, the intermediate data generated at different locations must be fairly placed to the distributed datacenters as regarding to the burden of busy or unused resources. Hence, based on the new cost model, one need to design cost-effective data placement strategies to meet the requirements of these different situations in cloud datacenters.

1.4 Research Contributions

Based on the discussion so far, we believe that the problem of managing intermediate data generated during the processing of workflow-based applications, deserves deeper study as a first-class problem. Motivated by the observation that the main issues is dealing with data placement, we summarize for convenience the contributions of this thesis work with respect to originally identified challenges and defined research objectives.

A survey of the big data management approaches in cloud datacenters: First, we provide a brief overview of the I/O data characterization for MapReduce application and models prediction based on Markov models. This study is the preamble for I/O data placement behavior in MapReduce workflow-based processing context. Second, we provide a deep survey on big data management approaches. A taxonomy of this kind of data management can be based on the main following criteria: i) operational storage cost, ii) dynamic or fixed pricing model, iii) data dependency constraints, iv) data placement strategy, v) federation/multi-cloud-based or single cloud provider scenario. Thus, it is our position that we must design a new storage system which combines most of these characteristics.

Studying I/O intermediate data placement behavior in MapReduce-Hadoop: To tackle the problem of I/O scheduling performance of intermediate data described above, we first propose a Markov model prediction to characterize I/O behavior of data spill access. This approach deals with intermediate data resulting from MapReduce processing in Hadoop cluster. The Markov model allows us to apply a prediction algorithm that enable to distinguish I/O intermediate data access from multiple application runs. By considering a well-defined methodology, we derive a benchmark characterizations that can be extracted from running three applications in a Hadoop cluster and that captures the I/O behavior of intermediate data files. Then, the I/O-spill characterizations are injected onto the Markov model. The study of I/O intermediate data access behavior allows to enhance the intermediate data as much regarding to their size, sources and the nature of the applications, and to position and orient our reflection to investment intermediate data placement approaches for extensible MapReduce and generic workflow applications in cloud datacenters.

Proposed models and algorithms for intermediate data placement:

— The first proposed algorithm is a new generic and exact model for intermediate data file placement in distributed cloud datacenters. The algorithm addresses both multi-cloud storage model and data placement resulting from processing workflow-based application by considering a federated storage system model characteristics including a dynamic pricing-based model for data management in multiple cloud storage

providers. The optimal exact algorithm (**Exact_Fed_BDWP**) is based on an integer linear programming model (ILP) that receives an input matrix of binary values as a dependency model for each file pair. The Exact_Fed_BDWP algorithm optimizes data placement cost in the federated cloud storage architecture when the constraints in terms of data localization, data dependency, capacity and topology of datacenters specified in the federated cloud storage model are known. Performance, fairness and scalability of the model are reported for practical instances. Despite its originality, the exact model can serve as a reference and benchmark for heuristic algorithm that will typically scale better.

— The second part of the proposed algorithms concern an extension of a placement approach for generic workflow with a specification of dependency type. To this end, we design exact (**SPL_LP**) and heuristic (**UNS_GREED_HEUR**) algorithms for intermediate data placement while satisfying application's requirements and minimizing total storage cost. In fact, the proposed approach involves two variants of the placement problem: splittable and unsplittable intermediate data dependencies including fractional and atomic demands respectively. Moreover, the proposed algorithms place the intermediate data by considering not only their source location within the different datacenters but also their dependencies using a Directed Acyclic Graph (DAG). The main goal is to save the total cost, including effort for transferring, storing, and moving them according to their needs. We first develop the SPL_LP algorithm based on the Linear Programming (LP) model which takes the needs of intra-job dependencies and shows that the optimal fractional intermediate data placement problem is NP-hard. To solve the unsplittable intermediate data placement problem from inter-job dependency, we elaborate the UNS_GREED_HEUR algorithm based on network flow optimization framework. The UNS_GREED_HEUR algorithm addresses the scalability of the model with increasing datacenters and intermediate data sizes (large graph) and focuses on the intermediate data placement resulting from intra-job dependencies. Considering a set of large clusters of data dependencies as input parameters leads to smaller structures and lower algorithmic complexity as compared to routing individual intermediate data files from a single datacenter source.

1.5 Thesis Outline

The remainder of this manuscript is organized as follows. Chapter 2 presents the most relevant big-data management strategies found in the literature. Chapter 3 outlines the design of the proposed I/O data placement and scheduling behavior that deals with a prediction

model for MapReduce-Hadoop processing framework. Chapter 4 presents a placement approach in a federated cloud datacenter that deals with data dependency inter-file generated from workflow-based processing. Chapter 5 proposes a novel data workflow management framework to deal with the total storage cost saving of the placement of data dependency type. Finally, Chapter 6 concludes the thesis and presents our future work in this research area.

Chapter 2

Big Data Management Approaches in Cloud Environment

Sommaire

2.1 Introduction

As mentioned earlier, the main objective of this thesis is the design and development of models and algorithms for data workflow placement in cloud datacenters while considering the different dimensions of the problem. These key dimensions are the variation and the dynamicity of the solution along with the storage cost and the cloud infrastructure model. To provide efficient solutions, the issue is addressed from different angles handling the constraints of the problem at different levels, thus existing state of the art methods and models need to be studied and discussed in order to effectively adjudicate the issues.

This chapter reviews the current state of the art and works on the areas related to this thesis. These studies are classified into related topics which presents the work objectives that depend mainly on the thesis positioning in relation to the existing research. The first research objective is the study of short-lived data placement behavior in MapReduce processing across storage clusters as a case study for the intermediate data placement decision. The second one is related to the research works that used the cost model to provide some features for

workflow data management from storage scenario that precisely presents the data placement problem in cloud datacenters.

2.2 I/O Data Placement Behavior from Data-intensive Computing

2.2.1 I/O prediction in MapReduce processing

Since MapReduce has become an effective and popular programming framework for parallel big-data workflow processing applications, accurate and comprehensive I/O storage has become a basic requirement for these applications. A significant effort in recent years has developed robust models and prediction approaches for I/O behavior representation of big data applications. Regarding the following approaches, authors focus on the coordination of MapReduce applications to mitigate I/O contention, and characterize the I/O behavior of HDFS and I/O request characteristics, and do not address the access behavior analysis of I/O intermediate files under I/O contention which has become one of the most important data-flow application performance bottlenecks.

In [KKC15], I/O characteristics of virtualized MapReduce applications are explored. Authors proposed an online I/O scheduler that detects the I/O burstiness of a virtual machine. Observation of virtual machine shows that concurrent I/O accesses are performed in a bursty manner, and these concurrent and bursty I/Os cause interferences among virtual machines running MapReduce applications. [MTK+15a] proposes a modeling technique to predict I/O performance for big-data applications running over the cloud. Big Data Application (or BDA) factors are identified and affect I/O performance through the measurements collected from a synthetic workload generator. [YWWL13] develops a prediction model for the resource provisioning problem for big-data systems. Systematic measurements are conducted from big-data application benchmarks, and an interference-aware solution is proposed that smartly allocates MapReduce jobs to different VMs. [Gro12] develops a model to estimate the I/O behavior of MapReduce applications when I/O interferences are included. The proposed model predicts the performance scalability of a job, which can help with making and analyzing scheduling decisions for a workload. [YLLQ12] proposes a statistical model to evaluate the effect of various configuration parameters of MapReduce jobs. In particular, statistic data analysis techniques are employed to identify the main components from workload and Hadoop configuration metrics that strongly impact the I/O workload performance. For I/O prediction, authors propose a metric related to the disk I/O intensity of a MapReduce workload which indicates the total amount of data read from and written to the local file system and HDFS.

2.2.2 Capturing I/O data complex patterns based on Markov model

While predicting I/O access patterns of data-intensive applications has long been an important goal in parallel and distributed environments, researchers have investigated statistical methods (e.g., markov predictors and hidden Markov models [Ree04, HBT+13], linear regression [Noo]), or frequent-pattern detection [LCLZ14], I/O profiling [MTK+15b] and online simulation [WKKB13] that are non-statistical methods. These approaches are predominantly based on spatial or temporal I/O behaviors that require a large number of observations to accomplish an accurate prediction. The Markov prediction model finds diverse applications in computer science, specifically in data I/O modeling and scheduling behavior prediction. Markov models can be used to establish the accuracy of those data I/O access to be measured, and it strikes an effective balance between predictive power and implementation complexity that takes long execution time or equires to offline trace-based training in order to converge. Some researches [PSS10, OR02, MR97, DSVK11, Kho] have proposed Markov-based prediction models that use these aspects to predict data I/O application performance.

[PSS10] implements a prefetching scheme based on a Markov model to predict memory references that cause a miss in the L1 cache. With the information captured from Markov miss history table and prefetch buffer, the proposed simulator maintains the next expected address, and thus improves the effectiveness of the prefetch strategy. [OR02] presents a Markov chain to predict I/O requests for scientific code applications. Authors constructed a Markov model's transition using a sparse matrix from I/O traces in Pablo-Self Defining Data Format (SDDF). All I/O traces were obtained using the Pablo I/O toolkit. Moreover, authors propose greedy prediction strategies that choose a sequence of file blocks from an I/O access pattern Markov model. Their results have shown high-prediction accuracy with variable block sizes and look-ahead lengths parameters using the proposed prediction algorithm. The work presented in [MR97] examines an approach to classify I/O access patterns using Hidden Markov models. Authors have characterized the I/O access using data training among previous program executions from benchmark experiments and sequential-/parallel-based processing applications. The proposed model demonstrates a better control over caching and prefetching policies as compared to models that are based on neural network access pattern classifications. [DSVK11] proposes a Markov Chain representation to generate I/O loads of datacenter applications. A state diagram is used to build the storage model using I/O traces collected from production servers. To validate the proposed methodology, authors compare I/O characteristics and performance metrics among the original and synthetic storage workload, and check the consistency of their results. [Kho] builds a Markov chain transition probability matrix, where each file access is a state in the chain. Learning file access chains from running multiple applications for one process would include noisy data

from the other processes. In order to solve this, authors attempt to filter out noisy data with different estimated threshold values and show the trade-offs between them.

Despite these issues, one can observe that Markov models are not applied in the context of I/O access patterns for MapReduce or/and workflow-based processing systems. Intermediate data access patterns are largely unexplored in actual statistical model prediction and in the I/O characterization whereas key skew in intermediate data has become one of the important big-data system performance bottlenecks.

2.3 Big Data Workflow Management in Cloud

A series of recent work studies that attempt to optimize big-data management in general over the cloud environment including, the data workflow routing while meeting the application requirements and/or minimizing the operational cost of the cloud service provider. Calculating the minimum data cost model is a seemingly NP-hard problem [CPL16, ZXZ+15, ZCLS16, RLZW16, GHKS13] aside of the complex dependencies among the intermediate data [EMKL15, ZXZ+15, YYL+12] generated in a distributed environment, data workflow application cost in the cloud is thus of a dynamic value. Saving the optimum data workflow cost in a cloud environment is determined by minimizing the various cost parameters in the data management perspectives [DSL+08, VSPD+13, XLX17, TTC15]. There are various cost factors which influence the moving data in cloud and most of research studies [XXLZ16, ZLWC14, LD11] did not deemed the correlation of huge generated data in their cost construction model, in addition to the type of intermediate data correlation, none has really raised the dependency type constraints that are disclosed in the previous chapter throughout the intermediate data placement decision.

2.3.1 Cost model for moving big data in cloud environment

Massive computation and storage resource capacity of cloud infrastructures allow scientific users to deploy data-intensive processing applications without infrastructure investment, where the scale of application data sets can be stored in the cloud. Different I/O strategies [DSL+08, VSPD+13, BKT13, Rei11, LSWL16, STT09] lead to different consumption of bandwidth, storage and computation resources and finally lead to different operational costs. However, scientific data user's need a new cost model that can consider the expenditure of their data-intensive application storage type [AJB11, RAH12, NPC14] in the cloud environment. Therefore, the storage model for calculating the costs related with the deployment of application processing and data-intensive management in cloud infrastructure has received a

lot of attention. Some previous works used the cost model to provide some features from cloud storage scenarios, but not necessarily for data placement or scheduling data type purposes.

2.3.1.1 Cloud cost model for data processing

The approach presented in [DSL+08] also explores only one aspect of using cloud environment for processing astronomy system montage as a data-intensive application, examining the trade-offs between different workflow execution modes and provisioning plans for cloud resources supporting both computation and long-term data storage. Authors picked a cloud service provider as the basic model for computational and cost model construction using the Montage application and the Amazon EC2 fee structure as an online case study simulation. The proposed cost models did not involve a data placement strategy.

The cloud federation concept is used in [VSPD+13] and [BKT13] evolved from the need to increase agility and efficiency-including solving scalability and management regarding processing time and job distribution on different cloud providers. A federated cloud is composed of multiple cloud service providers, public, private or hybrid, that can work together in order to meet the QoS requirements of cloud users. Authors in [BKT13] identify a redundancy strategy over federated compute resources, and then approximate by formula each quality level for single and cloud federation strategy. Consolidated storage integrating storage data from various clouds is presented in [VSPD+13]. The authors address the problem of large amounts of data that are placed in one storage system and that cannot be migrated to another vendor due to considerable expenditure and time completion. They consider a set of modular mechanism implementing a federation architecture over the VISION Cloud infrastructure assuming security issues through access control protocols. However, the authors of these works exploit the advantages offered by a federation architecture without focusing on the data workload placement problem.

2.3.1.2 Data placement strategy meeting saved cost in cloud environment

By investigating a computation and storage cost model, several research works have used a cost model to evaluate their strategies for placing data in the cloud through exact and/or heuristic approaches.

In [AJB11], authors identify an adaptive cost optimization heuristic for multi-cloud providers to decrease the storage service cost, and both a compression and placement algorithm are used to reduce data moving cost in the cloud storage. Through their cost model, the authors evaluate and compare different compression methods and placement strategies based on end-user

requirements, monetary models, data type and access frequency, but under no circumstances they address the data workflow type.

Authors of [NPC14] propose an exact approach for a storage cost optimization based on linear programming model using multiple public cloud storage providers. They first investigate the computation of ownership cost of cloud storage providers while meeting their profit. Second, they calculate the minimum cost for storing data in the cloud. However, data type and required correlation are not considered in the optimization data cost model.

Authors of [RAH12] propose an exact optimization model based on the integer linear programming problem of selecting the best storage services when taking into consideration application requirements and user priorities. The cost model is applied to different cloud providers, and the authors used BLAST and MODIS applications as data application workload to evaluate their strategy placement including application time-execution metrics and monthly budget restrictions. A total storage cost is solved while storage capacity and data workflow type requirements are not addressed in this work which is our study background.

Another aspect is studied in [GLJ17] that considers the workload distribution and data popularity of devices which affects the system's performance. They define three algorithm schemes to facilitate the assignment of data popularity changes to the heterogeneous cloud storage by optimizing the data movement from those to be migrated and relocated efficiently. However, they focus on hybrid data of different size and popularity, but they abstain from the complex data correlation as well as the induced cost from the data assignment.

The big data placement problem from a collaborative-aware environment that continuously generates data from different geographical locations has been disclosed in [XXLZ16]. The authors developed a solution to save the high cost incurring when managing the distributed big data, and they propose an approximation algorithm by reducing the data placement problem to the minimum cost multicommodity flow problem. Their solution addresses a fairly data placement respecting a fair usage of cloud services as Quality of Service (QoS) requirement of cloud provider while saving a computation and bandwidth costs. The context of their solution is closely analogue in our case study but differs mainly on the conditions and characteristics of workflow data applications and by no means disclosing intermediate data dependencies. They focus more on maximizing the system throughput in terms of data volume to be placed while saving the operational costs in the distributed datacenters.

Authors of [ZGG15] propose a cost minimization problem through the joint optimization of three factors, which are task assignment, data placement and data movement for big data applications in geo-distributed datacenters. The authors use two dimensional Markov

chains to study data transmission and computation in order to analyze the task completion time. Based on the two-dimensional Markov chain, they propose an optimization cost model based on a mixed integer linear programming to achieve the three objectives as data chunks placement, tasks distribution and datacenters resized to minimize operational cost. The same optimization parameters are developed in [BCA12]. The authors describe a resource allocation model considering time and cost sensitive execution tasks for data-intensive applications that are executed in a hybrid cloud environment. However, these research works have focused much more on factors related to task execution time and did not address the data type, I/O cost and correlation as such which are the focus of our research context.

Recently, [XLX17] has proposed strategy placements for large-volume user data of a social network in a distributed cloud datacenter while minimizing their communication and energy consumption costs as operational costs of accommodating various social networks at datacenters. The authors identifies a community fitness metric that aims at grouping the users of a social network into different group having interactions with each other while their accumulative update rate is relatively low. The fitness metric is used in the data placement algorithms while ensuring that the placed user data can be not only easily accessed and updated. However, the authors considered only the relations between the user community concept and their dynamic maintenance of the placed user data in an evolving social network, but they did not explore the use of the data correlation aspects generated from the social network scenario.

2.3.2 Data placement approaches in MapReduce and Workflow-based processing systems

MapReduce and workflow-based processing systems are migrating to the clouds in order to meet a maximum requirements of scientific applications by considering the expressivity of workflows with the modesty of the MapReduce paradigm. Jobs such as ad-hoc queries and periodic batch jobs are processed by the MapReduce framework in the form of workflow phases [EA12], and each job in a workflow of MapReduce jobs produces intermediate data results that are stored in the container (i.e., HDFS or local file system), and are used as intermediate data inputs by subsequent jobs in the workflow. At the same time, efficiently handling complex orchestrations of theses intermediate data flows on a cloud environment raises an important challenge for data management systems.

Works have been done on several types of optimization for these systems. This section focuses on one specific type of optimization, namely sharing intermediate data from computation produced between different workflow phases of MapReduce jobs. To that extent, the scheduling problem in batch data MapReduce processing systems is studied in [AKO08].

The authors present a scheduling technique for data-jobs sharing opportunities involving the scan of the input file with the goal of maximizing the likelihood of sharing scans. A similar approach optimization purpose is presented in [NPM+10b]. The authors use a cost model to save processing time and money for MapReduce in order to define an optimization problem finding optimal grouping of sets of queries and solve them using dynamic programming. Another MapReduce job scheduling approach is developed in [YS12]. The major factor that is studied is the locality constraints for reducing data sets transfer cost in limited bandwidth. For this purpose, the author in [YS12] presents a data placement strategy for improving locality and concurrency of multiple map-reduce jobs in a workflow application and investigates a scheduling algorithm considering precedence constraints among multiple map-reduce tasks and locality constraints.

The above works present a data-job scheduling issue which is not exactly the same as the data placement problem which does not have the same requirements, and does not focus on the optimization of the intermediate data scheduling as well as the incurred storage cost.

Research works on data placement problem from MapReduce application processing without referring to any specific workflow-based phases, are presented in [LHHH14, GHKS13, CPL16, ZLWC14, SHW+16]. In [CPL16], authors propose an optimization-based data placement technique to improve the performance of MapReduce tasks processing in cloud datacenters by minimizing global data locality and data access costs. The authors use a topology-aware heuristic algorithm based on an optimal replica distribution tree of distributed datacenters to network traffic reduction by considering data movement costs, but not necessarily about the data-tasks characteristics and incurring storage costs. An online optimization heuristic algorithm for a data placement problem is also presented in [ZLWC14] to improve the performance of the MapReduce framework processing from Internet Service Provider (ISP). It mainly focuses on minimizing the bandwidth cost bases on MAX contract pricing rules for uploading deferral big data by considering tolerance delay. The authors refrain from data access delay involving precedence constraints and do not consider the cases of data-job dependency processing in their heuristic placement algorithm. In [SHW+16], the energy saving dimension has been raised through a data placement approach in MapReduce processing systems. The authors focus on an equitable distribution of initial input data in order to improve the parallelism of data servers, thus reducing their idleness, and optimizing their energy consumption without really reflecting the operational storage cost and/or data correlation in the data placement decision.

The default Hadoop implementation considers a random block placement strategy [dSM15] within highly fault-tolerant through data redundancy. However, the default HDFS block placement policy assumes that all nodes in the cluster are homogeneous, and randomly place

blocks without considering any resource characteristic of nodes or/and type of data files. This basically means that the storage requirement for a file is increased by the replication factor without worrying about the cost.

2.3.3 Workflow and correlation-aware data placement

Running scientific workflow application in the cloud has to deal with data moving through multiple datacenters from a collaboration of scientists as part of different institutions. Complex correlations that are generated between scientific workflows data during their execution impose a heavy burden on computation intensive, massive storage, and communication in cloud, which hence incur considerable operational cost to cloud datacenter providers. The challenge here is to achieve such a workflow data dependency placement in a cloud environment so that the total storage cost is reduced. However, the workflow data placement problem in a cloud environment is NP-hard [ZXZ+15, SRJ+16, EMKL15]. In order to skirt its complexity, IT researchers sometimes consider theoretically unlimited cloud resources or relax some other conditions. Besides, they opt to heuristic approaches to solve the workflow data placement problem since the optimal solution is computationally intractable in large-scale cloud infrastructure. The following summarizes the most prominent workflow data placement strategies.

In cloud environment, a lot of related work has been done on workflow data management. The author of [SRJ+16] presents a heuristic data placement algorithm. It deals with improving the workflow's execution by clustering the interdependent data-sets and distributing them intelligently onto the same datacenters to reduce data transfers. The storage cost in this work is not considered and it focuses on the cost of data re-distribution through multiple datacenters only.

The same focus is presented in [EMKL15] to improve workflow performance by minimizing data movement across multiple virtual machines. The authors develop a generic population-based meta-heuristic optimization strategy for the data placement. Under this optimization problem, they highlight the intermediate data dependencies obtained during workflow execution and place them under the constraint of virtual machine storage capacity in order to minimize the data set movement between virtual machines.

The authors of [VOP11] propose a correlation-aware data placement strategy by exploiting arbitrary data dependencies easily expressed by the social read-intensive workloads. The proposed strategy places the data dependencies on the same node to reduce the communication overhead for multi-files read operations, thus fostering a multidimensional data locality

while at the same time ensuring that the storage and the query load on each server remain balanced.

In [ZXW16], authors establish a data placement algorithm based on data dependency clustering and recursive partitioning. The aim of the algorithm is to reduce the amount of data transmission and the time consumption during data-intensive application executions. In addition, the authors also consider a fixed data storage requirement, because of the problem of different ownership, some of the data sets may have a fixed storage location. The pursued strategy is extended with a heuristic to make frequent data movements occur on high-bandwidth channels of the entire cloud system.

The authors of [YYL+12] elaborate a cost-effective strategy for storing intermediate data workflow in a single cloud storage provider using the Amazon-based fixed pricing. The proposed model focuses on running a scientific workflow system in a cloud and automatically decide whether intermediate data-sets should be stored or deleted in the cloud storage provider by using an intermediate data dependency graph from the data provenance. The authors also consider the tolerance of computation delays for the relevant data processing. The works presented in [SMM+16] propose a local optimization model to minimize the storage cost saving of data processing in cloud computing. The authors consider only some of all data that are stored in the cloud without considering their correlation. In addition, most of the allocation strategies for data workflow applications in clouds do not remove intermediate data sets before the end of workflow system processing. This considerably differs from the approaches presented in [SMM+16] and in [YYL+12].

The authors of [YYLC10] present a matrix-based k-means clustering strategy for data placement in a scientific cloud workflow. The authors stress the movement of large volumes of data that can be automatically allocated among datacenters based on the data dependencies. The optimization model proposed in this work is done at the data movement level only and the authors do not define a storage cost optimization during the intermediate data placement decision. While a data workflow execution strategy may be used to meet dependency constraints, the approach proposed in [YYLC10], does not take into account the type of dependency in order to further optimize the data movement and storage cost.

A k-means heuristic algorithm to cluster a data dependency from a data-intensive scientific workflow execution is also presented in [WZDL14]. The authors deal with the data transfer problem which leads to low efficiency in actual workflow applications for scientists. The k-means algorithm consider data size and data dependency, and initially place them into the same datacenter at the workflow preparation stage, and then during the execution of the scientific workflow, they adopte a task replication strategy to reduce the volume of

intermediate data movement.

Beyond these research works, the dynamic variations of inter-and intra-jobs dependency workflow from generated intermediate data have nevertheless not been addressed with the same focus. They focused more on the clustering of dependencies than on their placement strategy themselves and the involved storage costs.

2.3.4 Summary

Table 2.1 summarizes the analysis and compares the relevant related work based on the criteria that characterizes our orientation, meeting the operational cost, pricing model, dependency constraint, placement strategy, federation/multi-cloud scenario. Unfortunately, all of them do not bring together all the relevant optimization aspects into a single approach to store, transfer and process workflow for big data while saving storage cost in a cloud environment.

2.4 Conclusion

Recent studies about data-intensive management in a cloud have shown a new way for the development of scientific workflow system. A large number of big data management approaches were developed due to their impact and efficiency in solving data workflow placement issues. Enumerating the most prominent solutions and discussing their general features regarding the purpose of this thesis has been the interest of this chapter. Our focus was to identify the main characteristics and goals that such work research share and/or diverge, when dealing with intermediate data access prediction and placement in a cloud. It is worth pointing out that all the proposed approaches do not necessarily handle or predict the intermediate data placement of workflow models, but data moving broadly in a cloud environment. Meanwhile, based on the literature review, we demonstrated that the core research issues of this thesis, i.e. intermediate data placement problem, are significant yet barely touched in the cloud. Hence, the next chapter presents our first contribution for this research direction to tackle the intermediate data placement behavior in MapReduce-hadoop cluster as a case study to address intermediate data as a first-class citizen.

Table 2.1 – Comparison of related work regarding Data Placement approaches in MapReduce and Workflow-based processing system.

Related work	Cost model parameters	Pricing model	Dependency constraint	Data placement strategy	Cloud deployment type	Exact vs Heuristic/ Online simulation
[YYL+12]	Storage, computation	Fixed	Yes	Yes	Single	Heuristic
[Rei11]	Storage, movement	Fixed	Yes	Yes	Single	Heuristic
[SRJ+16, EMKL15, VOP11]	Movement/ communication	No	Yes	Yes	Single	Heuristic
[SMM+16]	Storage	Fixed	No	No	Single	Heuristic
[ZGG15]	Storage, communication	No	No	Yes	Multi-cloud	Exact
[BCA12]	Running, transfer, computation	Fixed	No	Yes	Hybrid cloud	Heuristic
[AJB11]	Storage, request, transfer, computation	Fixed	No	Yes	Multi-cloud	Exact
[RAH12]	Storage, request, transfer, computation	Dynamic	No	Yes	Multi-cloud	Heuristic
[NPC14]	Storage, request, transfer	Fixed	No	Yes	Multi-cloud	Exact
[DSL+08]	Storage, transfer, computation	Fixed	No	No	Single	Online simulation
[VSPD+13]	storage, transfer, computation	Fixed	No	No	Federation	Online simulation
[BKT13]	computation, network	Fixed	No	No	Federation	Online simulation
[GLJ17]	Storage, bandwidth	No	No	Yes	Multi-cloud	Heuristic
[XXLZ16]	Storage, bandwidth	Fixed	No	Yes	Multi-cloud	Approximation
[SHW+16]	Energy, communication	Fixed	No	Yes	Multi-cloud	Heuristic
[EA12]	No	No	Yes	Yes	Single	Heuristic
[NPM+10b]	Running, I/O	No	Yes	Yes	Single	Exact
[YS12]	Transfer	No	Yes	Yes	Single	Heuristic
[LHHH14]	Communication, computation	No	No	Yes	Single	Heuristic
[GHKS13]	Bandwidth	Dynamic	No	Yes	Single	Heuristic
[CPL16]	Energy	No	No	Yes	Single	Heuristic
[ZXW16]	Transfer	No	Yes	Yes	Multi-cloud	Heuristic

Chapter 3

Intermediate Data I/O Interference Prediction from Co-scheduled Tasks in MapReduce-Hadoop Processing

Sommaire

3.1 Introduction

Parallel programming models [dSM15] such as MapReduce, Spark, Storm and Dryad are widely accepted for big data analytics in a distributed and parallel environment. MapReduce has received much attention and adoption since its introduction by Google. It generates numerous intermediate data which are produced as an output from one phase and used as an input for the next phase. These intermediate data are important for the completion and performance of parallel programs in cloud computing as a distributed and parallel environment. Intermediate data are stored in the local disks and fetched remotely by the tasks of the next phase. Since the total computation time for big data analytics depends on the amount of intermediate data on disk, its accurate and access behavior analyses are required and are obviously of the highest importance.

Meanwhile, cloud storage systems use modern servers that contain multiple disk drives and memory buffers. These servers are typically powerful with many CPU cores. Therefore, there is a high potential of activity that can be executed in parallel in each of these servers. Hadoop cluster, an open-source implementation of MapReduce, is often dominated by I/O bounds, particularly when reading and writing operations conducted on disk storage that are limited by CPU resources. Disk storage is either shared by multiple processes or by a number of parallel tasks (or jobs) which usually alternate between computations and I/O phases. A MapReduce application consists in many map and reduce operations involving reading and writing data on distributed file systems and local disks. Each map operation has a memory buffer used to store its intermediate data. The buffer size can be tuned by changing the Hadoop configuration. Each time the memory buffer reaches a certain threshold, a new intermediate file is created. In order to use multi-core systems, Hadoop schedules multiple operations to run concurrently on each server.

In this case, I/O resources of a server are divided among those co-scheduled tasks and are therefore exposed to I/O contention. As such, MapReduce application performance may degrade in unpredictable ways under I/O contention, even when the total utilization of the resources is low. This issue is particularly pronounced when multiple map processes are writing intermediate data files concurrently on the shared disks. This overlapped accesses pattern can increase I/O read cost at a later time during the same job processing. The need for effectively handling these intermediate data has become a major issue. Moreover, I/O intermediate data access has complex and irregular I/O patterns, mediated by multilevel I/O, notably file system policies and I/O scheduler optimized for simple sequential accesses. Therefore, foreknowledge of I/O intermediate data access behavior for the underlying layers in the I/O stack is hardly possible.

Hence, the main goal of this contribution is to predict the access behavior of intermediate data in MapReduce applications using a statistical Markov model. The intermediate data I/O interference prediction can significantly impact on application performance improvement. This model is based on the spatial locality of intermediate data blocks and it analyses the spill file sequentiality. The present work also proposes a prediction algorithm based on a Markov model for choosing sequence from the transition probability and predict future intermediate data requests. To validate the prediction model, a large number of observations from Hadoop servers have been done to extract I/O traces. Since the Markov model can discover intermediate data behavior at a low level without requiring the semantic of the information available at a higher level, the proposed simulator uses trace-driven simulations to generate I/O intermediate data from the Markov chain. The solution provides the best decision for the I/O optimization in MapReduce processing based on the prediction of intermediate data

interferences.

The remainder of this chapter is organized as follows. Section 3.2 outlines the I/O issues for MapReduce intermediate data processing. Section 3.3 presents the proposed Markov prediction model and the algorithm for predicting intermediate data access behavior on disk spill. Section 3.4 describes the evaluation and the validation methodology for the proposed approach. Section 3.5 offers a summary to conclude this contribution as well as some prior motivations and directions for the next chapters.

3.2 I/O Behavior of Intermediate Data in MapReduce-Hadoop Processing

Intermediate data in MapReduce are written and read by the same application in the form of $< key - value >$ pairs. MapReduce uses a divide-and-conquer approach in which input data are divided into fixed size units processed independently by parallel tasks. The tasks are executed and distributed across servers in the cluster. Fig. 3.1 illustrates the MapReduce processing with the two phases: the map and the reduce.

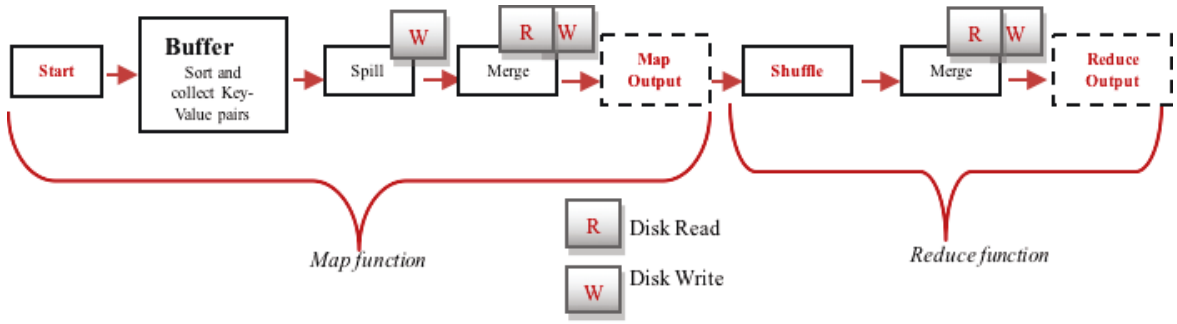

Figure 3.1 – Map and Reduce functions.

The map phase generates the intermediate data partitioned and serialized into an in-memory buffer. When the space used by the in-memory buffer reaches a given threshold, a background thread writes the output to a spill file with variable segment size on the local disk. If more than one spill are generated, individual spill files must be re-read and re-written in the merge phase. This behavior gives rise to an additional cost for disk I/Os. Any I/O performed by a map task phase is sequential. In the spill phase, spill files are written sequentially, and the merge phase sequentially reads them in an interleaved fashion. Under the shuffle phase, the reduce phase uses results from map tasks as input to a set of parallel reduce. All intermediate values associated with the same intermediate key are grouped together and passed to the reduce function. During the shuffle phase, the reduce task inputs are divided into segments,

with one segment being read from the output files of each map task spread over the entire cluster. While each segment is read sequentially, the set of segments for a particular reduce task is not stored sequentially and not guaranteed to be read in any particular order [Kho]. This situation occurs when writing concurrently intermediate data to the shared disk. As a result, there is a cost for doing a random I/O operation (disk latency and seek cost) involved for each segment of a reduce task input.

Furthermore, the tasks are not always executing the same phase and are not identical in duration. Moreover, even for tasks that are normally very similar in terms of execution time, this situation can occur because of the variance in the task execution time caused by the I/O contention. This hugely influences intermediate files that are written and read by the same application, so that any contention during the writing phase affects the performance of reading those files at a later time [GGY+13]. However, when multiple spill files are writing concurrently, fragmentation can occur for those files in many file systems, and therefore can reduces performance during read phases. So, being able to predict when spill files interfere and how this influences the spill segment size is an important part to understand and predict I/O behavior of intermediate data patterns. Once the interfered spill segment is characterized, the I/O optimization can find an interest in this prediction by revising the sort and merge phases, together with a better control of the intermediate data placement and scheduling strategies.

3.3 Methodology

Based on the observations above, a novel approach is proposed to predict the I/O behavior of intermediate data access patterns from MapReduce-Hadoop processing applications. Explicitly, this work concentrates on the characterization of the I/O behavior for spill file logs under I/O contention from the shared disk of a Hadoop cluster. However, we note that our prediction model is based on a Hadoop cluster but it can be easily adjusted to the original MapReduce on other processing platforms which are disk-based implementations. The different steps of the proposed approach are as follows (See Fig. 3.2):

Step 1-2: first, our methodology deploys Hadoop platform experimentation which is a reference system for processing big data. Since it is very complicated to trace existing MapReduce logs from real-world scenario such as Yahoo, Facebook, Google and Twitter clusters, performing a set of benchmarks as MapReduce application logs is necessary. Hence, on the Hadoop platform, a number of a map and reduce tasks are running to enforce good a prediction through three MapReduce benchmarks. **Step 3:** using these benchmarks, statistics

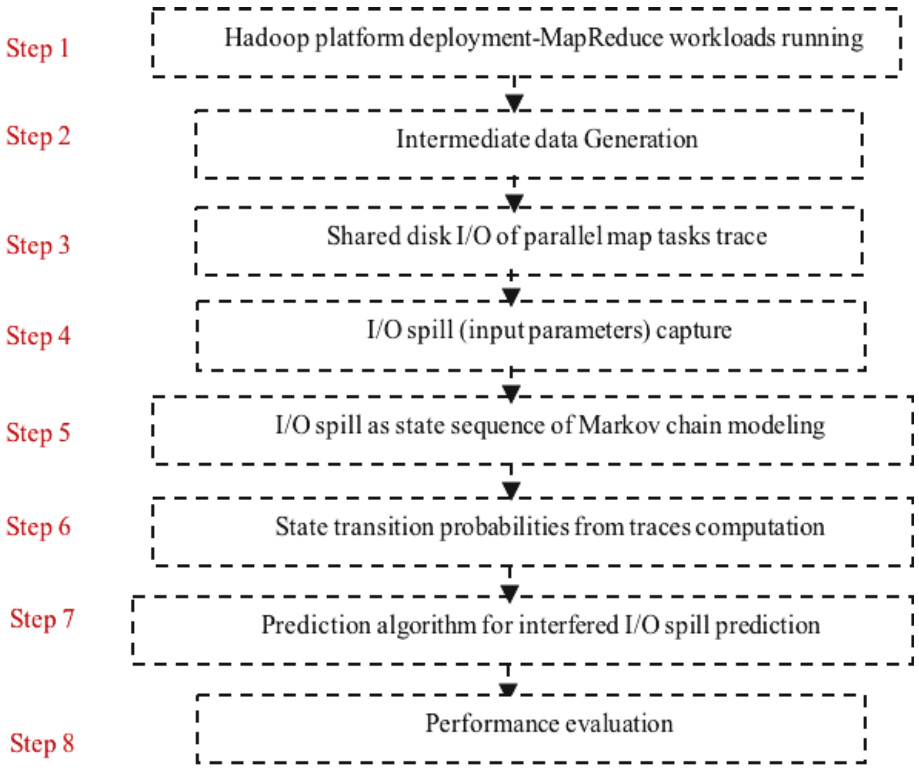

Figure 3.2 – Methodology overview depicting steps for characterizing interfered future spill segments on map tasks accessing the same disk concurrently.

are collected about disk I/O operations that are relevant to the intermediate data access patterns. **Step 4:** I/O spills are generated later and used to the I/O spill characterization. It is worth noting that a user may need to run a benchmark before using the methodology, i.e., a real case it is necessary to record the actual workload or to trace MapReduce logs before they are deleted on disks. **Step 5:** an offline trace-based training is used to converge in a Markov model. **Step 6:** according to the model, states corresponding to the I/O spill of logical blocks on disks and transitions represent the probabilities of switching among I/O spills. **Step 7:** a Markov model is proposed to predict the interfered I/O spill in the next step. **Step 8:** finally, a series of experiments is designed to evaluate the performance of the presented algorithm which validates the prediction accuracy of the Markov model.

3.3.1 System model

This section describes a simple Markov model based on sequence learning [SG01] which forms the core of our model. It is considered and evaluated in the rest of the chapter. This chain represents the behavior of disk I/O accesses of MapReduce applications by modeling a single I/O write request of spill phase during the processing of map operations. All I/O

operations to the shared disk are limited to intermediate data accesses.

The I/O spill behavior can be considered as a time dependent stochastic process which constitutes a Markov chain if the next probabilistic behavior or future access of the process depends on the present access of the process only, and is independent on its past state history. This is called the Markov property. Our model tries to solve the sequence prediction of time-homogenous Markov chain of first order (higher order Markov chains are not relevant to this paper). As a result, in this context, from a single step one can predict the future states. In this chapter, each I/O spill is followed by the same I/O spill on every write access. More specifically, the I/O spill is only determined by the current I/O spill that is accessed because observations are generated in the same homogeneous run with environmental requirements.

Definition 1. Markov chain C links successive observations R from the I/O block trace defined for the I/O accesses from concurrent map operations of one or multiple applications that share storage resources. These applications share the disk storage with fixed configuration parameters over a time period of MapReduce data processing.

Definition 2. Given a sequence observation $R = R_1, R_2, ..., R_t$ of random variable $x(t)$ on discrete state space E, each state $x(t)$ represents an I/O spill of intermediate data file. The spill file size for the corresponding states manifests the size of the in-memory buffer of the node storage as defined by *io.sort.mb* parameter. $|x|$ represents the I/O spill size. Execution window t represents the homogeneous discrete time which state has different value. t is defined within the spill phase time processing which is partitioned in multiple I/O spills that are generated by each map task slot.

Definition 3. The number of states N represents the number of I/O spills which reflects the map output total size of P parallel map operations. The size of state space E is not more than $N + 1$.

Definition 4. State transition $T_{i,j}$ at time t of variable $x(t)$ represents the probability that $x(t)$ moves from state i to state j. Assuming all I/O spills are generated by a single Markov process of first order, the conditional distribution of any future state $x(t + 1)$ only depends on present state $x(t)$. Where:

$$P(x(t) = j_0/x(t-1) = i_1, x(t-2) = i_2,) = P(x(t) = j_0/x(t-1) = i_1) = q_{i_1,j_0}(t)$$
$$\forall j_0, i_1, i_2, ... \in E \tag{3.1}$$

Definition 5. Probabilities $q_{i_1,j_0}(t)$ corresponding to the possible values of j_0 and i_1 are summarized by the $Q1_{(t-1;t)}$ matrix of (m x m) dimension, called the transition matrix from

time t - 1 to time t:

$$Q1_{(t-1;1)} = [q_{i_1,j_0}(t)] = \begin{bmatrix} q_{1,1}(t) & ... & q_{1,m}(t) \\ ... & ... & ... \\ q_{m,1}(t) & ... & q_{m,m}(t) \end{bmatrix} \tag{3.2}$$

Each of the matrix rows is a probability distribution, which implies that the sum of the elements of each row is equal to 1: $\iota = Q1_{(t-1;t)} \cdot \iota$, where ι is a vector of 1 of m size.

In the context of I/O spill from MapReduce applications, this work attempts to learn the Markov chain transition probabilities which represent the probability of switching between I/O block of map operations. Each observation in R is considered as an independent sample from an unknown distribution. R takes into account the spatial locality of the I/O block traces predicting their future values in the future time epochs from their current values.

Definition 6. Given a system parameter at the current time, the goal is to find out how I/O spill will be accessed on disk region from the current spill file. More precisely, we capture the state sequentiality of I/O spill from segments. Then, the estimation of one-step transition probability $P_{i,j}$ for i,j from 1 to N is defined by equation 3:

$$P_{i,j} = \frac{count(x_{i,j})}{count(x_{i,.})} = \frac{\sum_{t=0}^{t-1} x_{i,j}(t)}{\sum_{k=1}^{N} \sum_{t=0}^{t-1} x_{i,k}(t)} \tag{3.3}$$

Where:

— $x_{i,.}(t)$ is the number of times the I/O spill is in state i at time t ($t = 0, ..., t-1$);

— $x_{i,j}(t)$ is the number of times the I/O spill is moving from state i at time t to state j at time $t+1$.

3.3.2 Training Markov model

For each single disk I/O behavior, a Markov chain is built from blktrace-based I/O tracing [Kan09]. The Markov model is trained offline by running multiple applications simultaneously, and building the transition matrix for the map operations.

To build the Markov chain, a transition probability matrix must be identified through contingency tables of expected I/O write observations. Then, the probability vector distribution is computed for each state. The obtained traces are filtered to keep events of map operations and their I/O spill characteristics. Each map operation is represented by its parameters (I/O

spill trace). From these parameters, the expected transition is computed between each state as defined in Equation 3. Each transition captures the I/O spill behavior through the identification of the segment block size and the inter-distance between segment spill requests. Before launching the harvest of traces, values of the $mapreduce.task.io.soft.factor$ parameter of Hadoop are set to 0, specifying the number of I/O spill segments on disk to be merged at the same time. Thereby, this configuration allows to detect I/O spill segments sequentiality for each map operation before the I/O spills are merged in the merge phase on the local intermediate disk. However, a sequential operation or a request arrival is defined as one whose starting address immediately follows the last address in the previous request defined by the sector physical address and the size of the spill segment. The same idea was used in [LSD+14] to identify the sequentiality of spill segments. For each metric values change, another Markov chain is assigned and incrementally updates its parameters to reflect the characteristics of the most recent spill segment block. For different system metric values, the Markov chain is independent at time t.

Therefore, for the different parameters, a single Markov chain is generated. The update is done on another chain and this should not, under any circumstances, be influenced by environment parameters change (such as Hadoop-MapReduce platform version). However, learning from different successive observations on the same execution is done on the same Markov chain.

3.3.3 Prediction Algorithm for interfered spill requests

Table 3.1 – Input and Output of the Algorithm 1.

Parameters	Description
$TC[k, f]$	Set of Map tasks
TCk	$TCk = \{x_1, x_2, ..x_f\} \in TC[k, f]$, set of requests from Map task
$Q1[i, j]$	Transition matrix
$P_{i,j}$	$P_{i,j} \in Q1$: transition probabilities
$STCk$	Total sum of Map task requests
Sx	Total current states of a Map task
x_j	Predicted value of I/O spill request
x_i	Current value of I/O spill
$\|b\|$	I/O buffer size
x_{seqk}	Vector of sequential I/O spill requests for Map task k
$x_{Interfk}$	Vector of interfered I/O spill requests for Map task k

Using Markov transitions aims at detecting interfered I/O in the spill file from a map

operation among successive I/O spill from other map operations at block level of shared disk space. Hence, these non-sequential accesses represent interruptions of physical block addresses (non-sequential block) for an I/O spill that interfere with another I/O operation.

Algorithm 1: Prediction of interfered I/O spill requests.

Data: TCk, $Q1[i,j]$, $P_{i,j}$, $STCk$, Sx, x_j, x_i, $|x_i|$, $|b|$
Result: x_{seqk}, $x_{Interfk}$

```
while not EoF Q1[i,j] do
    for i = 1; i < N − 1; i++ do
        for j = 1; j < N; j++ do
            Sx ← Sx + x_i ;
            if P_{i,j} <> 0 and |x_i| < |b| and Sx < STCk then
                if x_j ∈ TCk then
                    x_Seqk ← x_i ;
                    else
                    x_Interfk ← x_i ;
                end
            end
        end
    end
end
Return x_Seqk, x_Interfk ;
```

The information from the transition probabilities cannot be used alone to detect these interfered I/O spills. There is a need for a prediction algorithm that predicts the next interfered I/O spill. The prediction algorithm requires indexing into an appropriate state row of the probability matrix and uses the state I/O spill size. Accordingly, our priority is to experience and know the I/O sequentiality inside the spill file and try to control the expected number of sequential I/O spill from the same map operation.

Our prediction algorithm is based on a greedy prediction since it chooses a sequence of I/O spill by repeatedly finding the most likely transition from current state x (any transition probabilities that are different from 0). By selecting all combinations from the transition probabilities, the proposed algorithm expects to control the maximum number of errors generated during the learning sequence. To achieve this, the algorithm compares each of the future access from the current state of a map task and collect non-sequential I/O spill of parallel map tasks. The stop condition on training sequences for each state is achieved when the total estimation of the one-step transition probability decreases up to a probability of 0.

Therefore, prediction results provide an I/O spill sequence that can be interfered and used as a request vector in $x_{Interfk}$. The prediction algorithm is presented in Algorithm 1. Table 3.1 shows the input/output of Algorithm 1. From the current I/O spill, the algorithm positions itself on the right row in the transition probability matrix. It browses all the columns by finding a sequence with a transition probability differing from 0. The prediction algorithm tests whether future accesses of the current I/O spill are on the same task or not and checks the size of the current state.

3.4 Experimentation Assessment & Validation

To assess the efficiency of the Markov model for predicting I/O spill behavior of MapReduce applications, a trace driven simulation is conducted. I/O traces are generated from benchmark applications running in parallel on a Hadoop platform. Table 3.2 shows the configuration parameters on five physical Hadoop distributed file system (HDFS) servers: one master server and four data servers.

3.4.1 Trace-driven assessment

I/O traces are captured using blktrace tool. The simulation is performed by a C++ program implementing Markov chains. In our experiments, recovered I/O traces are partitioned into two data sets: training intermediate data and current intermediate data. The training of intermediate data is used to build the Markov model, and the current intermediate data sets are used to assess the prediction accuracy of the Markov model and the prediction algorithm. Each trace file correlates with one disk behavior which has an entry for each accessed I/O spill. Training traces consist of record pairs (I/O size, LBA) of I/O spill, the PID, the process name, the thread-id and arguments. The simulator reads the trace file line by line and the Markov program takes the training intermediate data as an input parameters that contains I/O block issued by parallel map operations from the asynchronous flush calls. The simulator then creates distinct state from the record pair and sends the state to the transition probability function to compute the state transition.

3.4.2 MapReduce-Hadoop applications generating intermediate data sets

In order to evaluate different scenarios for the Markov prediction model, a number of representative data intensive applications for MapReduce-hadoop are needed. Applications that deal with large amounts of intermediate data are favored, having sufficient variety in the

Table 3.2 – Configuration of Hadoop server nodes for the experimentation.

Parameters	Descriptions
OS	Ubuntu Linux 14.04
Platform	Hadoop 1.0.3 version
Disk	4TB HDD (5400 tours / mn)
Processor	16-cores (2.5 GHz)
Memory	32 GB
Buffer size	80 MB
IOScheduler	CFQ
FileSystem	Ext3 (4 KB block size)
HDFS block size	512 MB

amount of data I/O intensity for the write phases from Fig. 3.1. It is clear that not all selected applications cover all the types of data I/O behavior of MapReduce-Hadoop processing applications, but these cover the most important I/O types, useful for understanding adequately the intermediate data I/O access behavior. To this end, Table 3.3 lists the intermediate data properties from three selected applications which are I/O bound and are coded in java and executed using jdk-1.6.0: Wordcount, Terasort and Kmeans. For each, a summary of parameters such as the data type, the size and the number of mappers and reducers per node are also provided. Each application generates 8 data sets randomly.

Table 3.3 – Characteristics of the intermediate data used.

Parameters	Wordcount	Terasort	Kmeans
Dataset	Random	TeraGen	Random
Total Intermediate data size	8192 MB	35840 MB	20480 MB
Map tasks per node	4	17	5
Reduce tasks per node	2	17	3

WordCount is a MapReduce program that counts the frequency of all the different words from an input file. It is a simple application often used to understand the MapReduce paradigm and included as a sample of representative of many map text manipulation tasks and reduce aggregation tasks in Hadoop platform. Each map task takes a line as input and breaks it into word emitting a $< key/value >$ pair for each founded word and 1 as a key, i.e., $< word, 1 >$. Then, each reduce task sums the counts for each word from map tasks and perform a global combine to generate the final result emitting a single $< key/value >$ pair of the word aggregation as a single record. A random text writer pattern is used to generate

input data in Hadoop-MapReduce environment.

K-means clustering is the most important online and iterative algorithm for data mining methods. We used it to partition n observations into k clusters with the MapReduce framework. In the first step, the map task read the sharing input data and compress the original data set into smaller clusters. Each map tasks then creates k initial clusters which are later sent to the reducer. A reduce task merges the clusters from each mapper and recomputes the centroids of all k clusters. Random data set are used to the clusterisation.

Terasort samples the input data estimating their distribution and performs a data partitioning using MapReduce to sort data into a total order. A 35840 MB data output size was generated by TeraGen instead of the 1 TB of data for the entire TeraSort sorting charge, which is sufficient to observe significant intermediate data I/O access.

3.4.3 Proposed approach validation

This section evaluates the prediction quality of the Markov model and Algorithm 1. The assessment concerns the prediction accuracy of I/O spill size with respect to the model size of a single step, and the number of sequential/interfered I/O spill predicted by Algorithm 1. Therefore, the chapter introduces two performance evaluations. The prediction accuracy metric is the ratio of the number of correct state model predictions labeled *"Correct Seq/interfI/O"* and the current I/O spill request labeled *"Current I/O Spill"* or current intermediate data. This metric is defined in (3.4):

$$Metric\_Pred\_Acc = \frac{\sum Correct\ Seq/interf\ IO}{Total\ Current\ IO\ Spill} \tag{3.4}$$

The second metric is the Mean Absolute Percent Error (MAPE) that measures the prediction accuracy of the proposed Markov model by measuring the size of the error in percentage terms. MAPE is defined in (3.5).

$$MAPE = \frac{1}{n} \cdot \sum_{j=1}^{n} |\frac{A_j - F_j}{A_j}| \tag{3.5}$$

Wherein, n is the number of predictions, A_j is the current value, and F_j is the predicted value.

3.4.4 Results

Fig. 3.3 depicts the fraction of the predicted state measured model size that exactly matches the current I/O spill during the processing of MapReduce applications. Map tasks generate hugely intermediate data of variable size while it is different for each application. The choice on the state size is the physical I/O size of spill file that has proved to matching well. High accuracy is showed for each model size of Terasort and Kmeans. This accuracy is near constant for the majority of I/O spill size ranging from 82% to 95%.

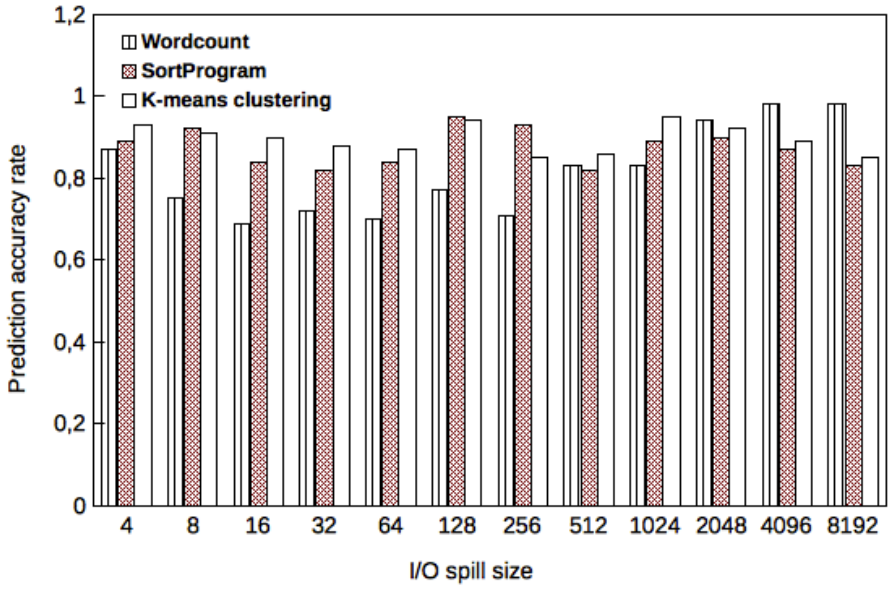

Figure 3.3 – Prediction accuracy result of I/O spill size (in KB) for the model size.

In fact, Kmeans is very I/O intensive which generates a large number of spill segments. It means that Kmeans performs heavily on I/O in the interleaved phases from concurrent map tasks. The same behavior is observed for Terasort, so that it is the most I/O intensive since it makes many partition records to have a sort order. Since the I/O bandwidth is large enough for MapReduce workloads this expects larger I/O spill sizes. The spill phase has soon become I/O bound with concurrent map tasks and producted hugely I/O.

Interestingly, some I/O spill blocks size are observed below 4MB. It is important to note that many segments deduced many index to be fragmented due to varying degrees of skewness of spill segment for the Kmeans and Terasort experiment applications. This leads to non-sequential access of I/O spill with much lower size than 4MB. Almost the same level of prediction accuracy for Wordcount and a little below as compared to Terasort and Kmeans.

Observation from the prediction accuracy rate reveals a range from 69% - 82% for I/O size

in the range from 4 KB to 512 KB and 83% to 98% for a size between 1024 KB and 8192 KB. It provides a nonlinear behavior with respect to the combination function that Wordcount used in the in-memory buffer for having a small number of reducers. Furthermore, we expect the obtained results from the interleaved accesses of Markov chain which are noise instead of another Markov chain against Terasort and Kmeans. Therefore, Wordcount generates a small amount of intermediate data. The segment size of 4 KB to 64 KB for each application may be justified with respect to the size of the metadata with the flush-call-out-dirty metadata of the spill file. These metadata are asynchronous I/Os that are serviced as multiple metadata key-value by the Ext3 file system which sets the block size to 4 KB. In conclusion, the model size and the testing intermediate data set size make use of the size of the local file system block. However, most of the I/O spills are observed in a small I/O size with a large number of I/Os.

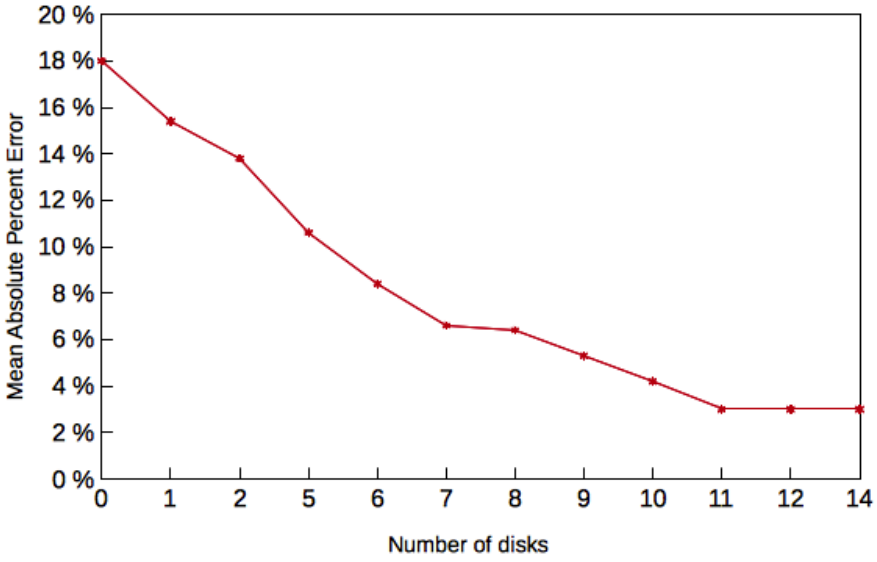

Figure 3.4 – Prediction accuracy result of Algorithm 1.

Fig. 3.4 describes the result of MAPE for the prediction Markov model. The obtained result randomly used 8 data sets to generate intermediate data. Hadoop uses random intermediate data placement on each disks having an available capacity. As expected, MAPE is linear with respect to the number of used disk traces. At a given level of number of disks, one can observe (eg. 11-case) the mean seems to be stabilizing at 3%. The present chapter provides for a less then 5 % margin of error in calculating the measures.

Several observed sequences (in term of request) utilization have been considered for our simulator to increase the prediction accuracy of the transition prediction model as depicted in Fig. 3.5. When the request sequences are more than 3000, the accuracy of our prediction

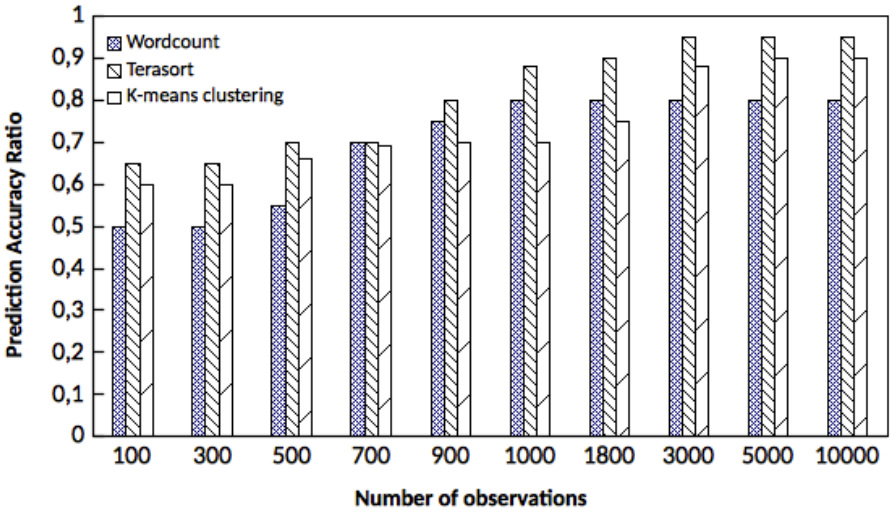

Figure 3.5 – Prediction accuracy result of Algorithm 1 based on the number of I/O request observations.

algorithm becomes stables at 96% for Terasort and 91% for Kmeans. However, the stability for Wordcount (at 81%) is maintained from 1000 request sequences due to the small number of intermediate data sets that are generated. These results are not surprising because the increase in number of request makes the estimation of Markov-model parameters more reliable, so the better accuracy prediction of the proposed algorithm. Therefore, keeping an observation time reasonable makes it sufficient to observe significant interfered I/O spills on Hadoop-MapReduce processing systems.

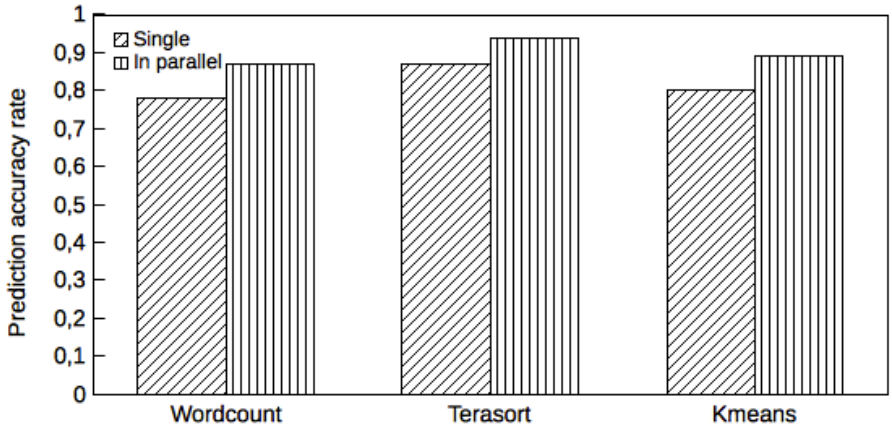

Figure 3.6 – Prediction accuracy of Algorithm 1 for processing applications from sequential and in parallel.

The proposed algorithm is also evaluated in different environments (See Fig. 3.6). It

is necessary to observe the difference in terms of the prediction accuracy for MapReduce applications in relation to the execution environment choice: sequential vs in parallel. The quality of the prediction algorithm for interfered spill segments is more accurate in sequential execution for each application. For all applications, the difference level of prediction accuracy rate is less than 9 %. Intuitively, this suggests that skewed data source or intermediate data generated in parallel can make a small limitation for prediction accuracy on the model size. This leads to generate a limit for the proposed algorithm which is precisely based on predicting the model size.

3.5 Conclusion

Due to the complexity of the intermediate data access pattern and the external conditions of the environment that influences their placement/scheduling, storing and processing, the need has become apparent to process intermediate data access patterns as a first-class citizen for big data workloads.

To achieve this, our efforts have been invested to build a prediction-based model on statistical Markov chain which can predict I/O behavior of intermediate data access in Hadoop-MapReduce processing system that has been adopted by several academic and industrial organizations. A methodology described in this work outlines the step to characterize I/O from three synthetic workload benchmarks across Hadoop platform and the construction of the Markov model. The proposed model uses knowledge about intermediate data files by tracing I/O accesses on lower-level shared disk. Using I/O block of spill files shows good construction of the model sizes. Our results have shown promise high-prediction accuracy of the proposed algorithm applied to generate predictions for interfered segment spills through the number of observations used that influences far the quality of the algorithm.

In addition, by studying detailed behavior of various representative intermediate data I/O access behaviors of Hadoop-MapReduce applications, the ultimate goal of the proposed prediction model is to improve intermediate data placement and I/O scheduling from data-intensive processing applications. This model can help to achieve that by using the information it provides to guide intermediate data placement and improve I/O scheduling decisions. Based on the observation and the prediction results, one can observe that for all of the intermediate data I/O intensive applications in Sec. 3.4.2, there are clear differences from prediction results, consequently different behavior for each such data-intensive application. This vary depending slightly in a non-linear way for each experiment application: the circumstances of the execution environment such as hardware parameters that influence data access behavior,

and especially the size of generated intermediate data, the type of application, data placement and scheduling as well as the number of parallel tasks.

Besides, intermediate data I/O performance in MapReduce-Hadoop refers to the rate at which the data are read and written from buffer memory to disk. And in parallel, one can represent the inter-spill requests as a kind of strong relationship or placement behavior of these two files. In the context of workflow big data processing, one can generalize this situation as the intermediate data transfer rate between the nodes in a cluster or in a distributed cluster such as Cloud datacenters. These circumstances prompt us to invest in methods and techniques for workflow data placement that are more elaborate in such a heterogeneous and distributed environment. More specifically, inter-file dependencies placement algorithm for multiple workflow-based tasks are the subject of the following contributions.

Chapter 4

Storage Federation Aware Big Data Workflow Placement

Sommaire

4.1 Introduction

Big data workflows have become an important paradigm since the introduction of scientific workflows and the need to formalize complex data-intensive scientific processes. After the development and wide adoption of MapReduce served as a motivation to develop big data applications, several workflow applications have been built or extended to enable programmability of MapReduce applications. Furthermore, the use of a cloud infrastructure for big data workflow application [1] facilitates the composition of the individual tasks to provide essential support to data analytics, high performance computing and on-line data storage. One common characteristic of big data workflow applications is the existence of intermediate data during the execution of workflow instances. This leads to the generation of a massive amount of intermediate results as data dependencies need to be hosted and managed over the cloud infrastructure. Handling large intermediate data dependencies in a cloud infrastructure is important for such operations that need a long time for execution since those dependencies

1. Examples: OpenStack-NovaOrchestration (https://wiki.openstack.org/wiki/NovaOrchestration), Apache Oozie (http://oozie.apache.org), Azkaban (azkaban.github.io), Cascading (www.cascading.org).

need to process intermediate results from different storage locations. As some intermediate data are too large to be relocated efficiently. This operation must take into account the dependencies between intermediate data in selecting their locality. Furthermore, scientific users share important intermediate data dependencies for cooperation and reproduction of new intermediate results. This led researchers to collaboratively work with other professionals or scientific users around the world and to handle and share intermediate data workflow enormously larger in size than before. By offering storage services in several geographically distributed datacenters, cloud infrastructures have enabled big data workflow applications to offer low-latency access to scientific user data.

However, the ever increasing volumes of scientific intermediate data address the need to interpret, move and store them more efficiently to the most appropriate datacenter. One fundamental issue in dealing with such scales of scientific user intermediate data results for a workflow application is how to efficiently place them in a distributed cloud datacenter while ensuring the dependency and scalability of the placed data such that the total storage cost of the cloud provider is minimized. On another note, cloud storage providers offer geographically distributed datacenters providing several storage classes with different prices. They can collaborate by sharing their respective resources and dynamically adjust their hosting capacities in response to their data applications. An important problem faced by cloud users is how to exploit these storage classes to serve an application with data requirements at minimum cost. A federation of existing cloud storage services supports the scientific users with a unified and combined view of storage and data services across several providers and applications. Recently, several studies have taken advantage of a variety of pricing plan of different resources in a cloud storage federation, where the cost can be optimized by trading through a negotiation storage vs. compute and network resources as well as cost optimization of data distribution across cloud providers [RHZ15, MVML12, VSPD$^+$13, BKT13] (here, the profit improvements is disregarded). None of these studies investigated the trade off between network and storage cost to optimize cost of data workflow placement across federated cloud storage provider. Our study is motivated by these pioneer issues as none of them can simultaneously answer the aforementioned questions (i.e., placement and cost saving of data workflow in cloud storage federation).

In addition to the fact that the previous chapter motivated us to consider a novel placement approach for big data workflow, hence addressing these questions makes the following main contributions: by exploiting cloud storage federation characteristics and data workflow requirements, we formalize and model the input and output parameters of the system cost model. We propose then a cost optimization problem in which the optimal cost of transferring, storing and requesting intermediate data is calculated. The exact intermediate data file

dependency is assumed to be known a priori in order to focus on the data workflow placement problem itself. In this approach, we propose an exact algorithm based on an integer linear programming model that takes into account the dependency requirements (valuable and unnecessary correlation) of these intermediate data for making decisions, and reallocates intermediate data requests with dependencies in a single datacenter to reduce the total storage cost.

This chapter is organized as follows: Sec. 4.2 describes the system model based on the cloud storage federation scenario with target assumptions. Based on this system model, Sec. 4.3 derives an exact optimization approach for allocating intermediate data on federated Cloud storage. Sec. 4.4 shows and discusses the simulation results obtained with a comparison scenario. Finally, Sec. 4.5 concludes the present chapter and presents some future work.

4.2 System Model

The target of this work is to build a federated cloud storage model to optimally allocate intermediate data workflow. To derive the model, some multiple assumptions are simplified as regard to the storage federation scenario which is focused on pricing negotiation between cloud storage service providers noted by P, which are supposed to be federated, and the storage monetary that is cheaper than local prices without federation. The federated datacenters receive large intermediate data access requests that are fairly scheduled to achieve competitive storage services, maximum storage resource utilization and prevent intermediate data lock-in. Hence, we first describe the federated cloud storage system and we discuss the model of data workflow and matrix representation that consider the dependency needs of the generated intermediate results.

4.2.1 Scenario Assumptions

The scenario introduced in Fig. 4.1 illustrates the assumed federated datacenters D from providers that are geographically distributed providing on-line mass storage to the scientific community collaborations for scheduling a set of intermediate data noted ID_i,. i being a single file with its respective size noted $size_i$ and d and d' indicate home datacenters where intermediate data are generated by a workflow instance and are temporarily stored.

The set of federated datacenters D are aggregated and interconnected in the form of an inter-cloud. They use native peer-to-peer communication links to shift intermediate results from a busy disk entity to those with an available capacity, efficiently use storage resources

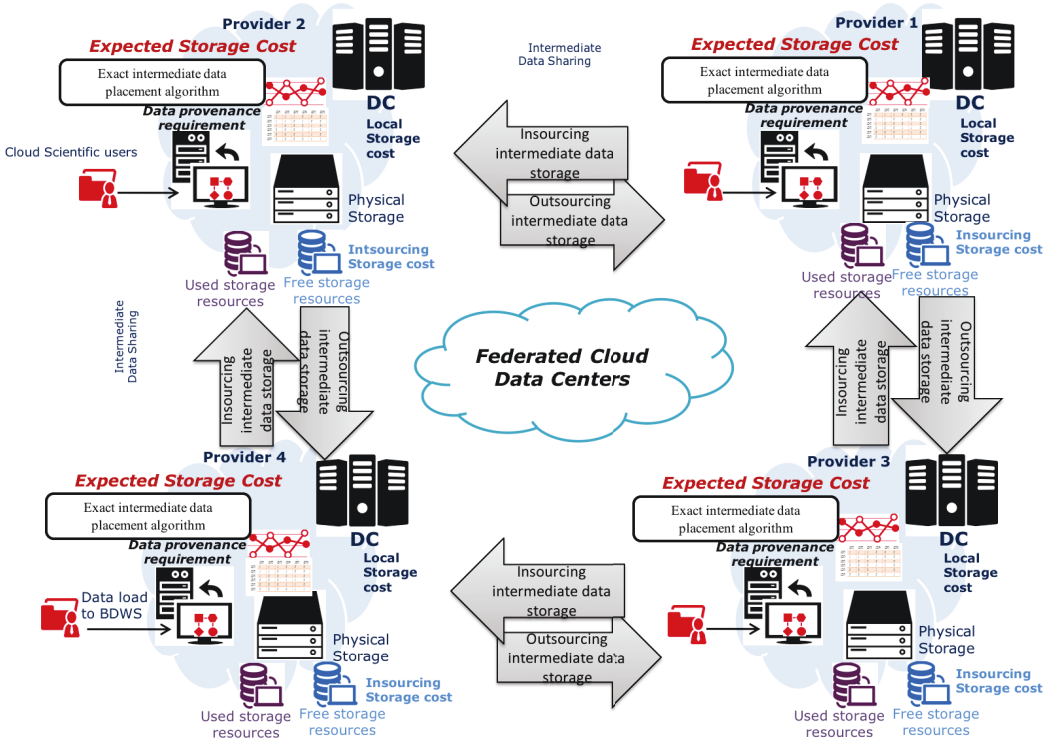

Figure 4.1 – Federated cloud datacenters scenario.

of the datacenters and balance intermediate data placement requirements. The bandwidth resource of each link is used to transfer the intermediate data file between each datacenter noted k. The intermediate data placement decision on one or multiple locations of the cloud storage federation involves the use of datacenters converging in local and storage federation for data internally reuse or by the other collaborative users. After the workflow instances have generated intermediate data locally, noted IDN_d on each datacenter, a memory temporary stores these intermediate data results before their scheduling and placement decision. The set of considered datacenters can be modeled as a matrix noted by $DChome_d$ which combines a home datacenter in order to emphasize the origin of the generated data. For the sake of convenience, we here only consider storage and data transfer resources of D as computing resource provisioning is similar to that of storage resources. Hence, to underscore the limited amount of available storage space on each federated datacenter, each member in the federation knows the storage resource quotas that is capped at SCF_k offered by the other federation members as well as their internal capacity that is capped at SCL_k. Let BCF_k be the data bandwidth shared between home datacenter and datacenter destination to transfer one unit of intermediate data file, and $DBmax_k$ be the maximum data bandwidth quotas (GB per month) provided among federated datacenters.

Each cloud storage provider that hosts federated datacenters cooperatively proposes a

storage cost noted by OSC_k, and a induces local storage cost noted by LSC_d for the hosted intermediate data internally. A storage federation price varies according to busy storage or data bandwidth resources formulated by S_{busy}. Each federated datacenter provides maximum cloud storage quota noted by $QCmax_k$ to every users, and the busy quota is reduced from the maximum capacity of storage and bandwidth resources. In addition, cloud storage providers must rise to the variation of the price in real-time as regard to various factors such as data access demands, cloud storage market rates, and datacenter localization. To unlock these constraints, we adapt a pricing strategy [TCTB11] to determine the storage federation cost of the providers that varies according to their busy quotas in a way that when the busy quota is high the monetary cost goes up and when it is low the monetary cost goes down. This pricing mechanism allows to dynamically set their insourcing/outsourcing storage or transfer prices by establishing the monetary cost in exchange for offering data storage space or data bandwidth or selling storage resources. Below, the mathematical formula that determines the insourcing / outsourcing prices for the data placement demands is:

$$S = \frac{QCmax_k - S_{busy}}{QCmax_k} * (S_{price} - ME_{price}) + ME_{price} \tag{4.1}$$

Basically, the equation (4.1) considers the minimum effective price (ME_{price}) that is a reference monetary cost to the amount of data storage or data transfer that providers do not fall below in order to address economic issues. The affected cost (S_{price}) for the end-users is fixed and varies according to standard on-demand cloud storage pricing plan that is based on reservation contracts or prepaid schemes [MSS16]. Moreover, the maximum capacity of the storage federation $QCmax_k$ is given by the totality of storage or transfer quota offered by each datacenter. A very important point to consider during the intermediate data placement on the storage federation is a data transfer in or from other federation members. It should be mentioned that in most cloud storage services, the monetary cost of data transfer is more expensive than the data storage itself. This is an important cost factor that must be considered in our placement problem. Therefore, the transfer cost of data insourcing and data outsourcing are noted respectively by ITC_k and OTC_k. The outsourcing and insourcing storage and transfer costs (OSC_k, LSC_d, ITC_k and OTC_k) are updated using equation 4.1, and depend on their available outsourcing versus local capacity and the minimum effective storage price of each generated intermediate data file.

4.2.2 Matrix model for intermediate data dependency

As stated earlier, each federated datacenter k receives intermediate data placement requests from multiple home datacenters. When new intermediate data files are generated by an instance workflow, correlations between each file pair have revealed. These correlations correspond to the inter-file dependencies generated from several workflow instances. Then, we define a binary integer matrix denoted by DEP that including a symmetric dependency value ($Dep_{i,j}$) for each pair of files. Equation 4.2 exposes the $Dep_{i,j}$ values:

$$Dep_{i,j} = \begin{cases} 1 & \text{dependency between the two files } i \text{ and } j \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

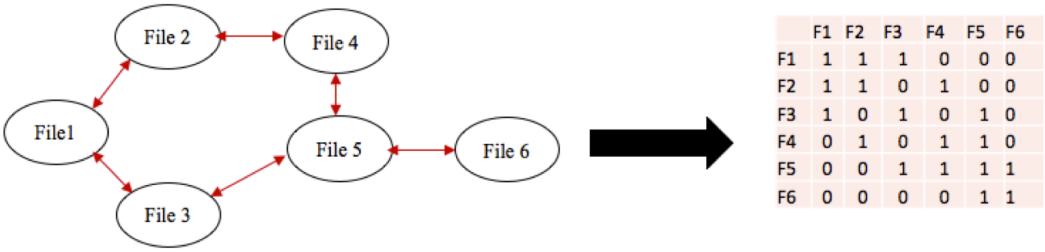

Figure 4.2 – Intermediate data dependency matrix.

Figure 4.2 illustrates an example of intermediate data dependencies and the corresponding matrix model. Each intermediate data file i has a dependency with itself and with some other file j. So if a set of intermediate data files ID_i that has a correlation with file j, this one is also correlated with the same set ID_i (symmetry). Therefore, the input parameters of the data placement model are the dependency values $Dep_{i,j}$ that are collected in matrix DEP.

In addition, various skewed of dependencies can be specified between the intermediate data during a freak execution of a workflow instance. Scientific users may encounter these errors [2] in the execution that cause unnecessary dependencies that need to be adjusted or re-run. Hence, some intermediate data dependencies are not valuable. In order to consider this situation, parameter λ_i^j is defined in the data placement model to denote scientific users tolerance of dependency files i and j:

$$\lambda_i^j = \begin{cases} 0 & \text{no tolerance to process } i \text{ and } j \text{ independently} \\ 1 & \text{otherwise.} \end{cases} \tag{4.3}$$

2. It can be an information about a replacement task, integrating a new input data that improves the reliability of the workflow execution.

The generated intermediate data dependencies with $\lambda_i^j = 1$, operate over a set of I/O requests between each other in the selected datacenter federation. These operations can be involved in remote access requests: data input adjustment, re-processing or data re-utilization. For each I/O access, there is a cost noted by $IOPC_{i,j}$. The values of the dependency matrix DEP is dynamically maintained for each set of generated files. Accordingly, between each pair of files i and j, the value of the λ_i^j parameter is defined. The amount and the size of intermediate data dependencies feeds the expected storage cost when scheduling intermediate data IDi on the federated locations. These later collaborate by sharing their respective storage resources and dynamically adjust their hosting capacities according to their intermediate data placement requirements.

4.3 Exact Algorithm

The proposed algorithm is an exact approach called Exact Federation Big Data Workflow Placement algorithm (ExactFed_BDWP) which leads to a mathematical programming approach based on an Integer Linear Programming (ILP). The ExactFed_BDWP algorithm is addressed and solved by a branch-and-bound method describing a set of valid inequalities of the big data workflow placement problem cited above. The proposed objective function is optimized under linear constraints. Some of these constraints are obtained according to a practical system in cloud data placement considering service scenarios and storage capacities. Finding the optimal placement requires the computation of storage cost for each possible instance solution from each intermediate data placement request (input parameters) on a federated and local datacenters. The federated datacenters use a cost model and ExactFed_BDWP algorithm according to the data dependency to schedule their intermediate data placement requests to the storage federation.

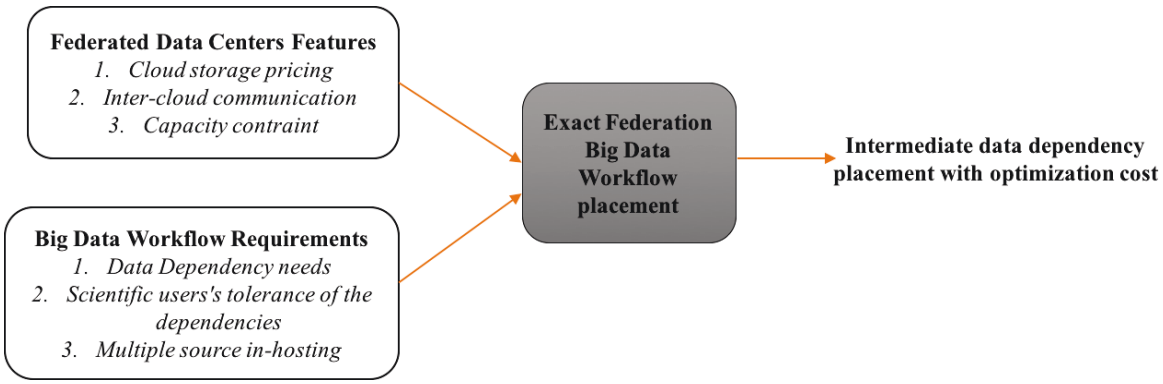

Figure 4.3 – Overview of data storage cost federation approach: input/output parameters.

As shown in Fig. 4.3, the storage selection decision has to lead to the minimum data storage cost in compliance with intermediate data dependency constraints (one datacenter hosting intermediate data dependency files) and maximum storage utilization for the federated datacenters. Hence, we discuss the cost model and the objective function that should be minimized considering the objective and the following procedures for the ExactFed_BDWP algorithm execution:

1. From a set of input parameters $Dep_{i,j}$ that considers a workflow instance processing intermediate data on home datacenters and make a data scheduling request to the storage federation, each datacenter from federation members exposes its insourcing/outsourcing storage cost based on the equation (4.1).
2. For each $Dep_{i,j}$ value, the cost of the newly calculated intermediate data placement solutions is compared with the currently lowest cost placement in each federated datacenter. The ExactFed_BDWP algorithm terminates after the set of all relevant intermediate data placement solutions have been checked. The ExactFed_BDWP algorithm is performed under the requirements of intermediate data owner and capacity constraints from the selected datacenters $DChome_d$ and from the federation members k, k'.
3. The ExactFed_BDWP algorithm keeps the intermediate data dependencies in a single datacenter while saving their storage, transfer and transaction costs. The intermediate data with dependency tolerance should be optimized according to the cost of I/O-demands requesting among federated datacenters.
4. The problem is solved on each workflow instance when a new intermediate data file is generated from a home datacenter. Such cost is assumed to change according to the federation features.

In the following, we introduce the use of the integer bivalent variables 0-1 x_{id}^{k} that tells which datacenter d hosts intermediate data file i to be placed in federated datacenter k. A glossary of all the used notations and their descriptions in the proposed exact model is shown in Table 4.1. Using these notations, the global objective function is given by equation (4.5):

$$MinCost = \sum_{idk}^{d \neq k} x_{id}^{k} \cdot size_i \cdot (OSC_k + ITC_k + OTC_d) + \sum_{idk}^{d=k} x_{id}^{k} \cdot size_i \cdot LSC_k + \sum_{ijdd'k}^{d \neq k} y_{ijdd'}^{kk'} \cdot Dep_{ij} \cdot \lambda_i^j \cdot IOPC_{i,j} \quad (4.4)$$

The objective function for the optimal scheduling and the placement of intermediate data

Table 4.1 – Table of notations.

Parameters	**Description**
P	Set of cloud storage providers.
D	Set of federated datacenters.
d, d', k, k'	Are used to designate home datacenters d, d' and federated datacenters k, k'.
$Dep_{i,j}$	Dependency coefficient value of intermediate data dependency as input to the model to designate affinity between files (i and j).
DEP	Dependency matrix including all dependency values of $Dep_{i,j}$.
$DChome_d$	A home datacenter matrix which brings home datacenter d that generates intermediate data files.
LSC_d	Local storage cost (dollar per GB) of cloud storage provider hosting datacenter d.
OSC_k	Outsourcing storage cost (dollar per GB) of cloud storage provider hosting datacenter k.
$size_i$	Size of intermediate data file i.
λ_i^j	Scientific users's tolerance of intermediate data dependencies of i and j which is a binary value, $\lambda_i^j = 0$ indicates scientific users have no tolerance to process i and j independently, and 1 otherwise.
i	Single intermediate data file.
ID_i	Set of intermediate data files.
IDN_d	Number of intermediate data generated and stored temporary in the datacenter d.
ITC_k	Insourcing transfer cost (dollar per GB) proposed by the provider to transfer data from home datacenter k.
OTC_k	Outsourcing transfer cost (dollar per GB) proposed by the provider to transfer data to another datacenter k'.
$IOPC_{i,j}$	Cost of I/O request (dollar per operation) of intermediate data files i and j on federated datacenters.
SCF_k	Storage space quotas offered and shared (GB per month) at datacenter k in the storage federation.
$SCmax_k$	Maximum storage space quotas (GB per month) offered and shared by all the federated datacenters.
SCL_d	Storage space quotas (GB per month) available at home datacenter d.
BCF_k	Data bandwidth (per data unit) quota offered and shared between home datacenter and destination datacenter to insource and outsource intermediate data storage.
$DBmax_k$	Maximum data bandwidth (GBs) quotas provided among federated datacenters.
Decision Variable	**Definition**
x_{id}^k	A binary variable, $x_{id}^k = 1$ if intermediate data file i is scheduled from datacenter storage d to outsourced datacenter k, and 0 otherwise.
$y_{ijdd'}^{kk'}$	A binary variable, $y_{ijdd'}^{kk'} = x_{id}^k * x_{jd'}^{k'}$.

can be expressed as the minimization of the cost of transferring, storing data workflow in the federated datacenters, where $x_{id}^{k} = 1$ is used to indicate that intermediate data file i is placed and $x_{id}^{k} = 0$ otherwise. Ideally, the ExactFed_BDWP algorithm should minimize the I/O transaction cost also when intermediate data dependencies are scheduled separately in a different datacenter, i.e. when $y_{ijdd'}^{kk'} = 1$. The objective function (4.5) is subject to several linear and integrity constraints (cited earlier) expressed respectively by equation (4.5) to (4.16):

Scheduling and placement constraint: For all intermediate data files ID_i generated and stored temporarily in each datacenter d, there is one or more outsourcing storage datacenters k in the federation that can store the set of files ID_i:

$$\sum_{ik} x_{id}^{k} = IDN_d \qquad \forall d \in D; \quad d \neq k \tag{4.5}$$

Hosting constraint: For each intermediate data file i generated and stored temporarily in datacenter d, there is only one datacenter hosting i:

$$\sum_{dk} x_{id}^{k} = 1 \qquad \forall i \in ID_i \tag{4.6}$$

Strong dependency constraint: For each coupled of generated intermediate data files i and j with $Dep_{i,j} = 1$ and with no dependency tolerance, ie. $\lambda_i^j = 0$, they are placed in the same federated datacenter k that has enough available volume:

$$x_{id}^{k} + x_{jd'}^{k} = 2 \qquad \forall i, j \in ID_i; \quad \forall k, d, d' \in D \tag{4.7}$$

Dependency splitting constraint: For each intermediate data dependency between i and j with a dependency tolerance $\lambda_i^j = 1$ that does not have available space from home datacenter d and d', they are placed in different federated datacenters:

$$x_{id}^{k} + x_{jd'}^{k} \leq 1 \qquad \forall i, j \in ID_i; \quad \forall k, d, d' \in D \tag{4.8}$$

Linearization constraint: To define relations between the bivalent variables $x_{id}^{k} * x_{jd'}^{k'}$ the following two constraints are defined:

$$x_{id}^{k} + x_{jd'}^{k'} - y_{ijdd'}^{kk'} \leq 1 \qquad \forall i, j \in ID_i; \quad i \neq j; \quad \forall k, k', d, d' \in D \tag{4.9}$$

$$\sum_{kk'dd'} y_{ijdd'}^{kk'} \leq \sum_{k} x_{id}^{k} \qquad \forall d \in D; \; \forall i, j \in ID_i \tag{4.10}$$

Storage capacity constraint: Each federated datacenter k has a storage space quota offered to the insourcing/outsourcing intermediate data placement:

$$\sum_{id}^{k \neq d} x_{id}^{k} \cdot size_i \leq SCF_k \qquad \forall k \in D \tag{4.11}$$

Each datacenter d has a storage space quota available to local intermediate data placement decision:

$$\sum_{id}^{k=d} x_{id}^{k} \cdot size_i \leq SCL_d \qquad \forall k \in D \tag{4.12}$$

Data transfer capacity constraint: Each federated datacenter has a data bandwidth quota offered to transfer intermediate data files from or out of the federated datacenter and the aggregation of these quotas cannot exceed the maximum data bandwidth $DBmax_k$ provided to the federation:

$$\sum_{id}^{k \neq d} x_{id}^{k} \cdot size_i \cdot BCF_k \leq DBmax_k \qquad \forall k \in D \tag{4.13}$$

Maximum capacity constraint in the federation: The placement of all generated intermediate data files ID_i cannot exceeded the total storage capacity of the federated datacenter $SCmax_k$:

$$\sum_{id} x_{id}^{k} \cdot size_i \leq SCmax_k \qquad \forall k \in D \tag{4.14}$$

Uniqueness constraint: Each intermediate data file i is generated from a single workflow instance and hosted in a single home datacenter d:

$$\sum_{d} IDN_d = 1 \qquad \forall i \in ID_i \tag{4.15}$$

Symmetry constraint: The dependency values of intermediate data matrix DEP are symmetric:

$$Dep_{ij} = Dep_{ji} \qquad \forall i, j \in ID_i; \quad \forall Dep_{ij} \in DEP \tag{4.16}$$

The optimal big data workflow placement model in a storage federation environment and objective function (4.4) can be summarized for convenience with all the valid conditions as problem 1:

$$\begin{aligned}\text{Min Cost} &= \sum_{idk}^{d\neq k} x_{id}^{k} \cdot size_i \cdot (OSC_k + ITC_k + OTC_d) \\ &+ \sum_{idk}^{d=k} x_{id}^{k} \cdot size_i \cdot LSC_k \\ &+ \sum_{ijdd'k}^{d\neq k} y_{ijdd'}^{kk'} \cdot Dep_{ij} \cdot \lambda_i^j \cdot IOPC_{i,j}\end{aligned}$$

Subject to:

$$\begin{array}{ll}
\sum_{ik} x_{id}^{k} = IDN_d & \forall d \in D, d \neq k \\
\sum_{dk} x_{id}^{k} = 1 & \forall i \in ID_i \\
x_{id}^{k} + x_{jd'}^{k} = 2 & \forall i,j \in ID_i, \forall k,d,d' \in D \\
x_{id}^{k} + x_{jd'}^{k} \leq 1 & \forall i,j \in ID_i, \forall k,d,d' \in D \\
x_{id}^{k} + x_{jd'}^{k'} - y_{ijdd'}^{kk'} \leq 1 & \forall i,j \in ID_i, i \neq j, \forall k,k',d,d' \in D \\
\sum_{kk'dd'} y_{ijdd'}^{kk'} \leq \sum_{k} x_{id}^{k} & \forall d \in D; \forall i,j \in ID_i \\
\sum_{id}^{k\neq d} x_{id}^{k} \cdot size_i \leq SCF_k & \forall k \in D \\
\sum_{id}^{k=d} x_{id}^{k} \cdot size_i \leq SCL_d & \forall k \in D \\
\sum_{id}^{k\neq d} x_{id}^{k} \cdot size_i \cdot BCF_k \leq DBmax_k & \forall k \in D \\
\sum_{id} x_{id}^{k} \cdot size_i \leq SCmax_k & \forall k \in D \\
\sum_{d} IDN_d = 1 & \forall i \in ID_i \\
Dep_{ij} = Dep_{ji} & \forall i,j \in ID_i, \forall Dep_{ij} \in DEP
\end{array}$$

Problem 1 – Big Data Workflow Placement Problem in Federation Storage Environment

4.4 Performance Evaluation

4.4.1 Setting Parameters

In order to evaluate the proposed model and to show the influence of using federated cloud storage characteristics, we performed a set of simulations with different input parameters. The evaluation model is performed under AMPL tools with CPLEX solver [3] as an ILP optimization program to solve the objective function (4.4). The assessment concerns the optimization cost of intermediate data dependency placement. To create a dynamic environment and unpredictable situations, we select randomly a number of geographical distributed datacenters ranging in [3,18] from three cloud providers. Among these datacenters, 8 are owned by

3. http://ampl.com/products/solvers/solvers-we-sell/cplex/

Amazon S3 (AM) [4], 4 by Google Cloud Storage (GO) [5] and 6 by Microsoft Azure (MZ) [6]. Each datacenter is restricted by storage and bandwidth capacities ranging from 10GB to 1000GB and from 1GB to 10GB respectively. Each outsourcing / insourcing demand is composed of a random inter-file dependency ranging from 50*50 to 1000*1000 organized in a matrix with 1 to 2 GB size per file. Intermediate data are affected to their home datacenter in a binary matrix ($DChome_d$). The insourcing/outsourcing storage and transfer monetary costs are given by equation (4.1). The I/O request cost and the affected monetary cost (S_{price}) for the end-users including OSC_k, ITC_k and OTC_k are set according to the pricing plan for each provider, Table 4.2 summarizes these different prices range. For an economic market purpose, the minimum effective price (ME_{price}) is set randomly and will be higher or close to the affected cost for the scientific community (ME_{price}=S_{price}*0,45). The binary value of λ_i^j is set randomly for each new generated intermediate data file.

Table 4.2 – Storage prices of the three cloud storage providers.

Prices —- Cloud storage providers	I/O cost ($/10000 operations)	Storage ($/GB)	Data transfer IN ($/GB)	Data transfer OUT ($/GB)
Amazon S3	[0.005-0.01]	[0.004-0.04]	[0]	[0-0.25]
Google Cloud Storage	[0-0.005]	[0.007-0.023]	[0]	[0.08-0.23]
Microsoft Azure	[0.015-0.0345]	[0.08-0.125]	[0]	[0-0.181]

4.4.2 Compared scenarios & performance metrics description

Since previous studies ([LHHH14, GHKS13, CPL16, ZLWC14, SHW+16, EA12, AKO08, NPM+10b, YS12, EMKL15, VOP11, ZXW16, YYL+12]) on data placement and cost saving of intermediate data dependencies in cloud storage federation differ and are not sufficiently close to the placement problem (see Sec. 2.3) that our approach deals with (the involvement of intermediate data dependencies at the lowest cost in the federated placement), we resort to a comparison with two following strategies: no-federation strategy on the one hand and a capacity-based placement strategy used in default Hadoop implementation [dSM15] on the other hand. Datacenters in the non-federation scenario turn in an autonomous way and depend on their own storage space resources to place intermediate data dependency. To elaborate this scenario, a relaxation of the ILP was built and consists in eliminating

4. https://aws.amazon.com/fr/s3/pricing/
5. https://cloud.google.com/storage/pricing
6. https://azure.microsoft.com/fr-fr/pricing/details/storage/

constraints (4.5), (4.7), (4.8), (4.11), (4.12) and (4.14). The intermediate data placement is scheduled entirely in each home datacenter thanks to the unlimited storage capacity (no loss). The outsourcing/ insourcing storage costs are obviously not integrated to solve the non-federation scenario considering just local dependencies. In a capacity-based scenario, the federated datacenters (nodes in the cluster Hadoop) randomly select the outsourcing storage to schedule the intermediate data files to the federation members only when their own resources are not available (nodes capabilities). Here, the selection is done arbitrarily to outsource intermediate data files without considering the dependencies (constraints 4.7 and 4.8).

We applied the following metrics to analyze the performance of the ExactFed_BDWP algorithm with the compared scenarios: (i) Total storage cost: this metric is defined by the objective function computed by equation 4.4 that measures the cost of transferring, storing and requesting intermediate data files to fulfill the evolving big data workflow requirements. This corresponds to the sum of all defined costs. (ii) Federation utilization: this metric shows the fairness of the intermediate data distribution on a selected datacenter in the federation. It is defined as the ratio between the amount of storage space used by intermediate data placement (both local and federation members) and the maximum amount of storage space for all intermediate data files placement. (iii) Convergence time: this metric measures the execution time of the ExactFed_BDWP algorithm in order to assess how fast the algorithm finds a solution to fulfill the intermediate data dependency placement.

4.4.3 Simulation results

Te performance of ExactFed_BDWP algorithm is evaluated relatively to the above scenario comparisons. The total storage cost evaluation of all algorithms through simulations is presented in Fig. 4.4 and 4.5. The results that show the fairness utilization of the federation are summarized in Fig. 4.6, 4.7 and 4.8. Finally, Fig. 4.9 and 4.10 depict the execution time of the ExactFed_BDWP algorithm. The obtained results on the figures below are a mean values delivered after the simulation of 15 runs including random input parameters (capacity and cost) averaged for each reported point (the confidence interval shows exact solution results with the different input parameters).

Figure 4.4 depicts the results of minimizing the total storage cost of the ExactFed_BDWP algorithm with no-federation and capacity-based scenarios while the number of datacenter is fixed to 9. Simulation results (100$, 137$ and 150$ respectively for the extreme case) correspond to the aggregation of those obtained for each federated datacenter that participates and receives outsourcing / insourcing storage demands of intermediate data placement with

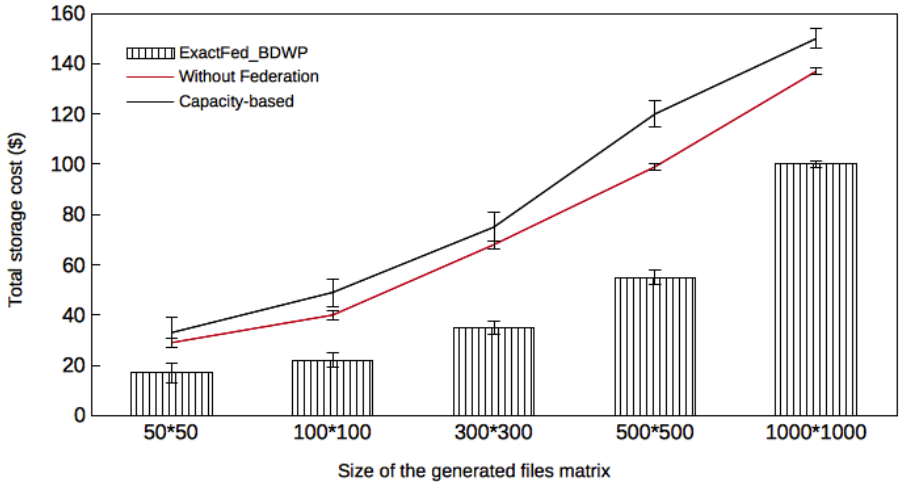

Figure 4.4 – Optimal total storage cost as regard to the dependency matrix size variation (DEP).

varying amounts of files (DEP matrix size variation). The general observation is that the ExactFed_BDWP algorithm testifies significant cost savings compared with the benchmark strategies. As expected, the results show that the ExactFed_BDWP algorithm outperforms the compared strategies with 27 % in average total storage cost saving for the non-federation scenario and 33.33 % compared to the capacity-based scenario while the matrix size reaches 1000*1000 corresponding to 1000 files of 1 GB. Figure 4.5 extends the cost saving evaluation for the ExactFed_BDWP algorithm by reporting performance as a function of dependency file pair size ($Dep_{i,j}$) while the number of datacenters is fixed at 9.

With the increase of the size of the matrix and the dependency files, the amount of stored intermediate data obviously increases in ExactFed_BDWP, non-federation and capacity-based algorithms (100$, 137$ and 150$ respectively), and this influences much more the two scenario comparisons. Admittedly, the cost of outsourcing file transfers is not included in the non-federation scenario, whereas the margin between their prices (insourcing prices) and those offered dynamically in the federation approach impacts on the total cost since the affected monetary costs are not negotiated and the non-federation scenario considers local dependencies and takes a fixed cost of home datacenter regardless to the price range. Similarly, for the capacity-based scenario that exhibits the highest average cost in both figures (4.4 and 4.5), their costs corresponding to insourcing/outsourcing transfer, storage and I/O cost substantially contrast on the total cost saving. The capacity-based scenario does not optimize the movement of intermediate data since it places them randomly to the different datacenters until the capacity is full without taking its dependencies in consideration. Al-

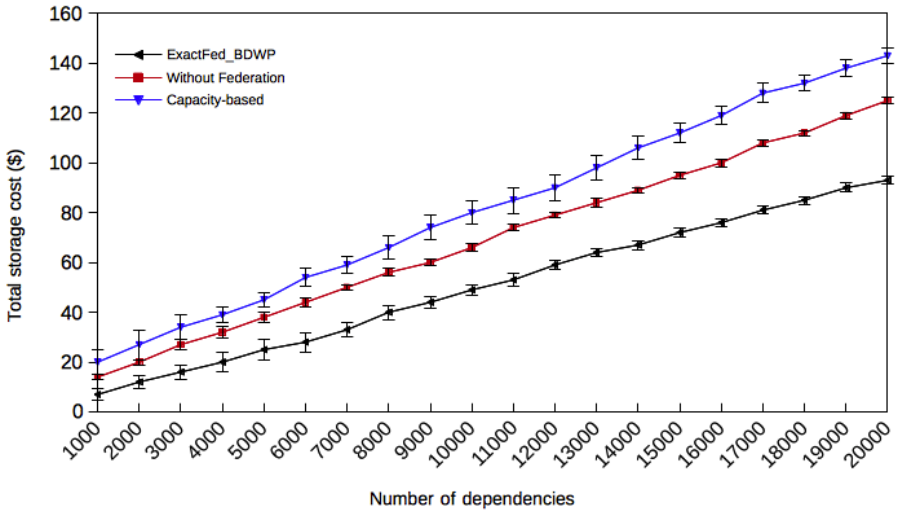

Figure 4.5 – Optimal total storage cost as regards to the number of dependency file pairs ($Dep_{i,j}$).

though the number of dependencies increases, the insourcing/outsourcing cost of intermediate data dependencies is minimized in the federation and this influences the total storage cost saving. In addition, the I/O request cost is minimized when there is a dependency tolerance.

The very important point in the federation is the placement balancing and fairness between federation members D for the intermediate data distribution. To achieve this, we set the amount of dependency files to 2000 GB (1000*1000 matrix size of 2 GB per file) that to be placed in federated datacenters and by both comparisons scenarios by setting randomly the home datacenters from provider P. The results are summarized in Fig. 4.6, 4.7 and 4.8 for the number of datacenters respectively of 6, 10 and 18 selected for files placement decision.

We clearly see that the intermediate data placement by the federated datacenter is the best balanced one for heterogeneous capacities and prices (with the exception of the unlimited capacities for no-federation scenario). The ExactFed_BDWP algorithm involves 6/6, 9/10 and 15/18 federated datacenters to be selected to schedule dependency files compared to 4/6, 5/10 and 5/18 for the capacity-based scenario and 3/6, 3/10, 5/18 only for the scenario without federation respectively in Fig. 4.6, 4.7 and 4.8. In fact, the storage federation contributes greatly and maintains the data distribution balancing among the members whatever cloud storage provider which participated (AM, GO and MZ). Moreover, the negotiated attractive prices influence for the placement decision as each provider tries to offer a dynamic pricing that balances the placement decision based on their capacity (equation 4.1) maintaining those collaborations among federated datacenter by fairly placing the intermediate data dependency

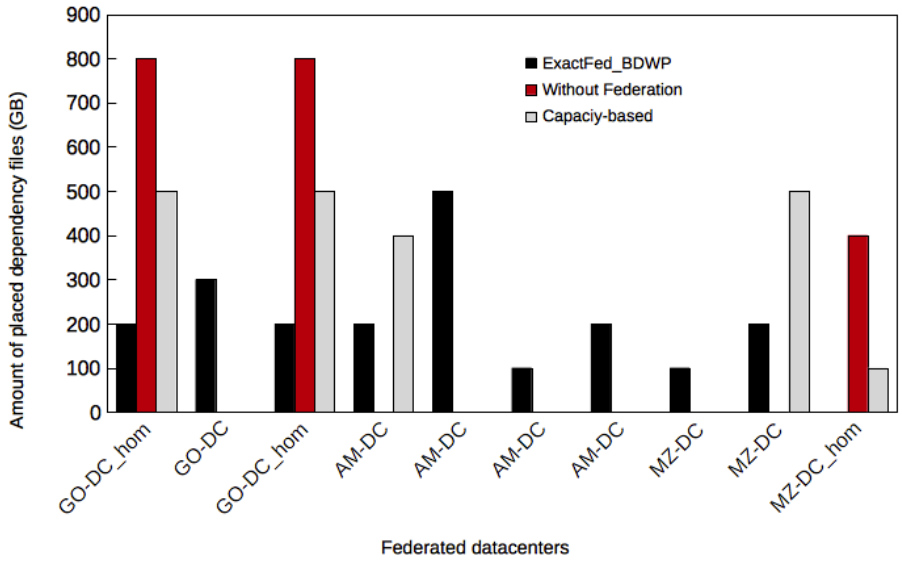

Figure 4.6 – Intermediate data distribution results for 6 federated datacenters.

distribution in the cloud federation thus reducing the charge for the scientific community. By contrast, as long as the no-federation scenario participates individually, the placement and storage cost is not optimal due to unlimited capacities (the entire amount of data stored is linear with respect to the expected cost) and fixed prices.

Figures 4.9 and 4.10 pursue the analysis for the ExactFed_BDWP algorithm time execution by reporting performance as a function of matrix size (1000 GB of intermediate data files) regarding to the federation size that ranges from 3 to 18 datacenters, and dependency size (obtained from the aggregation of dependency file pairs $Dep_{i,j}$) of 2000, 20000 and 200000 of dependency values (the number of datacenters is fixed at 9 for the results of Fig. 4.10). As the problem is NP-hard (limited by the branch-and-bound method), the execution time of the ExactFed_BDWP algorithm grows like the matrix size, especially when the number of datacenters is greater than 9 as reported in Fig. 4.9 (from 2 seconds to 4 minutes for all simulated instances). Figure 4.10 illustrates the evaluation results of the influence of dependency constraints (expressed in equation (4.7) and equation (4.8)) on the performance of the ExactFed_BDWP algorithm. This corresponds to the cases of: random default constraint (one could not force on the dependency constraint), splitting constraint (dependency files must be scheduled to different datacenter destinations), and strong constraint (dependency files must be scheduled to the same datacenter destinations) while the number of datacenters in the federation is set to 9.

For small number of dependencies (2000), the ExactFed_BDWP algorithm exhibits close

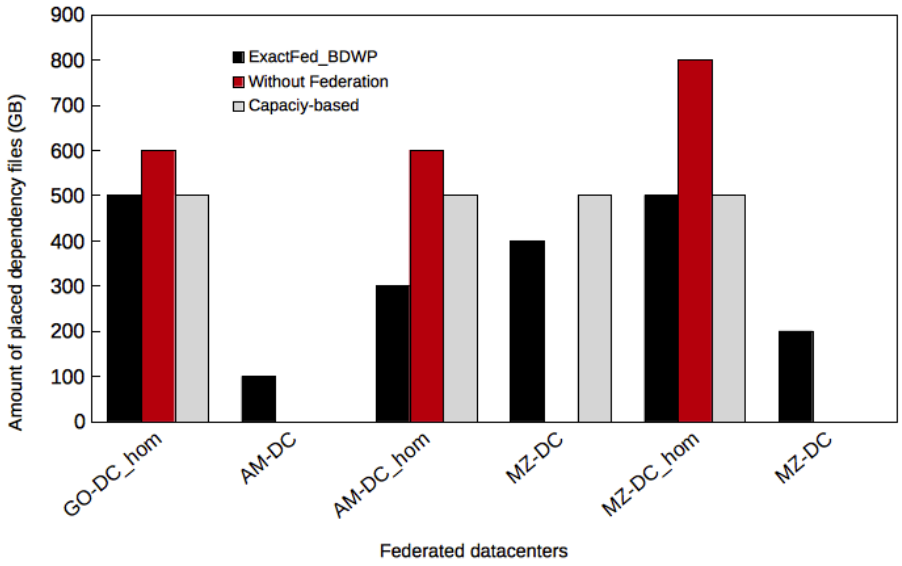

Figure 4.7 – Intermediate data distribution results for 10 federated datacenters.

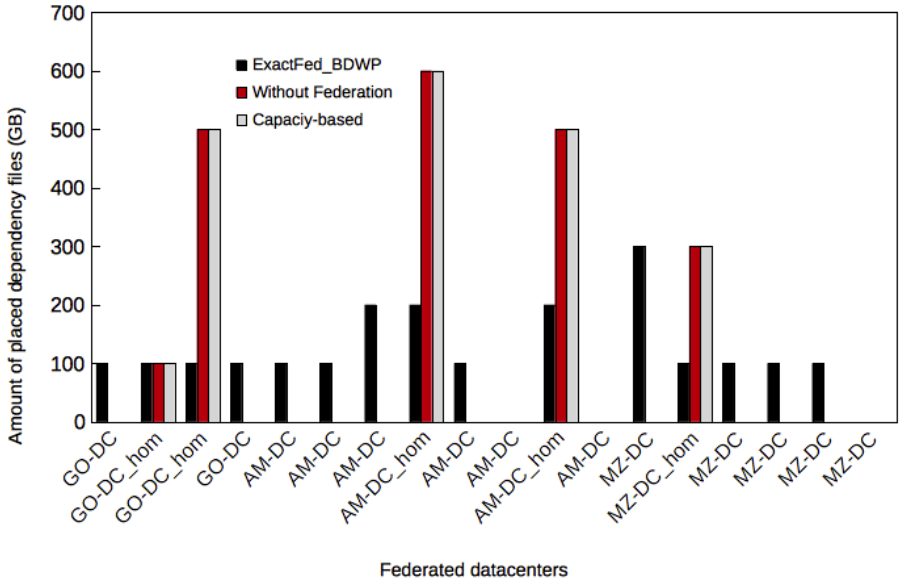

Figure 4.8 – Intermediate data distribution results for 18 federated datacenters.

performance irrespectively of the constraints. However, for a large number of dependencies (20000 and 200000), more important differences appear with splitting, default and strong dependency constraints standing out as an intermediate data dependency placement problem whose resolutions are ranging from one to fifteen minutes. With the splitting constraint case, the dependency file placement problem is solved faster as the space of feasible solutions is small (no inter-file dependencies to be considered in the resolution space). By contrast, in the

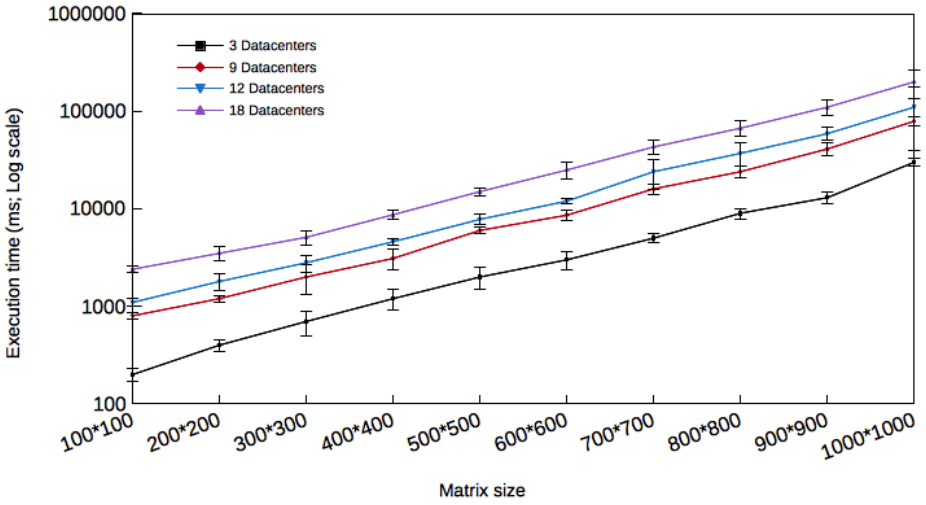

Figure 4.9 – Execution time of the ExactFed_BDWP algorithm with the different number of federated datacenters.

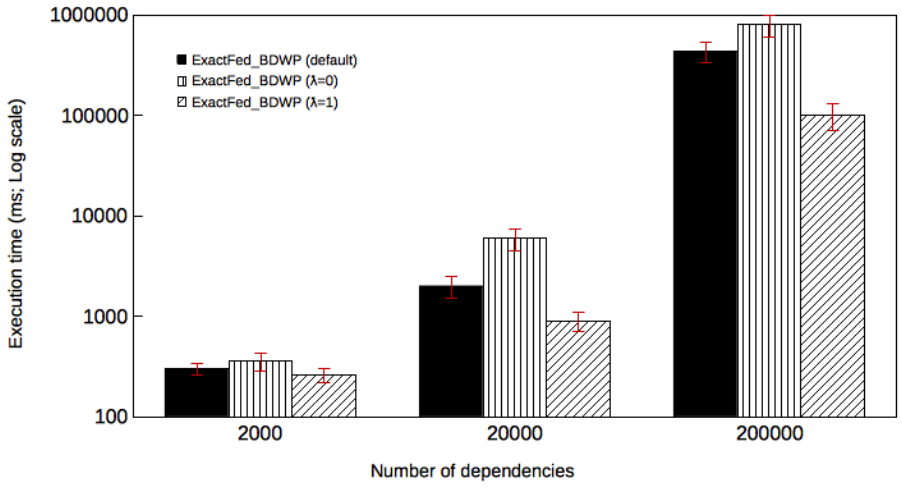

Figure 4.10 – Execution time of the ExactFed_BDWP algorithm with the different dependency parameter values λ_i^j while the number of datacenters is fixed to 9.

case of strong dependency constraint the placement problem becomes harder to solve because there is a huge inter-file dependency to be considered in the resolution space while satisfying all storage requirements.

4.5 Conclusion

The present chapter introduced intermediate data placement cost saving solution through a collaborative cloud storage environment. An exact federation algorithm (ExactFed _BDWP) based on an integer linear programming (ILP) model and the branch-and-bound method have been proposed to solve the problem of the inter-file placement that takes into account the storage federation characteristics. The ExactFed_BDWP schedules and places fairly intermediate data files taking into account their dependency requirements, size and cost saving over a distributed datacenter. A binary symmetric matrix is defined to represent the dependency for each pair of generated file in the same matrix, and home datacenter hosting the matrix are used to outsource intermediate data storage to the federation. The ExactFed_BDWP algorithm was tested and evaluated by the way of simulations on a set of data files generated randomly with two different scenarios. An effective and optimal solution in terms of total storage cost saving was shown by the ExactFed_BDWP algorithm against the other scenarios. The execution time of the ExactFed_BDWP algorithm increases as the size of the dependency matrix, and the number of datacenters involved in the federation. However, for small numbers of federated datacenters, the ExactFed_BDWP algorithm remains fast and achieves optimal scheduling and placement of intermediate data dependency.

The convergence time of the algorithm also matters in terms of swift response to additional data placement requests since some workflow applications require the placement of large data sets corresponding to a very complex dependency among a set of files, and can thus put very stringent requirements on extended data placement. Therefore, the next chapter proposes a new efficient and scalable heuristic algorithm based on network flow concepts to solve the data workflow placement problem faster.

Chapter 5

Scalable Cloud Big Data Workflow Placement Algorithms

Sommaire

5.1 Introduction

This chapter addresses the big data workflow placement to support their sharing and processing more efficiently in multiple cloud datacneters according to varying application dependency types and scientific users resource requirements. More specifically, it deals with a new approach that consideres inter and intra-job dependencies. In fact, the dynamic nature of data workflow applications, new trend of data files inter and intra-application generated in the cloud environment at any time reflects new types of data dependencies during the execution of a set of workflow applications. These dependencies are in the form of a set of correlation (clusters) between several files generated from multiple sources. They have different sizes and requirements for each type of data workflow. The need for usage and sharing of these data dependencies should be retrieved effectively by scientific collaborators who run a specific processing from multiple geographic locations. The change and frequency of use of these data dependencies mean that scientific users' needs change over time thus the entire placement

or appropriate amount of these clusters that will be placed in the different locations must include required changes. Therefore the cost inducing in their routing, storage and use also changes over time regarding to the amount of intermediate data including inter and intra-job dependencies .

The previous chapter treated, among other the problem differently that has introduced an ExactFed_BDWP approach to the intermediate data dependencies placement problem. This solution is optimal in minimizing the storage cost and performs well for small numbers of datacenters as well as for an amount of dependencies between file pairs, but exhibits long execution times for large-scale instances and does not handle no-symmetrical dependency (among group of files). To address larger scale problems and manage the plethora of requirements in data workflow applications and cloud environments, two algorithms are proposed to find optimal and near optimal solutions in polynomial time.

This chapter proposes a new approach that differents from previous works that do not take into account dependency types. Hence, the main contributions are summarized as follows. We first formulate the intermediate data placement dependency problem from multiple workflow applications [1] in a distributed datacenter. Then we propose an optimization model for the problem that deals with dependency constraints. The proposed model is combined with a total storage cost minimization by applying an exact and greedy heuristic algorithm while reducing the problem to the minimum cost multiple-source multicommodity flow problem respectively for intra and inter-job dependencies. Since the intermediate data routing from the nature of intra-job dependencies can be split and placed to different datacenters, the problem is called minimum cost multiple-sources splittable multicommodity flow. This allows to reduce the potential intermediate data movement cost among multiple datacenters. This approach deals with the placement problem of dependencies between a set of files that can be split during their placement that differs completely from the ExactFed_BDWP approach. Another approach refers to the intermediate data placement problem from inter-job dependencies. We formulate this problem as an unsplittable demand. However, as most of these problems are NP-hard, it is difficult to actually obtain an optimal solution based on exact methods. Besides, greedy approaches appear to be the simplest but effective algorithms for unsplittable flow problems [Kol03, Kry05, BBA07, CCGK07, PRF11], they are easy to implement and scale linearly with the number of instances. Thus, the use of greedy concepts yields to a good approximate solution to our intermediate data placement problem. Experimental results prove that the proposed algorithms are very promising in terms of total storage cost minimization as well as by showing that even with divergent conditions, the cost ratio of the heuristic algorithm is close to the optimal fractional solution.

1. An application refers to a job in data workflow processing.

The remainder of this chapter is organized as follow: Sec. 5.2 introduces the system model and problem definition according to the environment and data models. The proposed algorithms for the optimal intermediate data dependency placement are presented in Sec. 5.3. Sec. 5.4 discusses the performance evaluation under the implementation design and analysis simulation results of the proposed algorithms.

5.2 System model

5.2.1 Cloud storage infrastructure and assumptions

For the intra and inter-job data workflow placement problem depicted in Fig. 5.1, the objective is to route and store a set of intermediate data considering their dependencies generated by a collaborative tasks [2] from multiple physical sites while saving their transfer, movement and storage costs. Without any loss in generality, we assume that the collaborative tasks, that 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 placement are our most significant concern, the task assignment model can be assumed to be existing and simplified. The problem of placing the intermediate data files is close to the well-known minimum-cost multiple-source-flow problem as an optimization problem described in [Asa00] that involves simultaneously shipping multiple commodities through a single graph so the total flow obeys the arc capacity constraints by optimizing the cost.

With the stated objectives and requirements, the modeling starts by considering a set of geographically distributed datacenters [3] as a directed graph-based model $G = (DC \cup A, E)$ which forms a cloud infrastructure, and constructs a shared computation and storage limited set of resources for processing and storing the data workflow. Scientific 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 [WK09]. 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

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

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

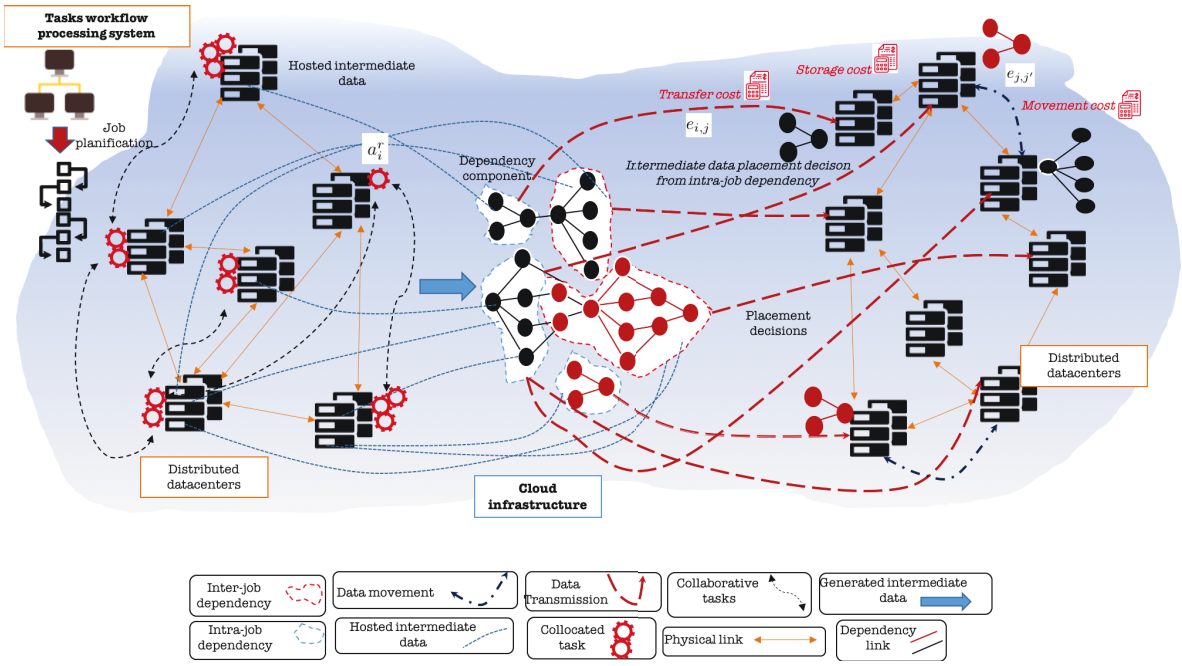

Figure 5.1 – The system model.

intermediate data routing respectively) links respectively between source datacenter dc_i and destination datacenter dc_j and between destination datacenters dc_j and $dc_{j'}$ that are geographically distributed around the world and 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.

5.2.2 Intermediate data dependencies graph-based model

Placing intermediate data with the same correlation to a single destination datacenter can significantly decrease the amount of data dependency movement [LD11]. This leads to consider a vector of all intermediate data files denoted by Φ^M and $|\Phi^M|$ its size, representing a 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 of 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 the Directed Acyclic Graph (DAG) which takes the advantage of a topology ordering, thus defining relations among nodes [WLN+13, LNW+11]. Figure 5.2 depicts the DAG representing 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 dependency are atomic and must be synchronized

for their processing. By contrast, for the intra-job dependency, these files are deduced from a partial correlation with an asynchronous processing [SRJ+16].

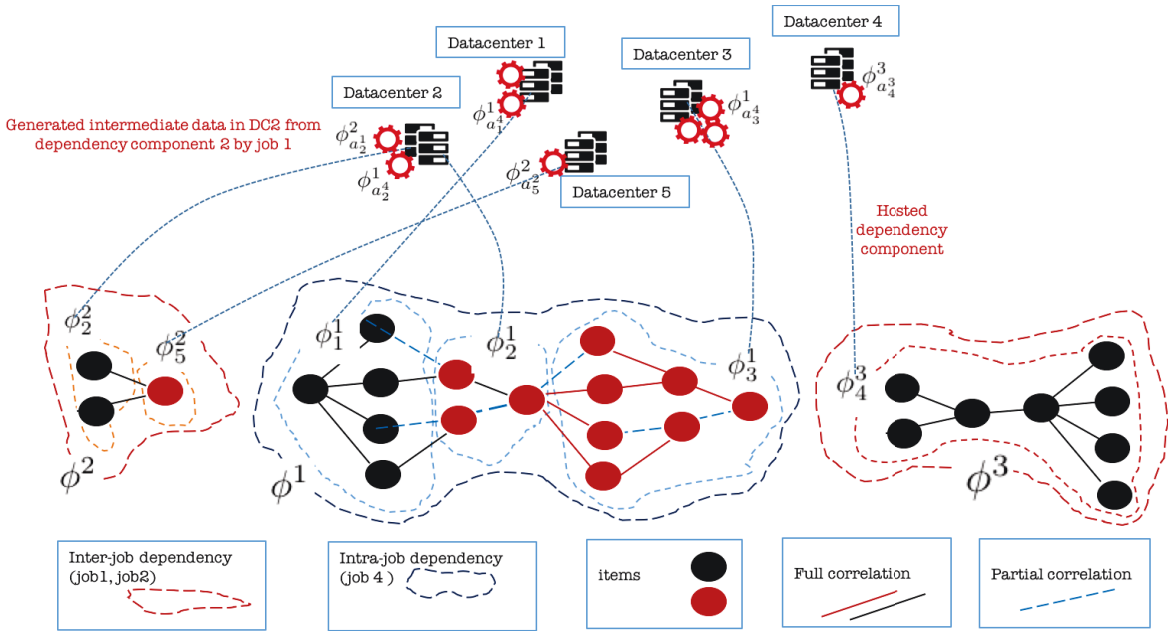

Figure 5.2 – DAG-based model for generated intermediate data files (intra and inter job dependency) from multiple source datacenters.

Let $\phi^m(t)$ and $\phi_i^m(t)$ 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 $|\phi_i^m(t)|$ be their respective sizes. At the end of each workflow phase, generated intermediate data file $\phi_i^m(t) \in \Phi^m$ 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 [WL15, JLS+11]. 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 [KR09]. We assume that intermediate data dependencies clustering is given a priori and is beyond the scope of the present work.

5.2.3 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 [RM15, AS14]. 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 by 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 $\phi_i^m(t)$, is denoted by $s_{i,j}^{avail}(t)$. Let $w_{i,j}^{avail}(t)$, $w_{j,j'}^{avail}(t)$ 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 inefficient utilization of its resources [ACC+14]. 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 [HSS+10]. 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 [ZXW16].

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 [XL15], 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 intermediate data dependency 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 1 summarizes the notations used in the present work.

5.3 Placement algorithms

In this section, the intermediate data dependency placement problem in a distributed dataceneter is reduced to a minimum-cost multiple-source multicommodity flow problem (MCMF) in G. Since two dependency types are conspicuous in a collaborative task workflow processing, two variants are materialized for the intermediate data dependency placement problem. In fact, in the case of intra-job dependencies type, the routing of intermediate data dependencies can be performed using multiple links. When this assumption is omitted, i.e.

Table 5.1 – 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
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 to place the intermediate data files
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 for one unit of intermediate data file $\phi_{a_i^r}^m(t)$
$W_{i,j}$	A 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

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 can by no means be fractionated. Thus, the MCMF problem seems to be hard to solve. For this aim, we formulate the intra-job splittable intermediate data dependency placement based on the LP approach, and we propose a heuristic approach as an approximation algorithm for the intra-job unsplittable intermediate data dependency placement.

5.3.1 Intermediate data placement in the case of intra-job dependencies

This section presents an exact analytical algorithm for splittable variant of the intermediate data dependency placement problem (SPL_LP). In this approach, we aim to provide an optimal routing that assists a placement of the intermediate data dependencies from collaborative workflow tasks from multiple datacenters in cloud infrastructures G.

The SPL_LP algorithm is a LP model through 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 are to route and place intermediate data dependencies $\phi^m(t) \in \Phi^M$ that is 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 5.1) are thus defined for the intermediate data problem in the case of intra-job dependency cited above:

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 for all t, 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 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 also, considering the same dependency component m at time

slot t. Formally:

$$\sum_{j \in DC} x_{i,j}^{m}(t) - \sum_{j \in DC} x_{j,i}^{m}(t-1) = \phi_{i}^{m}(t) \quad \forall m, t, i. \tag{5.1}$$

3) Capacity constraint of intermediate data flows: Each intermediate data dependency flow $x_{i,j}^{m}(t)$ 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 $\phi_{a_i^r}^{m}(t)$ on $x_{i,j}^{m}(t)$ of which all these flows take a same link $e_{i,j}$, hence:

$$0 \leq \phi_{a_i^r}^{m}(t) \leq x_{i,j}^{m}(t) \quad \forall i, j, a_i^r, m, t. \tag{5.2}$$

4) Capacity constraint of data transfer link: In G, each link $e_{i,j}$ may have a capacity constraint as data bandwidth routing constraint. Equation (5.3) 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:

$$\sum_{m \in M} w_{\phi} \cdot |\phi_{i}^{m}(t)| \cdot x_{i,j}^{m}(t) \leq w_{i,j}^{avail}(t) \quad \forall i, j, t. \tag{5.3}$$

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

$$\sum_{m \in M} \sum_{t \in T} w_{\phi} \cdot |\phi_{i}^{m}(t)| \cdot x_{i,j}^{m}(t) \leq W_{i,j} \quad \forall i, j. \tag{5.4}$$

5) Capacity constraint of data movement link: In G, each link $e_{j,j'}$ may have a capacity constraint as data bandwidth routing constraint. Equation (5.5) 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:

$$\sum_{m \in M} \sum_{a_i^r \in A} w_{\phi} \cdot |\phi^{m}(t) - \phi_{a_i^r}^{m}(t)| \cdot x_{j,j'}^{m}(t) \leq w_{j,j'}^{avail}(t) \quad \forall j, j', t. \tag{5.5}$$

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

$$\sum_{m \in M} \sum_{a_i^r \in A} \sum_{t \in T} w_{\phi} \cdot |\phi^{m}(t) - \phi_{a_i^r}^{m}(t)| \cdot x_{j,j'}^{m}(t) \leq W_{j,j'} \quad \forall j, j'. \tag{5.6}$$

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

corresponding component capacity. Formally:

$$\sum_{i \in DC} x_{i,j}^{m}(t) \leq \phi_{i}^{m}(t) \quad \forall j, m, t. \tag{5.7}$$

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:

$$\sum_{m \in M} |\phi_{i}^{m}(t)| \cdot x_{i,j}^{m}(t) \leq s_{i,j}^{avail}(t) \quad \forall i, j, t. \tag{5.8}$$

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

$$\sum_{m \in M} \sum_{t \in T} |\phi_{i}^{m}(t)| \cdot x_{i,j}^{m}(t) \leq S_{j} \quad \forall i, j. \tag{5.9}$$

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 as depicted in Fig. 5.3 (on page 87). Source and sink nodes are also subject to a constraint that enforces all the intermediate data dependency flows starting on s_{source} to ending s_{sink}. Formally:

$$\sum_{i \in DC} x_{s_{source},i}^{m} = \sum_{j \in DC} x_{j,s_{sink}}^{m} \quad \forall m \in M \tag{5.10}$$

9) Data transfer cost: Equation (5.11) denotes the data transfer cost on link $e_{i,j}$ which intermediate data dependency flows are routed.

$$C(w_{i,j}) = \sum_{i \in DC} \sum_{j \in DC} \sum_{m \in M} \sum_{t \in T} |\phi_{i}^{m}(t)| \cdot x_{i,j}^{m}(t) \cdot w_{\phi} \cdot c_{w_{\phi}} \tag{5.11}$$

10) Storage cost: Equation (5.12) denotes the storage cost of destination datacenter

dc_j which intermediate data dependency flows are routed. Formally:

$$C(s_j) = \sum_{i \in DC} \sum_{j \in DC} \sum_{m \in M} \sum_{t \in T} |\phi_i^m(t)| \cdot x_{i,j}^m(t) \cdot c_{s_j} \tag{5.12}$$

11) Data movement cost: The proportion 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:

$$C(w_{j,j'}) = \sum_{i \in DC} \sum_{j,j' \in DC}^{j \neq j'} \sum_{m \in M} \sum_{t \in T} |\phi^m(t) - \phi_{a_i^r}^m(t)| \cdot x_{j,j'}^m(t) \cdot w_\phi \cdot c_{w_\phi} \tag{5.13}$$

12) Objective function: The objective of the intermediate data placement problem is to find, for a given set of dependency flows $x_{i,j}^m(t)$, 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 following expression:

$$Minimize \ \ (C(w_{i,j}) + C(s_j) + C(w_{j,j'})) \tag{5.14}$$

The goal of solving data-task workflow placement as the minimum cost multicommodity flow problem is to minimize equation (5.14) under the constraints of equations (5.1) to (5.13) can be formulated as a Linear Program (See Problem 2), which is optimized with respect to flows $x_{i,j}^m(t)$. Under this formulation, problem 2 is a LP model and is thus polynomial. However, the optimization is carried out with respect to flows $x_{i,j}^m(t)$ that are bounded and constrained as a result of the amount of intermediate data dependencies $\phi_{a_i^r}^m(t)$ generated by single task a_i^r. This converges the SPL_LP algorithm into a non-polynomial time regarding to size $|\phi_{a_i^r}^m(t)|$ on large instances. The larger these atomic amount, the more difficult solving problem 2, since the splitting of flows $x_{i,j}^m(t)$ becomes marginal. Since a dependency component cannot start before the intermediate data dependencies of their predecessors is materialized, the unsplittable version of the intermediate data placement problem considering all flow for each dependency component from inter-job must be sent along a single link,

making the problem NP-hard [Asa00]. Due to the intractability of the placement problem from inter-job dependencies, we introduce a heuristic to address larger scale instances in a reasonable time.

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

Subject to:

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

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

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

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

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

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

$$\sum_{i \in DC} x^m_{i,j}(t) \leq \phi^m_i(t) \;\; \forall j, m, t$$

$$\sum_{m \in M} |\phi^m_i(t)| \cdot x^m_{i,j}(t) \leq s^{avail}_{i,j}(t) \;\; \forall i, j, t$$

$$\sum_{m \in M} \sum_{t \in T} |\phi^m_i(t)| \cdot x^m_{i,j}(t) \leq S_j \;\; \forall i, j$$

$$\sum_{i \in DC} x^m_{s_{source},i} = \sum_{j \in DC} x^m_{j,s_{sink}} \;\; \forall m \in M$$

$$C(w_{i,j}) = \sum_{i \in DC} \sum_{j \in DC} \sum_{m \in M} \sum_{t \in T} |\phi^m_i(t)| \cdot x^m_{i,j}(t) \cdot w_\phi \cdot c_{w_\phi}$$

$$C(s_j) = \sum_{i \in DC} \sum_{j \in DC} \sum_{m \in M} \sum_{t \in T} |\phi^m_i(t)| \cdot x^m_{i,j}(t) \cdot c_{s_j}$$

$$C(w_{j,j'}) = \sum_{i \in DC} \sum_{\substack{j,j' \in DC \\ j \neq j'}} \sum_{m \in M} \sum_{t \in T} |\phi^m(t) - \phi^m_{a^r_i}(t)| \cdot x^m_{j,j'}(t) \cdot w_\phi \cdot c_{w_\phi}$$

$x^m_{i,j}(t), x^m_{j,j'}(t)$: continuous

$\phi^m_{a^r_i}(t)$: discrete

Problem 2 – Splittable intermediate data dependencies placement problem.

5.3.2 Intermediate data placement in the case of inter-job dependencies

The intra-job intermediate data dependency placement is compared to the solution of the SPL_LP approach, and requires the placement of amount of intermediate data dependencies into a single destination datacenter. In order to deal with this case, a naive greedy solution considers an integer commodity of dependency component m from different sources as a single source flow unlike the SPL_LP 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 [Asa00]. The latter involving less connection, the local search of the optimum on a specific graph that reduces the search space accelerates the execution time of greedy solutions. As such, a greedy optimization framework is proposed in this section to represent the placement problem of inter-job intermediate data dependencies. Then, we develop an efficient greedy algorithm based on the proposed optimization framework, and analyze the time complexity of the proposed algorithm.

5.3.2.1 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 graph parameter c represented by cost function $E \rightarrow R$ and capacity function u: $E \rightarrow R$

The first part of the construction of the network flow graph G_p (see Fig. 5.3) concerns the assignment of the input flows from multiple sources. For each collocated tasks $a_i^r \in A$ that generate intermediate data $\phi_{a_i^r}^m(t)$ in the same source datacenter dc_i, there is a virtual source datacenter node $dc_i(\phi_{a_i^r}^m)_p$ in DC_p for each collocated task. For all generated intermediate data $\phi^m(t)$ from multiple collocated tasks belonging to the same dependency component $m \in M$ that are hosted temporarily in a single source datacenter dc_i in G, 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 component $\phi^m(t)$ hosting 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 $dc_i(\phi_{a_i^r}^m)_p$, $dc_i(\phi^m)_p$ 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 $\sum_{m \in M} \sum_{a_i^r \in A} \phi_{a_i^r}^m(t)$ hosted in the different virtual source datacenter nodes $dc_i(\phi_{a_i^r}^m)_p$. Source node s_{source} is connected with link $(s_{source}, dc_i(\phi_{a_i^r}^m)_p)$ in E_p to each $dc_i(\phi_{a_i^r}^m)_p$ and from this latter to $dc_i(\phi^m)_p$ represented by link $(dc_i(\phi_{a_i^r}^m))_p, dc_i(\phi^m)_p)$, involving cost $c(s_{source}, dc_i(\phi_{a_i^r}^m)_p) = 0$, as well as a link capacity demand that is assigned as the set of intermediate data dependencies $\phi_{a_i^r}^m(t)$ generated from each collocated task in the source datacenter at time slot t, i.e:

$$u(s_{source}, dc_i(\phi_{a_i^r}^m)_p) = u(dc_i(\phi_{a_i^r}^m)_p, dc_i(\phi^m)_p) = |\phi_{a_i^r}^m(t)|. \tag{5.15}$$

Link $(dc_i(\phi^m)_p, dc(\phi^m)_p)$ from each virtual dependency source datacenter node $dc_i(\phi^m)_p$

to the corresponding virtual dependency component node $dc(\phi^m)_p$ from the same dependency component $m \in M$ is added to G_p. Its cost is $c(dc_i(\phi^m)_p, dc(\phi^m)_p) = 0$, and its capacity is the amount of dependency component from all source datacenters that temporarily stored them, i.e:

$$u(dc_i(\phi^m)_p, dc(\phi^m)_p) = \sum_{a_i^r \in A} |\phi_{a_i^r}^m(t)| = |\phi_i^m(t)|. \tag{5.16}$$

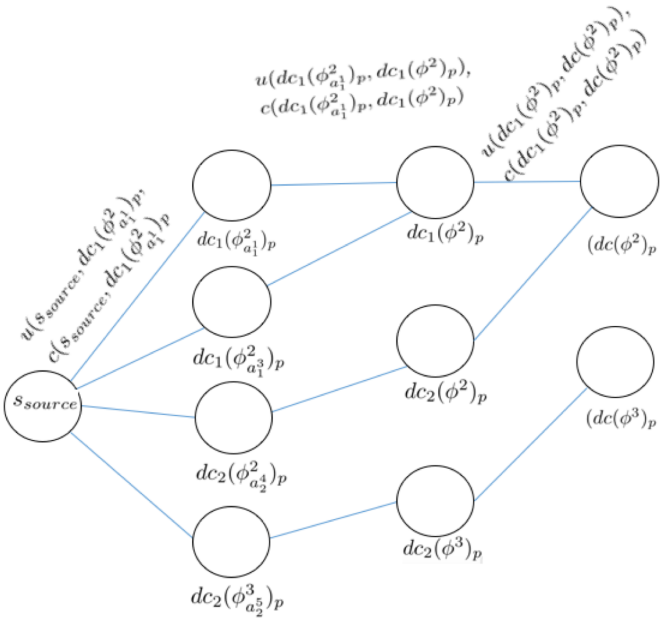

Figure 5.3 – The first part of the network flow graph construction G_p (two types of virtual dependency component nodes corresponding to three virtual dependency source datacenter nodes where four tasks are collocated in graph G.

The second part of the optimization framework deals with the identification of potential links for routing intermediate data dependencies to the destination datacenter. Thus, 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. All virtual destination datacenter node dc_{j_p} are 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 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} 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. We associate positive cost $c(dc(\phi^m)_p, dc_{j_p})$ along 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 dependencies $\phi^m_{a_i^r}$, i.e:

$$c(dc(\phi^m)_p, dc_{j_p}) = c_{w_\phi}(dc(\phi^m)_p, dc_{j_p}) + c_{s_j}(dc(\phi^m)_p, dc_{j_p}). \tag{5.17}$$

In addition to the cost of virtual link $(dc(\phi^m)_p, dc_{j_p})$, we assign capacity $u(dc(\phi^m)_p, dc_{j_p})$, which is the amount of intermediate data $\phi^m_{a_i^r}(t)$ 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 link $(dc(\phi^m)_p, dc_{j_p})$ is created in graph G_p, routing of the intermediate data dependency component $\phi^m(t)$ considers only the available amount of 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:

$$u(dc(\phi^m)_p, dc_{j_p}) = \frac{w^{avail}_{dc(\phi^m)_p, j}(t)}{w_\phi \cdot |\phi^m_{a_i^r}(t)|}. \tag{5.18}$$

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

$$u(dc_{j_p}, s_{sink}) = s^{avail}_{dc_{j_p}}(t) - |\phi^m(t)| \tag{5.19}$$

Figure 5.4 shows the representation of the generated directed flow graph $G_p = (DC_p \cup A_p; E_p; u; c)$

5.3.2.2 Greedy heuristic "UNS_GREED_HEUR" algorithm

This section proposes a greedy heuristic algorithm named UNS_GREED_HEUR for the minimum cost inter-job intermediate data dependency placement problem through the re-

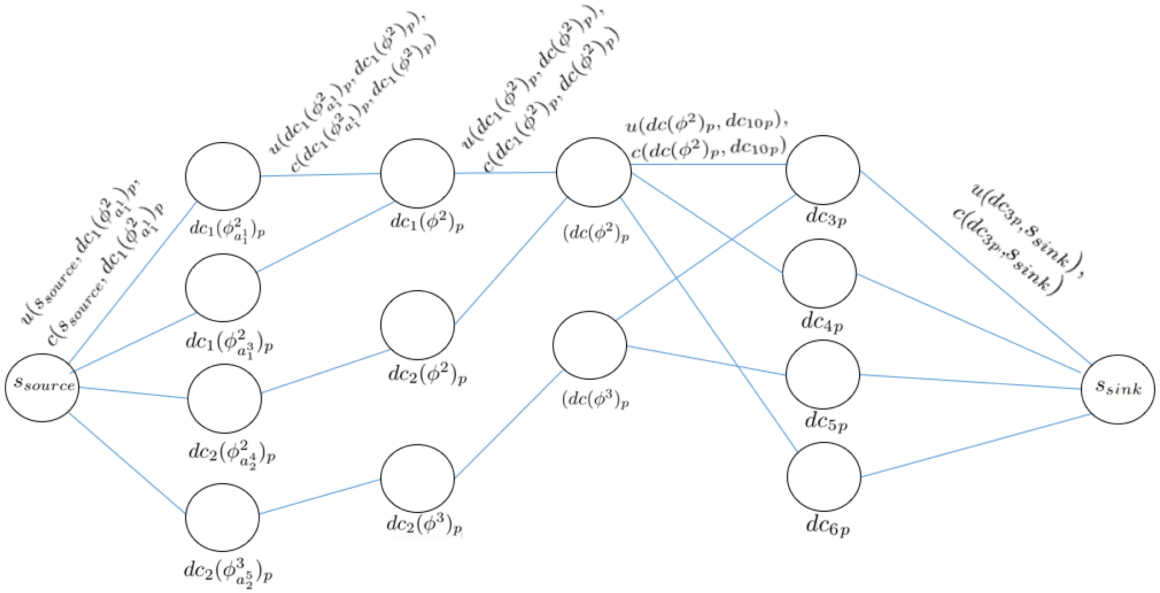

Figure 5.4 – The generated directed flow graph $G_p = (DC_p \cup A_p; E_p; u; c)$

duction to the minimum cost of unsplittable multicommodity flow with multiple dependency component sources in a flow graph G_p.

Let $S_{dc_j,min}$ be the minimum storage capacity of 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 cloud environment, it is realistic to assume assumption $|\phi^m_{max}| \leq S_{dc_j,min}$ from the construction of the flow graph G_p. Since a splittable exact algorithm is a relaxation of the unsplittable heuristic algorithm, we assume that a feasible solution for the splittable exact algorithm is fractional feasible flow f_0 that satisfies all demands of dependency component ϕ^m.

Since all dependency components are known a priori, so is their generation order. Hence, the UNS_GREED_HEUR 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 UNS_GREED_HEUR algorithm execution on a network flow graph G_p:

Step 1. Let $f(\phi^m)$ be the dependency component flows for all dependency intermediate data-task flows $\sum_{a_i^r \in A} f(\phi_{a_i^r}^m)$ with the minimum total storage cost from s_{source} to s_{sink}. Flows $f(\phi^m)$ route dependency component commodities $\phi_{a_i^r}^m$ from different virtual source datacenter nodes connected from source node s_{source} to their destination datacenter nodes $dc_j(\phi^m)_p$, and 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 ascending order of their respecting size as dependency component demands: $|\phi^1|$, $|\phi^2|$, ..., $|\phi^k|$, whith $\phi^1 \geq \phi^2 \geq \phi^3 \geq ... \geq \phi^k$. 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 the list L_ϕ. The algorithm scans each dependency component value $\phi_{a_i^r}^m(t)$ 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, $dc(0)_p \in DC_p \setminus \{s_{source}, s_{sink}\}$, these 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 values $f(\phi_{a_i^r}^m)$; Define a residual capacity $u_{res}(s_{source}, s_{sink})$ from s_{source} to s_{sink} in order to decrease the flows routed 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 $f(\phi_{a_i^r}^m)$ 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.

5.3.2.3 Time complexity

To build 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 latter 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 have 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 component 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 for finding solutions to the intermediate data dependency placement problem from multiple sources for unsplittable demand is $O(2M^2 + |DC|^2 \times (M^2 + 1) + M + |DC|)$ in the worst case.

5.4 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 UNS_GREED_ HEUR algorithm for the dependency intermediate data placement problem and compare it with the SPL_LP algorithm, random and uniform strategies, named RANDOM_HEUR and UNIFORM_HEUR respectively. The RANDOM_HEUR uses the data placement strategy as in default Hadoop scheduler [dSM15]. It is based on the idea that, upon the placement, the algorithm randomly selects a datacenter to host the intermediate data until its capacity is exhausted and then selects another one (random capacities and random costs). The UNIFORM_HEUR 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 [AJB11, YYL+12, RLZW16, ZXZ+15]. Subsequently, the performance evaluation overall intends to present relevant comparisons between the solutions found by the UNS_GREED_ HEUR algorithm with the optimal ones found by the SPL_LP algorithm in terms of performance metrics as optimality, scalability and convergence time. Additionally, a set of simulations have been reserved to compare the federation algorithm with the heuristic solution. Finally, the section concludes and comments the obtained simulation results.

5.4.1 Implementation details

The proposed UNS_GREED_HEUR solution is evaluated through a C++ language implementation using standard libraries with the g++ GNU compiler, version 2.30. The SPL_LP 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 UNS_GREED_HEUR and SPL_LP 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 that are represented in a directed graph including a list of adjacent nodes as source datacenters and destination datacenters which are connected to each other by a random movement links as different topology. This simulation program could be done through a set of tasks from collaborative processing jobs that process intermediate data and generate their dependencies beforehand. Moreover, 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 10GB to 100GB [ZXZ+15], including their dependencies that are generated randomly as correlation links in DAG from input to output intermediate data. The latter simply are assigned randomly to the set of source datacenters in charge of temporarily storing them.

The type of intra-job dependency is described by probability 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. We give also dependency parameter value β generated randomly from range [1, 20] which represents the number of clusters randomly grouping intermediate data-task. For the inter-job dependency, we set probability parameter value 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 values β). On all the experiments carried out, we exclude the case when $\alpha = 1$ and $\beta = 20$ which means that the intermediate data are completely independent. The same dependency parameter values β are assigned for both intra-and inter-job dependencies according to each experiment.

We can see that the arrival rate of the intermediate data is lower then their dependencies to

real-like driving big-data workflow situations. In addition, the simulation test are conducted for 48h in order to validate the need of the UNS_GREED_HEUR algorithm and to estimate the probability to have a good solution for the intermediate data dependency placement problem.

We also considered both capacity and cost of cloud infrastructures, and among other things, the storage space capacity of the datacenters which are randomly set from range [10GB, 1000GB] [EMKL15], and the transfer link capacity of one unit of intermediate data transmission between distributed datacenters from the initial problem to the solution which are randomly drawn from range [1, 10] Gbps [XXLZ16] with a random transfer cost (in $) ranging from 0 to 0.09. Both storage and transaction costs (in $) of one unit of intermediate data dependencies are set within [0.02, 0.04] and [0, 0.09] respectively, in relation to the typical charges in Amazon S3 [4].

5.4.2 Simulation results

In this section, we first present the performance evaluation result of the proposed algorithms (splittable and usplittable) regarding the effectiveness of the two solutions against comparison scenarios. Second, we expose the comparison results between the federation and heuristic algorithms (ExactFed_BDWP and UNS_GREED_HEUR). To this end, we study the optimality of the UNS_GREED_HEUR algorithm in terms of the total storage cost ratio. Finally, we explain the results of the scalability and the convergence time of the UNS_GREED_HEUR heuristic, and the limit beyond which the SPL_LP and ExactFed_BDWP algorithms become unfeasible for scaleable cloud infrastructure.

5.4.2.1 Impact of the amount of routed intermediate data on the performance of unsplittable and splittable placement algorithms

First we investigate the performance of UNS_GREED_HEUR and SPL_LP algorithms with the comparison scenarios, and weighed them with the total storage cost which they induce. For the specific needs of this simulation, we vary the amount of intermediate data that 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 these experiences, we reflect the concerns of dependency parameters on the behavior of the proposed algorithms. In this case, each solution found in the algorithms is a mean of the results obtained by varying dependency parameters

4. https://aws.amazon.com/fr/s3/pricing/

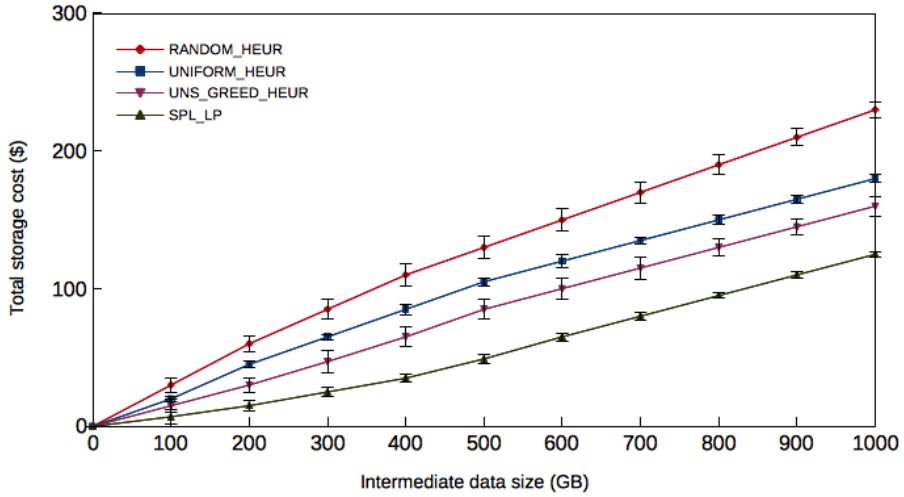

Figure 5.5 – The total storage cost of Algorithms UNS_GREED_HEUR, SPL_LP, RANDOM_HEUR and UNIFORM_HEUR by varying the intermediate data size while the number of datacenters is set to 50.

α and β from range [0, 1] and [1, 20] respectively, as defined above.

Figure 5.5 depicts the curves of total storage cost delivered by the proposed algorithms and the two other heuristics. The figure shows that both UNS_GREED_HEUR and SPL_LP algorithms outperform algorithms RANDOM_HEUR and UNIFORM_HEUR 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 1000GB, and the UNS_GREED_HEUR algorithm achieves a nearly optimal storage cost of \$160, which is lower than the costs of the RANDOM_HEUR and UNIFORM_HEUR algorithms (43% and 12% respectively). Clearly, the gap between UNS_GREED_HEUR and UNIFORM_HEUR algorithms is very small since the UNIFORM_HEUR is independent of the capacity of cloud infrastructure, so the cost at time slot within the datacenters contrast within the placement decision.

Figure 5.6 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).

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 UNS_GREED_HEUR, SPL_LP, RANDOM_HEUR and UNIFORM_HEUR to \$4900, \$3300, \$7000 and \$5400 respectively. Typically this means that the cost of algorithm UNS_GREED_HEUR is 10% and 42% less than those of UNIFORM_HEUR and RANDOM_HEUR algorithms, while the result of SPL_LP algorithm as expected remains the best total storage cost. These results show that the uni-

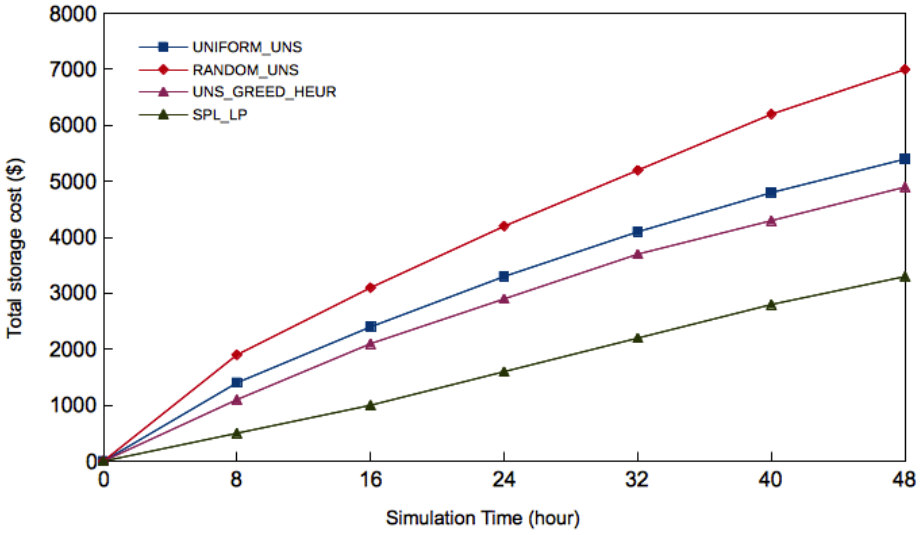

Figure 5.6 – Total storage cost of Algorithm UNS_GREED_HEUR, SPL_LP, RANDOM_HEUR and UNIFORM_HEUR when the simulation time is extended to 48h, while the number of datacenters is set to 50.

form capacity constraint of the scale of intermediate data directly affects their placement cost as in the case of UNIFORM_HEUR algorithm. Otherwise, by excluding a some datacenters offering the lowest cost because of their inability to host data an/or random selection as in the case of RANDOM_HEUR algorithm.

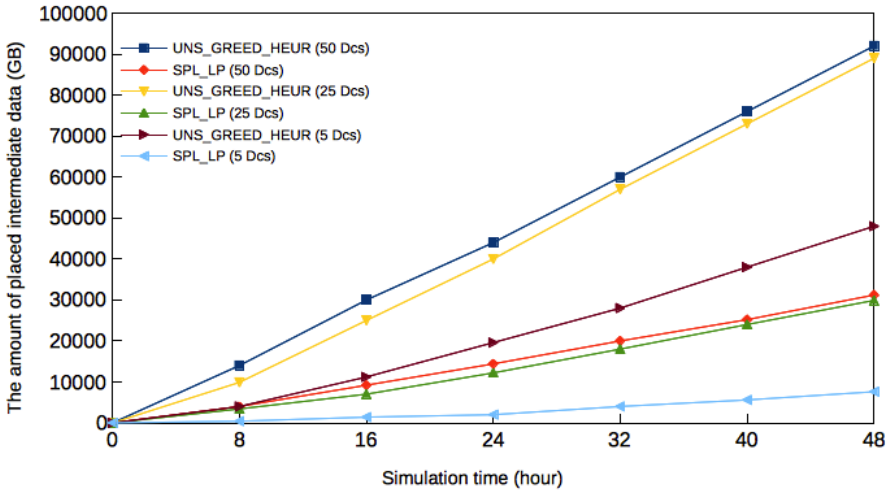

Figure 5.7 – The amount of intermediate data accumulated per time slot for the unsplittable and spilttable algorithms, datacenter number ranging from 5 to 50.

The aim of the simulation results depicted in Fig. 5.7 is to quantify the amount of intermediate data placed continuously per time slot by varying the number of datacenters

from 5 to 50 along with their capacities. The lengthening of the simulation at time slot 48 shows that 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 UNS_GREED_HEUR 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 SPL_LP algorithm. In contrast, with the data bandwidth capacity defined in the UNS_GREED_HEUR algorithm that is shared by a single dependency component. However, in the SPL_LP algorithm, the amount of placed intermediate data is less in comparison with the UNS_GREED_HEUR solution, because it expands the search space for the exact solution since it is based on the simplex method. Thus, the UNS_GREED_HEUR algorithm handles large datacenter instances for which the SPL_LP algorithm has more difficulty to find solutions as an amount of intermediate data placement as regard to the lower total storage cost.

5.4.2.2 Impact of dependency parameters on the performance of unsplittable and splittable placement algorithms

In this section, we study the impact of dependency parameters α and β on the performance of unsplittable and splittable placement algorithms in terms of optimal cost. Since the types of intermediate data dependency that are processed by the two proposed algorithms are different, we need a variation of a quantitative value to compare the proposed algorithms for achieving a useful analysis and allowing optimal cost to the unsplittable placement solutions to be more efficiently identified. For this purpose, interval values are considered to linearize the two types of dependency. When dependency types diverge (in arguing the amount of intermediate data dependency represented by α in correlation with that represented by β). Furthermore, assessment scenarios of the proposed algorithms correspond to varying dependency parameters (α, β) pair values. Simulation results according to the following pair ranges of (α, β)= (0.1, 18), (0.3, 14), (0.5, 10), (0.7, 6), (0.9, 2) are reported on figures below, keeping the value of α to 0 and with same dependency values β for the UNS_GREED_HEUR algorithm while the number of datacenters is set to 50.

Figure 5.8 depicts the best optimal cost achieved by the objective function for SPL_LP and UNS_GREED_HEUR solutions. The UNS_GREED_HEUR algorithm performs very well close to the optimal one and achieves a cost of $3000 at time slot 48 that is near to

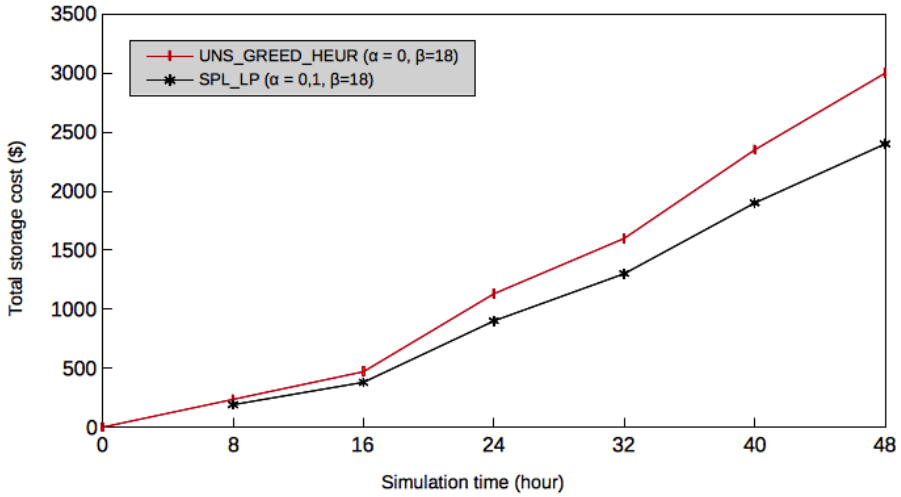

Figure 5.8 – UNS_GREED_HEUR heuristic versus SPL_LP exact solution for the total storage cost when $\alpha = 0.1$ and $\beta = 18$.

the optimal result of \$2400 for the SPL_LP algorithm. This is due to the fact that the behavior of the two algorithms have to deal with the same types of correlation $\alpha = 0.1$ (practically, no amount of intermediate data is splitted with SPL_LP, and 0 defaults to the UNS_GREED_HEUR) with $\beta = 18$ (divergence among intermediate data dependencies). Therefore, this has a direct impact in the reduction of transfer, storage and movement costs for both algorithms.

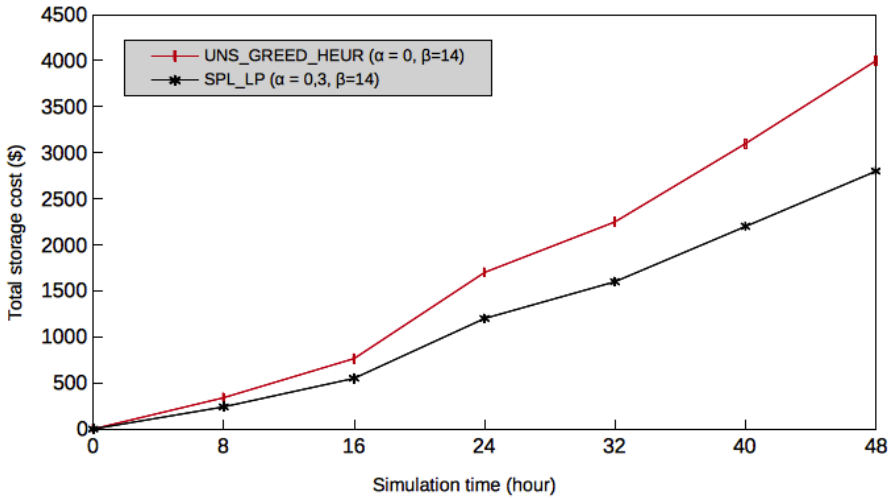

Figure 5.9 – UNS_GREED_HEUR heuristic versus SPL_LP exact solution for the total storage cost when $\alpha = 0.3$ and $\beta = 14$.

Figures 5.9 and 5.10 depict the second best-case results for the total storage cost which do

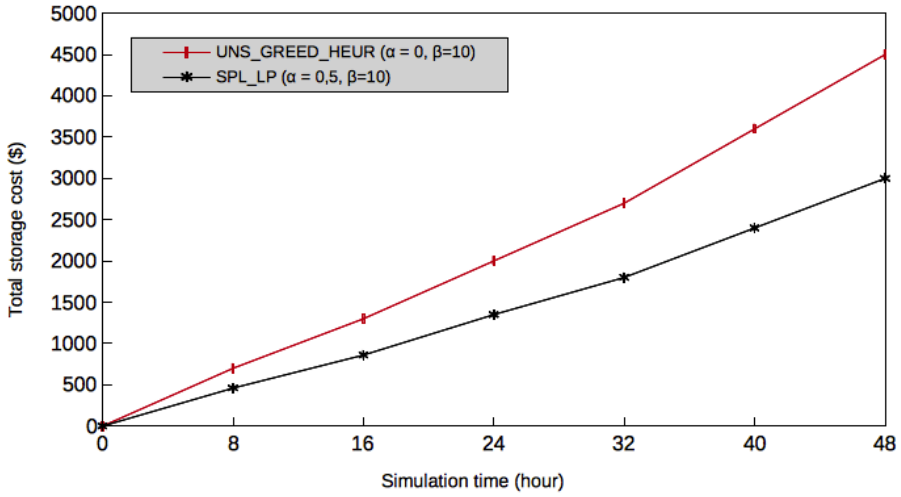

Figure 5.10 – UNS_GREED_HEUR heuristic versus SPL_LP exact solution for the total storage cost when $\alpha = 0.5$ and $\beta = 10$.

not exceed \$2800, \$3000 for the SPL_LP algorithm, and \$4000, \$4500 for the UNS_GREED_HEUR algorithm at time slot 48. Since, the amount of movement of intermediate data dependencies are marginal to half ($\alpha = (0.3, 0.5)$) in the SPL_LP algorithm, the movement cost is reduced, which reflects the total storage cost. In addition, the UNS_GREED_ HEUR 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.

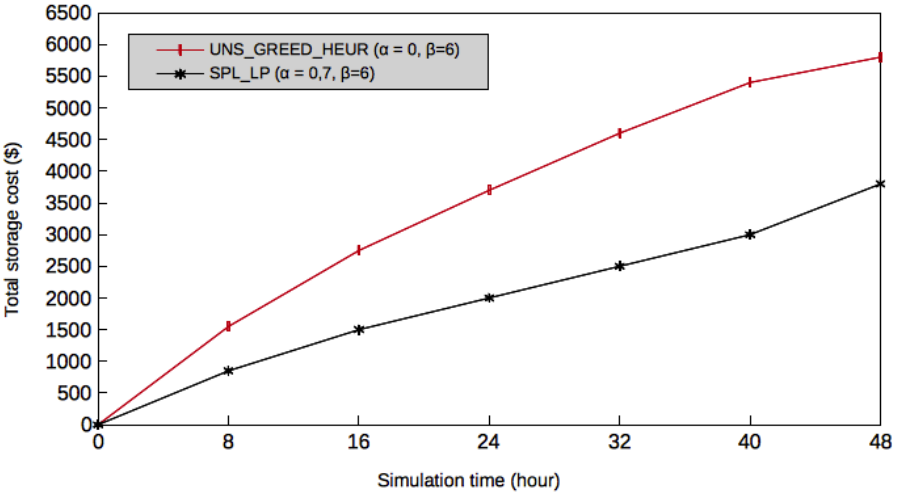

Figure 5.11 – UNS_GREED_HEUR heuristic versus SPL_LP exact solution for the total storage cost when $\alpha = 0.7$ and $\beta = 6$.

In the continuity of the comparison between unsplittable and splittable solutions, Fig. 5.11 shows the case when the amount of movements of intermediate data dependencies increases more than half ($\alpha = 0.7$) with a growing volume of intermediate data dependencies ($\beta = 6$). This gives a total storage cost of \$3800 and \$5800 respectively for SPL_LP and UNS_GREED_HEUR algorithms. It can be seen that the total storage cost of the two algorithms dependents on the amount of intermediate data dependencies. This validates our analysis of the results discussed above (Figures 5.8, 5.9 and 5.10). Fig. 5.12 shows a real compromise in which the highest total storage cost is found for both algorithms. The total storage cost reaches \$4200 and \$7200 for SPL_LP and UNS_GREED_HEUR algorithms respectively at time slot 48, since the amount of intermediate data dependencies that transit between destination datacenters is significant ($\alpha = 0.9$) for the SPL_LP algorithm (defined by variable $x^{m}_{j,j'}(t)$). In contrast, the UNS_GREED_ HEUR algorithm processes more intermediate data dependencies grouped onto two clusters. In addition, the same capacity values were considered for each solution found with the different values of α and β. Therefore, it has less opportunity to find datacenters that have the capacity to allocate those large clusters and at the same time offer a better cost. This influences considerably the search for the optimal result which is a real compromise for both unsplittable and splittable algorithms.

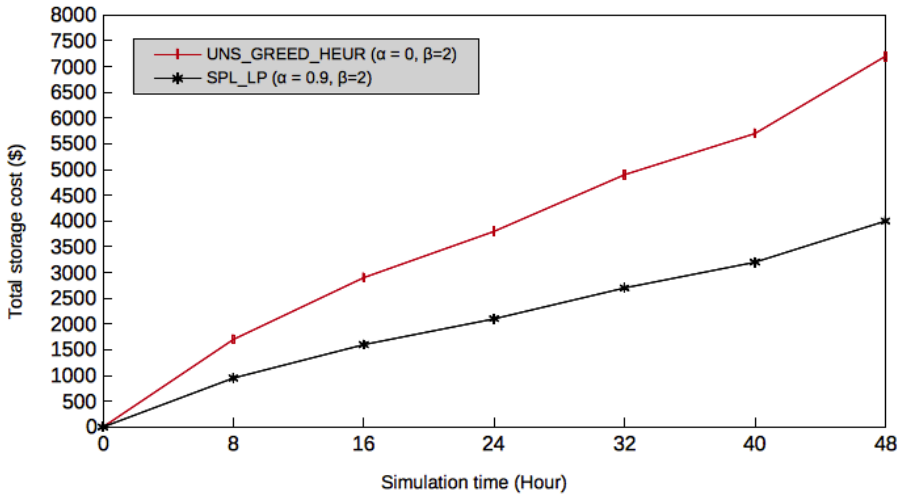

Figure 5.12 – UNS_GREED_HEUR heuristic versus SPL_LP exact solution for the total storage cost when $\alpha = 0.9$ and $\beta = 2$.

The performance of the UNS_GREED_HEUR algorithm as compared to the SPL_LP fractional optimal solutions in terms of total storage cost is represented as a cost ratio between the cost delivered by 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 splittable

Table 5.2 – Gaps between UNS_GREED_HEUR heuristic and SPL_LP algorithms in term of cost ratio

DC \ $(\alpha;\ \beta)$	(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

variant of the placement. The cost ration of the heuristic $HEUR$ is $\epsilon = \frac{HEUR}{FRAC\_OPT}$. The cost ratio of the different curves above (Fig. 5.8 to 5.12) is reported in Table 5.2 when the number of datacenters varies from 5 to 50.

It can be seen that the cost ratio of the UNS_GREED_HEUR algorithm is no more than 1.85. Indeed, for simulated instances in the ranges 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 heuristic algorithm performs close 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 heuristic algorithm encounters some difficulties in finding an optimal solution. Thus, the cost ratio of UNS_GREED_HEUR algorithm reaches 1.81. Mind that, in the UNS_GREED_HEUR algorithm, we assumed the feasibility of the solution by scaling datacenter capacities, thus there is solution to the problem when $\beta = 2$. The cost ratio of the UNS_GREED_HEUR algorithm in this case reaches 1.85, which diverges considerably from the optimal solution, as a condition for finding any solutions that match 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 restrictions. Indeed, each proposed algorithm responds differently to the dependency requirements as well.

We also considered special cases 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 SPL_LP 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 UNS_GREED_HEUR 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 (α, β) = {(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 UNS_GREED_HEUR algorithm that is reported in Table 5.2 increases as dependency parameters deviate (α, β) = {(0.7, 6); (0.9, 2)}.

5.4.2.3 Convergence time of unsplittable and splittable placement algorithms

To pursue the extensive experiments, we evaluate the effectiveness of our algorithms and compare them in terms of scalability and convergence time from input parameters. For this 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 100GB to 1000GB. Obviously, the values of the dependency parameters must also varying in order to better understand the behavior of the execution time of the proposed algorithms regarding to these dependencies. Thus, the value of dependency parameters is set as specified in Sec. 5.2.2. Algorithm running times are recorded as follows.

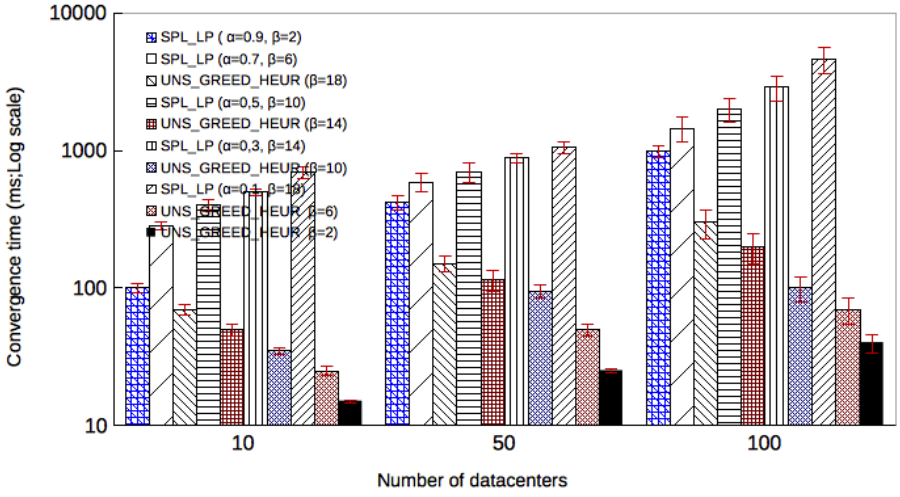

Figure 5.13 – Time execution comparison between UNS_GREED_HEUR and SPL_LP algorithms for different datacenter size when the amount of hosted intermediate data are 100GB.

First, the running time of the UNS_GREED_HEUR algorithm solves our intermediate data dependency placement problem, one to four orders of magnitude faster than the SPL_LP exact solution on the experiments which amply depend on the dependency parameters. However, the SPL_LP algorithm solves the NP-hard problem in exponential time for the 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 as 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, UNS_GREED_HEUR algorithm is much easier to solve than the SPL_LP algorithm.

Indeed, Fig. 5.13 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 UNS_GREED_HEUR and SPL_LP algorithms respectively. For datacenter sizes which is below 50, the convergence time remains fairly reasonable too, with less than 0.15 and 1.05 seconds for UNS_GREED_HEUR and SPL_LP algorithms respectively. For the latter, it slightly increases when the number of datacenters is beyond 100 (about 5 seconds). In fact, the SPL_LP algorithm performance gradually degrades with input network topology and exponentially grows for wide range (not shown in Fig. 5.8). The following figures (5.14 and 5.15) show these facts.

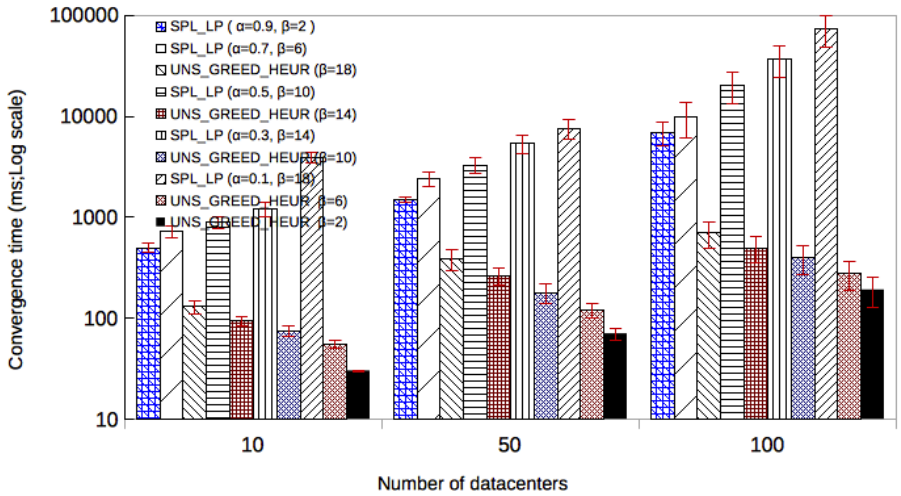

Figure 5.14 – Time execution comparison between UNS_GREED_HEUR and SPL_LP algorithms for different datacenter sizes when the amount of hosted intermediate data are 500GB.

Figures 5.14 and 5.15 shows the worst cases for the SPL_LP algorithm. The time needed for convergence grows mainly for an amount of placed intermediate data that varies between

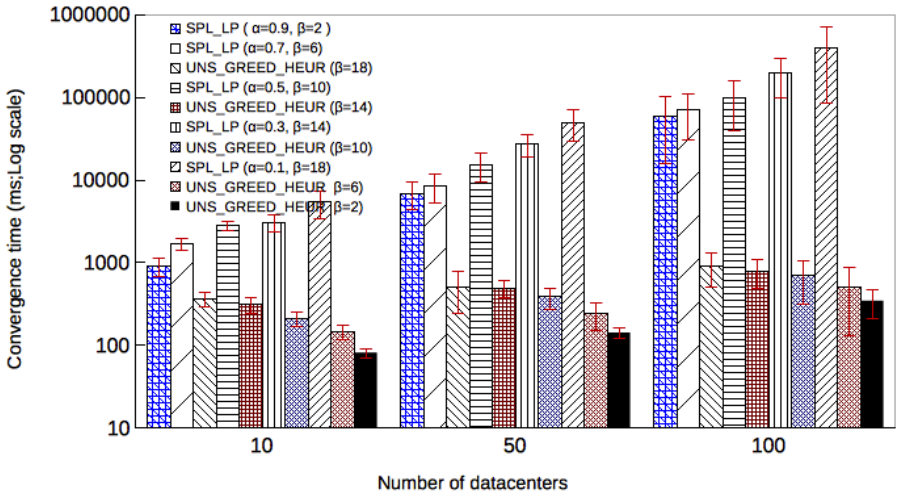

Figure 5.15 – Time execution comparison between UNS_GREED_HEUR and SPL_LP algorithms for different datacenter sizes when the amount of hosted intermediate data are 1000GB.

500GB and 1000GB for the simulated scenarios from 50 to 100 datacenters, while the time running the UNS_GREED_HEUR remains very fast to find solutions with a convergence time improvement ratio ranging in $[10^1,10^3]$ as compared to the SPL_LP algorithm. Although dependency parameter values increase the number of routing β from 2 to 18 commodities, the heuristic algorithm scales better already for these large instances and is more robust in ensuing scenarios and simulations. By contrast, the SPL_LP algorithm performance reacts poorly to dependency parameter variations. 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. 5.14.

The convergence time increases with the number of datacenters as well as slightly less in the scale amount of intermediate data dependencies (see Fig. 5.15) since the data bandwidth capacity is limited to a maximum of 10 GB for data transfer, but, with more transfer links from the scale number of datacenters. The gap between the algorithms is in range $[10^2,10^4]$ with an improvement factor in favor of the UNS_GREED_HEUR algorithm. As expected for the SPL_LP algorithm that was buildt 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 solutions faster from than the NP-hard problem, for an intermediate data dependency placement that goes beyond 500GB for 100 datacenters, the convergence time remains widely slow at about 7 minutes.

In conclusion, the running time of the proposed algorithms depends mostly on the cloud infrastructure topology, and slightly less on the scale of the amount of intermediate data dependencies for the SPL_LP algorithm. Besides, the change in dependency parameter values influence largely the SPL_LP algorithm performance and much less the UNS_GREED_HEUR algorithm. This validate our motivation for the use of a heuristic approach to find solutions faster even if there are bound to be approximated as reported in Table 5.1.

5.4.2.4 Federation and heuristic algorithms comparison

We emphasized in the previous chapter the lack of the scalability of the ExactFed_BDWP algorithm in relation to the growth of the amount of file dependencies, especially in the number of datacenters in the federation. The low scaling property above a given size (beyond 18 as demonstrated simulation results in previous chapter), as well as the growth of the matrix-based inter-file dependencies from the federation model, motivates for more scalable algorithms. Even though SPL_LP and UNS_GREED_HEUR algorithms solve the problem differently through the dependency requirements (intra-job and inter-job for SPL_LP and UNS_GREED_HEUR respectively, regarding to inter-file for ExactFed_BDWP), we extend the analysis for the ExactFed_BDWP algorithm by comparing its performance with the two proposed algorithms. This evaluation aims at finding the conditions that are favorable for the execution of ExactFed_BDWP algorithm. To this end, we set the necessary simulation in terms of input parameters: number of datacenters and the amount of intermediate data. Then, we conduct the extended simulation for β value (10, 14 and 18) for both algorithms while keeping the value of α to 0.1 for the SPL_LP algorithm. The amount of dependencies is ranging in [100GB,1000GB] for each value. We adopt the similar simulation conditions as previous chapter regarding capacity and price (no I/O request and movement costs). All figures below depict the results of 15 averaged runs.

a) Convergence time of ExactFed_BDWP

The first targeted simulations consist in comparing the ExactFed_BDWP and the proposed algorithms for an amount of intermediate data dependencies varying for 100 GB, 500 GB and 1000 GB while ,the number of datacenters are ranging in [20, 80]. Figures 5.16, 5.17 and 5.18 present the collected required time to find a solution for the intermediate data dependency placement for ExactFed_BDWP, SPL_LP and UNS_GREED_HEUR algorithms. These three figures highlight the limits of the ExactFed_BDWP algorithm when increasing the number of datacenters. It depicts better performance in execution time for the

UNS_GREED_HEUR algorithm which is designed to reduce convergence times and scale for very large input parameters. The UNS_GREED_HEUR algorithm finds optimal solutions in less than 1 second for all simulations which is 2 to 6 times faster than the ExactFed_BDWP algorithm with an increasing number of intermediate data size and evolving cloud datacenters. However, the SPL_LP algorithm can be achieved in milliseconds to seconds time scales (Fig. 5.16 and 5.17), and in a few minutes time scales as shown in Fig. 5.17 .

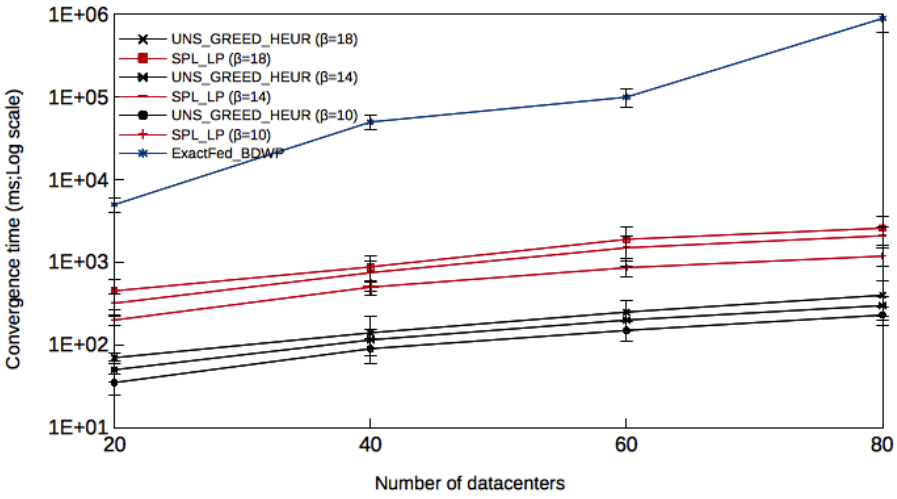

Figure 5.16 – Time execution comparison of ExactFed_BDWP algorithm with UNS_GREED_HEUR and SPL_LP solutions for various number of datacenter when the amount of hosted intermediate data is 100 GB.

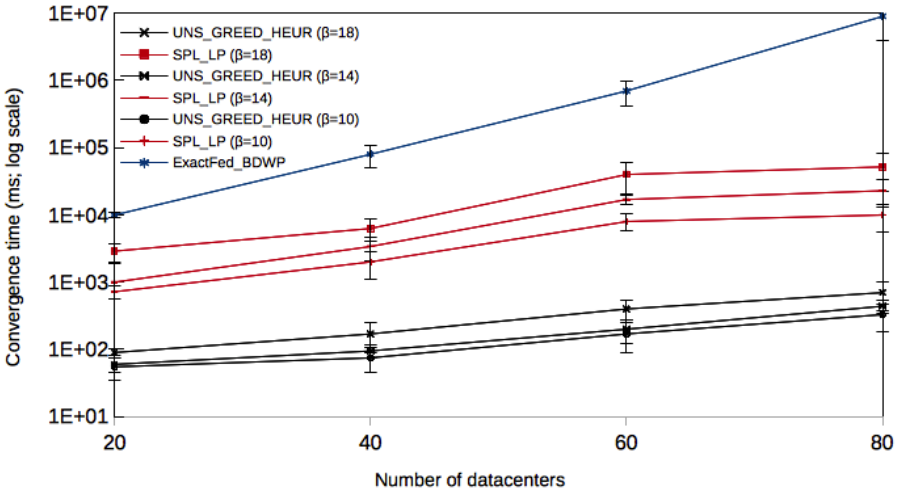

Figure 5.17 – Time execution comparison of ExactFed_BDWP algorithm with UNS_GREED_HEUR and SPL_LP solutions for various number of datacenter when the amount of hosted intermediate data is 500 GB.

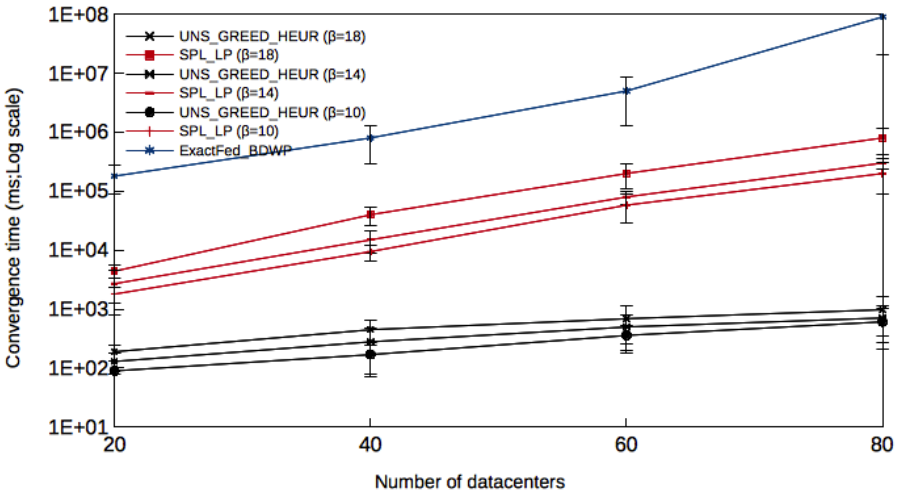

Figure 5.18 – Time execution comparison of ExactFed_BDWP algorithm with both UNS_GREED_HEUR and SPL_LP solutions for various number of datacenter when the amount of hosted intermediate data is 1000 GB.

As suspected for the ExactFed_BDWP algorithm, that relies on the branch-and-bound method, the execution time remained the least on scale regarding to the amount of dependencies, especially for large number of datacenters, i.e beyond 40. As the problem is NP-Hard, the convergence time of the ExactFed_BDWP algorithm grows exponentially when increasing the number of datacenters and dependency size by comparing each file pair involving inter-file dependencies which greatly increases the resolution space (more than one hour in the extreme case). Indeed, the ExactFed_BDWP algorithm is an offline solution that suffers from high-time complexity and is computationally prohibitive, against the SPL_LP approach that delivers semi-online solutions with real variables in each time slot (about 13 minutes in the extreme case) and deals with a large set of clusters in routing, thus greatly reducing the research space. Thus we confirm the scalability is mostly governed by the size of the federation and the amount of dependencies in the routing.

b) Impact of the amount of routed intermediate data dependencies on the performance of ExactFed_BDWP and UNS_GREED_HEUR algorithms

To pursue the performance evaluation of ExactFed_BDWP we only compare to heuristic solutions. Indeed, we estimate the total storage cost in routing the amount of dependencies by varying their size ranging in [100 GB, 1000 GB], while the number of datacenters is set to 20. We also highlight the results obtained for dependency value β = (10, 14 and 18) of the UNS_GREED_HEUR algorithm. The obtained results are not surprising and they reflect the expected results for the ExactFed_BDWP algorithm performance. Fig. 5.19 indicates that

ExactFed_BDWP slightly exceeds the solutions found using the heuristic for total storage costs in 190, 180, 155 respectively for each dependency values (10, 14 and 18) and $115 for the exact algorithm. In this result, the cost ratio does not exceed 1.65 when intermediate data size reached 1000 GB.

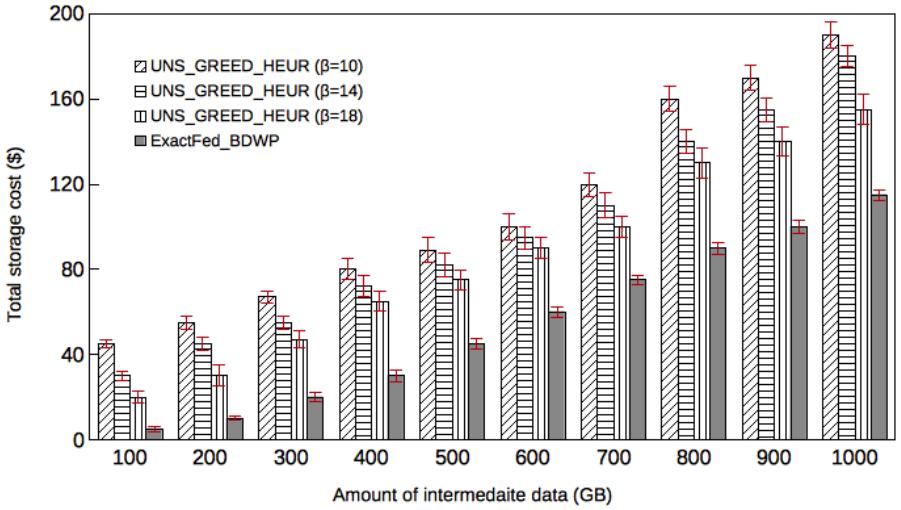

Figure 5.19 – The total storage cost of the Algorithms ExactFed_BDWP and the UNS_GREED_HEUR heuristic by varying the number of datacenter when the amount of hosted intermediate data are set to 1000 GB.

The combined simulation results on a single graph allows to verify the fairness aspect of the ExactFed_BDWP algorithm. In fact, since storage and bandwidth resources are shared in the federation, the cloud storage providers hosting datacenters commit to cooper the insourcing and outsourcing prices. The results of the ExactFed_BDWP algorithm have to be balanced either in terms of number of datacenters (as reported and observed on previous chapter) as well as in the offered price to respond to the increasing amount of big data workflow placement. This situation is widely observed in the Fig. 5.19 that illustrates that each point for the ExactFed_BDWP algorithm is linear for the total storage cost with negligible margin between each (the three curves). However, we see a somewhat larger margin (about ten dollars) between UNS_GREED_HEUR algorithm curves that depend not only on the size of the dependencies, but also on the capacity and fixed prices for each datacenter.

5.5 Conclusion

This chapter presented and evaluated an exact (SPL_LP) and a greedy heuristic (UNS_ GREED_ HEUR) algorithm for intermediate data placement while saving the total storage

cost for task workflow processing systems across distributed datacenters from cloud storage provider. The presented solutions solve the case of intra-and inter-job dependencies including fractional and atomic demands respectively. The SPL_LP algorithm based on LP model introduces new locality constraints on the optimal placement of intermediate data dependencies so 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 datacenter storage capability as well as a movement optimization. Despite our simple and clean formulation, the large number of datacenters and the variation of intermediate data dependency parameters as well as their localization and constraints makes it only tractable for small or medium instances, this is even greater for an unsplittable demand as the ILP model that is still NP-hard. In order to ensure the placement of inter-job dependency-based intermediate data, we developed a novel greedy optimization framework based on a heuristic algorithm which solves the problem in fast time making an assumption of an optimal fractional solution. Extensive experiments by simulations show that the greedy heuristic algorithm performs closer to the exact formulation solutions (SPL_LP and ExactFed_BDWP), and boots higher performance as compare to other heuristic scenarios acting as benchmark. We evaluated also the convergence time of the proposed algorithms that it improves by several orders of magnitude for the greedy heuristic algorithm as compared to the exact algorithm and handles the cloud infrastructure of hundreds of datacenters in practical convergence times.

Chapter 6

General Conclusion and Future Works

Sommaire

6.1 Conclusion & Discussion

The present work discloses a great niche issue in intermediate data storage management and cloud models, i.e. the cloud data-workflow management. In fact, cloud infrastructure or provider is the default solution today for managing and deploying scientific and complex data applications. More specifically, this thesis deals with the problem of placing intermediate data as a first-class citizen resulting from big data workflow-based applications, run by considering its multiple facets and levels to provide not only a specific solution, but also a generic and complete approach. To cope with the problem of intractability, we proceed to its resolution in different stepping approaches and three data workflow placement strategies were proposed while making use of different theoretical approaches. The overall goal of these approaches is to design and develop models and algorithms for intermediate data placement while considering different dimensions of the problem. These key dimensions are application needs and requirements, the adaptability of the solution, the type of cloud infrastructure model and the cost optimization. Prior to, a primary approach was considered in this thesis to take a renewed look at the problem of managing intermediate data in a more focused way. This study discussed salient features of the intermediate data access behavior, and introduced and led to a placement problem in the broadest sense for big-data workflow in cloud environment. Our manuscript is fivefold including contributions. Hereby, we summarize them.

Chapter 1 introduced big data workflow applications in the cloud, which is the background of this research. Chapter 1 also described the research problems and objectives of our work, the key issues to be addressed and the primary structure of the manuscript.

Chapter 2 provided a detailed overview of the data-workflow management approaches in cloud datacenters. Our survey are classified into related topics which present the work objectives that depend mainly on the thesis positioning in relation to existing research. We mainly focused on a taxonomy of the different approaches of the relevant related work and compare them based on the criteria that characterizes our orientation meeting: i) operational storage cost, ii) dynamic or fixed pricing model, iii) data dependency constraint, iv) data placement strategy, v) federation/multi cloud-based or single-cloud provider scenario.

Chapter 3 addressed the study and understanding of intermediate data access behavior of MapReduce tasks processing in Hadoop cluster. This latter has served as a mean of experimentation which is a reference system for processing big data. Three benchmarks based on MapReduce application logs are run on this platform to collect disk I/O traces. A statistical Markov model-based sequence learning is proposed to represents the behavior of disk I/O accesses (collected traces) of MapReduce applications by modeling a single I/O write request of spill phase during the processing of map operations. Chapter 3 also proposed a prediction algorithm that use Markov transitions to detect interfered I/O in the spill file from a map operation among successive I/O spill from concurrent map operations at block level of shared disk space. To assess the prediction model quality and the efficiency of the algorithm, a trace-driven simulation is carried out. The assessment concerns the prediction accuracy of I/O spill size with respect to the model size of a single step, and the number of sequential/interfered I/O spill predicted by the proposed algorithm. The approach provides the best decision for the I/O optimization in MapReduce processing based on the prediction of intermediate data interference. Moreover, in relation to our other contributions, the harmonization of the operations of the MapReduce tasks (co-scheduling tasks) on disks, reflects a particular case of a data workflow-based model.

Chapter 4 presented an exact federation big data workflow placement algorithm (Exact_Fed _BDWP) in distributed cloud datacenters. The Exact_Fed_BDWP algorithm was formulated and solved using both ILP and branch and bound methods. The proposed algorithm takes into account dependency requirements (valuable and unnecessary correlation) of intermediate data file pairs that are shared and reused in a federation environment which avoids using the pay-as-you-go model. In fact, a federated datacenter that participates and cooperates for optimal intermediate data placement across multiple cloud storage providers identifies favorable conditions for joining a federation and helps both scientific users and service providers to bring the cost down drastically when deploying, and the big-data workflow storage. The Exact_Fed_BDWP algorithm receives input matrix parameters including dependency values between file pairs and their respective localization (home datacenters) to achieve optimal insourcing and outsourcing decisions as output parameters. The latter

concerned the minimization cost that includes transferring and storing a data workflow in federated datacenters, and reallocating intermediate data requests with dependencies in a single datacenter to reduce the total storage cost. The Exact_Fed_BDWP algorithm showed to be efficient in minimizing the storage cost and exhibited an acceptable execution time for practical storage federations and typical file dependency sizes by providing a better balance among datacenters.

Chapter 5 described a new generic approach that was intended to overcome the absence of a dependency type and large instance problem with Exact_Fed_BDWP approach. A linear programming model based exact algorithm and a greedy heuristic algorithm (SPL_LP and UNS_GREED_HEUR) have been developed to address the splittable and unsplittable variant of intermediate data dependencies placement problem as minimum cost multiple-sources multicommodity flow problem respectively. The proposed algorithms combine multiple criteria such as data bandwidth and datacenter storage capability as well as a movement optimization in routing and placing intermediate data in the cases of intra-and inter job dependency. The SPL_LP algorithm achieved the best cost and slightly less on scale of the amount of intermediate data sizes regarding to varying dependency parameters. This motivated the use of the UNS_GREED_HEUR algorithm to find faster solutions even if they are bound to be approximate. The UNS_GREED_HEUR algorithm operates on cloud datacenter infrastructure topologies as a greedy framework optimization graph to reduce convergence times by several orders of magnitude as compared to both SPL_LP and Exact_Fed_BDWP algorithms.

6.2 Future Research & Orientation

During our work, we faced different complex problems related to the data workflow management problem in cloud environments. We solved some of them and included others in our future work. The potential future directions of this research include:

— The Exact_Fed_BDWP algorithm could be extended to improve convergence times. In order to ensure this, we plan to reduce the size of the matrices containing dependency matrix and home datacenters. For large instances, binary matrix become very dense and a reduction method is necessary to compute the objective function very quickly as this confines even more the convex hull problem.

— The Exact_Fed_BDWP algorithm can also be extended to enhance storage and network resource utilization by considering a novel pricing strategy. This also allows to integrate the profit for cloud storage providers that is excluded in our approach. The

storage pricing strategy must capture the time processing constraints. It must include the constraints and requirements of workflow applications execution in a shared and collaborative environment in terms of reservation plan or subscription. For example, resource providers must therefore reduce the rejection of reservations and not impose restrictions such as minimal notice periods, which reduce the effectiveness of reservations as a mean of allocating the desired resources at a desired time.

— We have validated the proposed heuristic which is an approximate solution of the SPL_LP algorithm through practical simulations. However, the UNS_GREED_HEUR algorithm can also be extended in order to have a better theoretical approximation with a guaranteed approximation ratio based on the proposed optimization framework. In fact, it is very difficult to approximate our unsplittable problem, which means that it must be built upstream.

— Our proposed approaches are offline (ExactFed_BDWP and UNS_GREED_ HEUR) and semi-online (SPL_LP). Among the means of making them online is to add predictors for the ported algorithms. The correlation of the set of intermediate data can be clustered to provide a convenient way to generate correlated multivariante random variable distributions for each type of dependency and to present a solution for the difficulties of transformation of the density estimation of the intermediate data dependencies as input parameters.

— The intermediate data dependency placement problem could be extended to handle collaborative tasks, and inject synchronization time constraints between the processes performed by these tasks. Adding the time constraint makes the execution of the problem in dynamic environment.

— There are workflow implementations of the MapReduce programming model (e.g., Cloudgene, Zookeper, Oozie and Cascading), and a natural extension and improvement of the Markov model is to add these different implementations to have a generic prediction model to the problem optimization of data and task workflow scheduling and placement.

— Another natural extension of the work is to efficiently and reliably handle intermediate data, including failure types inducted in execution error of tasks and their cost. Therefore, it aims at finding a trade-off between availability and storage cost of the intermediate data replica. For these purposes, we plan to improve intermediate data

placement cost by applying data replication and multi datacenter distance cost as well as considering the scenario of task failure in the cost model to support substitution of collaborative tasks since we have introduced the processing of the non-valuable dependencies induced precisely by the errors of execution of the tasks in the Exact_Fed_BDWP approach.

Annexe A

Résume en Français

A.1 Introduction

À ce jour, l'univers numérique est confronté à la suite de l'explosion des données. Ce volume massif de données est capturé par des organisations, tel que l'augmentation des médias sociaux, l'Internet des objets (IOT) et les multimédia, à une base régulièrement croissante dans le monde. Cette quantité de données est disponible à partir de plus de 1 trillion de pages Web actuellement accessibles sur le Web. Comme l'indique *International Data Corporation*(acronyme de IDC) [VON+15], la quantité de toutes les données numériques générées, créées et consommées en une seule année passera d'environ 3 000 EB en 2012 à 40 000 EB en 2020. Actuellement, environ 90% des données numériques disponibles ont été créées au cours des deux dernières années [Gob13]. Ainsi, acquérir, stocker, guider et traiter de manière exponentielle ces énormes quantités de données numériques récemment créées, constitue un défi complexe que n'on nomme souvent l'essence du Big Data. En effet, le Big Data décrit la croissance continue des données hétérogènes, structurées ou non structurées, qui sont générées et collectées à partir de toutes sortes de sources de données (citées auparavant). La gestion du Big Data avec les formats de données diversifiés est une base principale pour la concurrence dans les business et la gestion en soi. Par conséquent, le Big Data pose un défi aux organisations industrielles ainsi qu'aux chercheurs scientifiques qui leur présentent une gamme complexe de problèmes d'utilisation, de stockage et d'analyse. S'attaquer au besoin du Big Data exige fortement des changements fondamentaux dans l'architecture des systèmes de gestion de données. Parmi eux, figurent les systèmes de traitement de *workflow* hautement distribués qui sont au cœur de la gestion du volume massif et complexe du Big Data. Les données de ces systèmes peuvent être des entrées pour des applications données ou des résultats intermédiaires qui doivent être stockés et gérés efficacement. Certaines applications de ce type incluent des techniques de traitement de données scientifiques à haute performance, des applications de traitement de données scientifiques intensives et le streaming en temps réel. Ces applications sont soumises à une série de phases de calcul. Les *frameworks* de *workflow* intègrent et coordonnent plusieurs *jobs* qui peuvent contenir plusieurs tâches collaboratives [HPL13, KJH+14, CBHTE10]. Certaines de ces tâches sont exécutées séquentiellement, mais

d'autres peuvent être exécutées en parallèle sur une plateforme distribuée. Par exemple, une organisations scientifique telle que le Telescience Project Research [LDU+] exécute des tâches scientifiques parallèles dans un *pool* de ressources partagées et hétérogènes. Chaque tâche génère non seulement des données sur les microscopes et l'image biomédicale, mais a également besoin des résultats intermédiaires de ses tâches collaboratives sur l'analyse d'image biomédicale pour des études de corrélation. Une autre organisation scientifique, qui est le *Climate Corporation Research* basée sur le système de tâches de type *workflow*. Ce projet a adopté des capteurs de composants situés sur plusieurs emplacements afin de capturer et de générer une quantité massive de données, y compris des champs agronomiques, environnementaux et météorologiques à haute résolution. Une grande quantité de données sont générées quotidiennement à partir de ces systèmes de *workflow* de traitement des données qui sont extrêmement importants avec une grande diversité de types, mais il devient difficile de les traiter et de les stocker efficacement. De même, d'autres applications traitent des données massives sous forme d'un *workflow* de plusieurs tâches en utilisant le paradigme de calcul MapReduce. Ce dernier est adopté et intégré par des entreprises valorisantes dans le monde comme Google, Facebook, Amazon et LinkedIn. Un tel écosystème d'applications nécessite une composition flexible des tâches d'un *workflow* prenant en charge différentes phases de traitement.

Entre-temps, l'émergence du *Cloud Computing* offre une nouvelle connaissance clé pour les entreprises de sous-traitance (externalisation) d'infrastructures informatiques (IT) qui peuvent être requises et retournées à la demande avec des modèles de tarifications flexibles [HSS+10]. Le Cloud fournit principalement des services de stockage et de traitement de données, optimisés pour une haute disponibilité et une durabilité. Ainsi, en adoptant les modèles de stockage et de traitement dans un Cloud à travers les centres de données distribuées, le déplacement des tâches collaboratives d'un *workflow* vers le Cloud peut directement effectuer des opérations de stockage et de traitement de données à grande échelle et complexes, au détriment d'une performance. Malgré la transition rapide vers l'utilisation des services dans le Cloud, certains challenges critiques sont soulevés et restent maintenus. Un problème difficile, tant pour les entreprises que pour les chercheurs scientifiques, est de savoir comment traiter, stocker et gérer cette masse de données générées par de telles applications (*Big Data Workflow*) de manière rentable et efficace pour obtenir le niveau de performance souhaité. D'autre part, certaines fonctionnalités importantes du *Big Data Workflow* telles que le partage de données ou la réutilisation des résultats intermédiaires et la réplication géographique, sont des principales options, bien que beaucoup d'autres ne soient pas supportés [Tud14] : le transfert géographiquement distribués, l'optimisation des coûts, la gestion des tendances de données générées, la qualité de service différenciée, les compromis personnalisables entre coût et performance. Tout cela nous intrigue dans le sens où les applications du *Big Data Workflow* sont souvent coûteuses (en temps et en argent) ou difficiles à structurer en rai-

son des difficultés et des inefficacités dans la gestion des données dans un environnement tel que le Cloud. Compte tenu de cela, la fourniture de services diversifiés et efficaces pour la gestion des données dans le Cloud sont des jalons clés pour la performance d'exécution de ces applications. Par conséquent, cette thèse se concentre sur le problème de la gestion du Big Data dans le Cloud pour les applications de *workflow*, tout en assurant un stockage et un traitement rentables de leur données générées et distribuées. Cependant, l'adoption de grandes fonctionnalités du *Big Data Workflow* dans un Cloud distribué est un défi de taille. Pour cela, nous proposons de nouvelles approches simples et efficaces pour la gestion des données de ces applications, tout en considérant leurs besoins et exigences fonctionnant dans une infrastructure de Cloud dont les centres de données géographiquement distribuées. Plus précisément, cette thèse traite le problème qui consiste à router et placer les données et les résultats intermédiaires résultantes des applications basées sur des traitements de type *workflow*. En considérant leurs caractéristiques, cela permet de fournir non seulement une solution spécifique, mais aussi une approche générique et complète pour la gestion des données de ces applications qui devraient bénéficier de toute l'attention des communautés scientifique et industriel.

A.2 Contributions

Dans ce contexte, la première contribution de cette thèse aborde le problème de performance de co-ordonnancement et d'accès des entrées/sorties (E/S) des données intermédiaires des applications de type Hadoop-MapReduce. Les performances de leurs systèmes d'E/S a motivé de nombreuses optimisations, notamment au niveau de la gestion et du placement de ces données. Afin de mener à bien ces optimisations, il est nécessaire de comprendre, de modéliser et de prédire les accès des E/S de ces applications. Ainsi, l'approche proposée côtoie les données intermédiaires résultantes du traitement de trois applications déployées dans un *cluster* Hadoop. L'approche proposée implique d'abord un modèle de prédiction à base de Markov. Ce modèle caractérise le comportement des E/S liées aux données de la phase *spill* à partir des tâches *map* concurrentes. Puis, un algorithme de prédiction est proposé qui exploite ce modèle de Markov. L'algorithme proposé permet de distinguer les différents accès interférés et prédire ainsi les futures opérations d'E/S aux disques. De ce fait, en considérant une méthodologie bien définie nous dérivons les caractéristiques qui peuvent être extraites à partir de l'exécution de ces application (*micro-benchmarks*), et qui capture ainsi le comportement des E/S des fichiers de données intermédiaires. Ensuite, la caractérisation de ces accès est injectée dans le modèle de Markov. L'étude du comportement des accès des E/S des fichiers *spill*, ou tout simplement le placement des données intermédiaires sur le disque permet de songer à une meilleure gestion de ces données là concernant leur taille, leurs sources et la na-

ture des applications qui les génèrent. De plus, cela a permis de positionner et d'orienter notre réflexion vers des stratégies de placement des données intermédiaires pour des applications utilisant le modèle MapReduce extensible dont les *workflows* génériques déployés dans des centres de données de Cloud. La deuxième partie des contributions concerne une proposition d'une approche rentable pour le placement des données d'un *workflow* dans un environnement coopératif tel que le Cloud fédéré. Le but de l'approche est de faire fonctionner un algorithme basé sur un modèle analytique et exacte ILP (acronyme de *Integer Linear Programming*) pour le placement de données intermédiaires de dépendances dans de multiple centres de données de taille moyenne provenant de différents fournisseurs de stockage dans le Cloud. L'algorithme répond à la fois aux caractéristiques et contraintes du modèle de fédération de stockage dans le Cloud et aux exigences du placement des paires de fichiers (dépendances symétriques) de données résultantes du traitement d'un *workflow* tout en minimisant le coût de stockage induit pendant l'acheminement et le placement de ces données. La dernière partie des contributions présente une nouvelle approche afin d'aborder le problème de placement des données intermédiaires d'un *workflow* dans un sens plus large. En effet, l'approche proposée met en jeux deux algorithmes. Ces algorithmes traitent deux autres types de dépendances, inter-et intra-*job* (dépendances asymétriques), pour le placement des données intermédiaires générées par l'exécution d'un ensemble de *jobs* de *workflow*. D'où l'implication de deux variantes : un algorithme exacte relaxé et une heuristique satisfaisant les exigences de ces deux types de dépendances tout en minimisant le coût de stockage total des données intermédiaires routées et stockées dans le Cloud. L'algorithme exacte basé sur le modèle LP (acronyme de *Linear Programming*) aborde le placement des données pour le cas des dépendances intra-*jobs*. Celles-ci, sont dites fractionnelles (*splittable*) car les caractéristiques et la particularité de ce type permettent de router et de placer les données et résultats intermédiaires séparément dans un environnement géographiquement distribué. Par contre, l'heuristique qui est basée sur une approche de type *greedy* traite le type de dépendance inter-*job*, dont les données intermédiaires sont atomiques (*unsplittable*) durant leur routage et leur placement dans le Cloud. Principalement, leur objectif est d'économiser les coûts, y compris les efforts de transfert, de stockage et de déplacement ou de migration de ces données de dépendance en fonction de leurs besoins et exigences. En considérant de grands ensembles de données de corrélation (*cluster*) comme paramètres d'entrée et des instances très larges conduisant à une complexité algorithmique inférieure pour l'heuristique par rapport aux deux autres algorithmes exactes proposés.

A.3 Organisation & structuration

Le présent document est organisé comme suit. Le chapitre 2 résume les principales stratégies de gestion du Big Data les plus pertinentes trouvées dans la littérature en relation avec les objectifs et les focalisations de cette thèse. Le chapitre 3 décrit la conception et l'étude du comportement de placement et de co-ordonnancement des E/S des données intermédiaires à travers un modèle et un algorithme de prédiction. Le chapitre 4 présente notre premier algorithme (Exact_Fed_BDWP) de placement des données intermédiaires de dépendance de type inter-fichier d'un *workflow* dans une environnement multi-Cloud et fédérés. Le chapitre 5 propose une nouvelle approche et un *framework* (SPL_LP et UNS_GREED_HEUR) d'optimisation et de gestion des données et des résultats intermédiaires d'un *workflow* pour gérer l'économie des coûts de stockage agrégés (ensemble des coûts liés au stockage) du placement de ces données de dépendance de type inter-et intra-*job*. Enfin, le chapitre 6 conclu nos contributions et expose nos travaux futurs dans ce domaine de recherche justifié.

A.4 Travaux antérieurs

Cette partie fournit un aperçu sur la caractérisation des E/S des données des applications de type MapReduce, ainsi que les modèles de prédiction qui étudient le comportement des E/S et qui sont basés sur l'apprentissage séquentiel et les chaines de Markov. Cette étude est le préambule au contexte de recherches antérieures sur l'ordonnancement de données intermédiaires dans les applications basées sur un traitement de type *workflow* (MapReduce est considéré comme un modèle initial de type *workflow*). Ensuite, à travers un tableau récapitulatif, nous fournissons une étude résumée concernant les approches de gestion des données ainsi que leur different objectifs d'optimisation dans un environnement tel que le Cloud.

A.4.1 De la gestion à la prédiction des accès des données de MapReduce

Depuis que MapReduce est devenu un framework de programmation parallèle efficace et extensible pour les modèles de traitement des données des applications *workflow*, la précision et la compréhension du stockage de base pour ces données deviennent une exigence fondamentale pour la bonne execution de ces applications. Par ailleurs, ces applications produisent énormément de données intermédiaires transférées entre chaque phase de traitement. Les données intermédiaires possèdent des caractéristiques différentes de celles de données significatives (l'entrée et la sortie d'une application). Alors que les données d'entrée et de sortie devraient être persistantes et susceptibles d'être lues à plusieurs reprises (pendant et après

l'exécution d'une application), les données intermédiaires sont des données transitoires qui sont habituellement écrites une fois par une seule phase de traitement, et sont lues une fois par la phase suivante. L'exécution parallèle de ces applications ainsi que le partage de ressources des E/S rend encore plus difficile la gestion de données intermédiaires, principalement lors des interferences qui surgissent entre ces applications pendant leur exécution. Un effort important a été fourni ces dernières années pour mettre au point des modèles robustes et des approches de prédiction pour la représentation du comportement des accès aux disques des applications des données complexes. Dans [KKC15, MTK+15a, YWWL13, Gro12, YLLQ12], les auteurs se concentrent sur la coordination des applications MapReduce pour atténuer les interferences des E/S, et caractérisent ainsi le comportement des E/S de HDFS (acronyme de Hadoop Distributed File System) et les requêtes émises par ce dernier. Cependant, les auteurs n'ont pas abordé le traitement et l'analyse du comportement des accès des données et résultats intermédiaires sous les interferences des E/S des applications concurrentes, qui devient l'un des principaux goulets d'étranglement pour les performances des applications de type *workflow*.

A.4.1.1 Prédiction des E/S des applications de données intensives

La prédiction des modèles d'accès des E/S des applications de données intensives ont été longtemps un objectif important dans un environnement de stockage parallèle et distribué. Les chercheurs ont étudié des méthodes statistiques (e.g., les prédicteurs Markoviens, les modèles de Markov cachés [Ree04, HBT+13] et la progression linéaire [Noo]) ou des méthodes non statistiques comme la détection des patterns fréquents [LCLZ14], le profilage des E/S [MTK+15b] et la simulation online [WKKB13]. Ces approches sont principalement basées sur la caractérisation du comportement spatiaux ou temporels des E/S nécessitant un grand nombre d'observations pour réaliser une prédiction précise et de qualité. Le modèle de prédiction de Markov trouve diverses applications en informatique, en particulier dans la modélisation des E/S des données et la prédiction du comportement de leur ordonnancement. Les modèles de Markov peuvent être utilisés pour établir la précision et le comportement des accès des E/S concernant les données à gérer. Ils établissent un équilibre efficace entre la puissance prédictive et la complexité de la mise en œuvre qui nécessite un long temps d'exécution. Ces modèles aussi requièrent un apprentissage *offline* à base de trace afin de converger. Certains travaux de recherches ([PSS10, OR02, MR97, DSVK11, Kho]) ont proposé des modèles de prédiction basés sur les chaines de Markov qui ont utilisé ces aspects d'apprentissage pour prédire les accès aux mémoires de stockage des applications, à savoir, les données de simulation et de calcul ou les données personnelles. Cependant, on peut observer que les modèles de Markov ne sont pas exploités dans la modélisation et la caractérisation des E/S générées par

les application de type MapReduce. Les profils d'accès aux données intermédiaires ainsi que la caractérisation de leur comportement sont en grande partie inexplorés dans les modèles statistiques prédictives.

A.4.2 Gestion des données intensives dans un environnement de Cloud

Une série d'études récentes ont été menées au sujet du problème de placement de données intensives dans un environnement de Cloud qui motivent ainsi les objectifs de cette thèse. Ces travaux adressent le problème de acheminement des données des applications *workflows* en respectant leurs exigences et leurs caractéristiques et/ou en minimisant le coût opérationnel induisant pendant le stockage de ces données dans une infrastructure de Cloud. Par ailleurs, le calcul du modèle de coût pour l'acheminement et le placement des données dans un Cloud est un problème NP-difficile [CPL16, ZXZ+15, ZCLS16, RLZW16, GHKS13]. Cependant, il existe des travaux qui prennent en compte les caractéristiques des applications *workflows*. Ces caractéristiques montrent principalement des dépendances complexes entre les données intermédiaire générées et distribuées lors de l'exécution de ces applications [EMKL15, ZXZ+15, YYL+12], d'où la valeur dynamique du coût induit pendant le placement de ces données dans un environnement tel que le Cloud. L'économie du coût optimal pour la gestion des données d'un *workflow* dans un environnement de Cloud est déterminé en minimisant différents paramètres de stockage [DSL+08, VSPD+13, XLX17, TTC15]. En effet, il existe différents facteurs de coût qui influencent sur le déplacement et la migration des données entre les centres de données répartis dans le Cloud. La plupart des travaux de recherche [DSL+08, VSPD+13, BKT13, Rei11, LSWL16, STT09] n'ont pas jugé dans la construction de leur modèle de coûts, la contrainte de dependance des énormes ensembles de données générés, en plus du type de corrélation des données intermédiaires, aucune n'a vraiment soulevé les types de dépendances qui seront traitées dans cette thèse tout au long de la décision de leur acheminement et de leur placement dans le Cloud.

A.5 Contribution 1 : Prédiction des interférences des E/S des données intermédiaires à partir des accès concurrents des taches MapReduce-Hadoop

A.5.1 Présentation

L'objectif principal de cette contribution est de comprendre le comportement des accès et de l'ordonnancement des E/S aux disques des données intermédiaires dans un environnement

d'execution concurrent. Ainsi, nous proposons de prédire le comportement d'accès des fichiers de données intermédiaires dans les traitements concurrents des applications de MapReduce en utilisant un modèle statistique de Markov. Ce modèle est basé sur la localité spatiale des blocs de fichiers de données intermédiaires et il analyse la séquentialité des fichiers générés par la phase *spill* durant l'exécution de ces applications (fichiers générés lors des écritures ou des appels flush () sur un disque). Aussi, nous proposons un algorithme de prédiction basé sur le modèle de Markov pour choisir les séquences de probabilités de transitions des E/S et prévoir les futures requêtes d'accès des données intermédiaires. Afin de valider le modèle de prédiction, un grand nombre d'observations provenant des serveurs de stockage d'Hadoop ont été effectués pour extraire des traces des E/S sur les fichiers *spill*. Par ailleurs, le modèle de Markov caractérise les accès des E/S à un niveau bas (niveau disque et ordonnanceur) sans nécessiter la sémantique de l'information disponible à un niveau supérieur (système d'exploitation). Ainsi, l'évaluation proposée utilise des expérimentations axées sur des traces qui génèrent les E/S des données intermédiaires qui construisent et valident le modèle de Markov. La solution pourra, entre autres, aider à améliorer les méthodes d'optimisation des E/S, et de fournir les bonnes décisions pour le placement des données intermédiaires (dans les traitement MapReduce ou de type *workflow*). Cependant, il est important de souligner que le modèle de prédiction proposé est basé sur des traces collectés sur des serveurs d'une plateforme Hadoop, mais il peut être facilement adapté aux traitements distribués MapReduce sur environnement basés sur les accès aux disques.

A.5.2 Méthodologie

Afin d'élaborer le modèle de Markov, nous identifions une méthodologie qui déploie l'expérimentation de la plateforme Hadoop, un système de référence pour le traitement de données intensives. À partir d'un scénario réel, il est très compliqué de tracer des fichiers temporaires (*logs*) existants et d'y accéder, tels que dans les clusters de Yahoo, de Facebook ou de Google. D'où la nécessité d'exécuter et de déployer un ensemble de *micro-benchmarks* afin de récupérer des *logs*. Par conséquent, sur une plateforme Hadoop, un nombre d'exécution de tâches *map* et *reduce* sont effectuées pour enfoncer la qualité de la prédiction et de l'apprentissage à travers l'exécution de trois applications de type MapReduce : Wordcount, Kmeans et Terasort. À l'aide de ces applications, des statistiques nécessaires sont collectées sur les opérations des E/S aux disques concernant les accès des données intermédiaires. Les E/S des fichiers *spill* sont générées plus tard et utilisées pour la caractérisation de ces accès. Ainsi, un apprentissage *offline* basé sur des traces ainsi que des traitements spécifiques sur celles-ci sont utilisés pour converger vers un modèle de Markov. Selon ce modèle, les états correspondent aux blocs logiques des E/S des fichiers *spill*, et les transitions représentent les probabilités de

commutation entre chaque E/S des fichiers *spill* consécutifs. Ainsi, un modèle statistique de Markov est construit et combiné avec un algorithme pour prédire les futures E/S interférées. Enfin, une série d'expériences est conçue pour valider la qualité et la précision de l'algorithme et du modèle de Markov.

A.6 Contribution 2 : Algorithme exacte pour le placement des données intermédiaires de type inter-fichier dans le Cloud fédéré

Dans cette partie, nous abordons la deuxième contribution qui est un algorithme d'optimisation appelé "*Exact Federation Big Data Workflow Placement*" (ExactFed_BDWP) qui se base sur le modèle ILP (acronyme de Integer Linear Programming) et la méthode de résolution de *branch-and-bound*. Nous abordons l'algorithme de ExactFed_BDWP décrivant un ensemble de contraintes du problème de gestion et de placement des données intermédiare cité auparavant. Nous formalisons la fonction objective à optimiser ainsi que les contraintes linéaires sous forme d'un ensemble d'équations et d'inégalités valides d'écrivant le problème. Certaines de ces contraintes sont identifiées en se basant sur les systèmes réels de placement de données dans le Cloud en considérant les scénarios et les services de stockage proposés par les fournisseurs de Cloud. Trouver un placement optimal et économique des données générées par les applications de type *workflow* nécessite le calcul du coût de stockage pour chaque solution possible à partir de chaque requête de placement de données intermédiaire sur les centres de données fédérés et locaux. Les centres de données fédérés utilisent un modèle de coût dans l'algorithme de ExactFed_BDWP en fonction des dépendances inter-fichiers (entre paire de fichiers) pour acheminer et placer les requêtes de stockage de données intermédiaires dans le Cloud fédéré. La décision de la sélection du stockage doit conduire à un coût minimal pour le placement des données intermédiaire tout en respectant les contraintes de dépendances (un seul centre de données hébergeant les fichiers de dépendance) et une utilisation maximale de stockage pour les centres de données fédérés. Par conséquent, nous exposons le modèle de coût et la fonction objective à travers le problème 1 qui devrait être minimisée compte tenu des procédures suivantes pour le fonctionnement de l'algorithme de ExactFed_BDWP :

1. À partir d'un ensemble de paramètres d'entrées $Dep_{i,j}$ (équation (A.2)) concernant les instances d'un *Workflow* traitant des données intermédiaires sur des centres de données résidentiels et qui font une demande d'acheminement de données vers la fédération de stockage après chaque traitement, chaque centre de données des membres de la fédération expose le coût d'externalisation et d'internalisation du stockage en fonction de l'équation (A.1) :

$$S = \frac{QCmax_k - S_{busy}}{QCmax_k} * (S_{price} - ME_{price}) + ME_{price} \tag{A.1}$$

$$Dep_{i,j} = \begin{cases} 1 & \text{dépendance entre deux fichiers } i \text{ et } j \\ 0 & \text{sinon.} \end{cases} \tag{A.2}$$

2. Pour chaque valeur de $Dep_{i,j}$, le coût de la solution de placement de données intermédiaires nouvellement calculé est comparé aux coûts de placement possible les moins élevés dans chaque centre de données fédéré. L'algorithme de ExactFed_BDWP se termine après que l'ensemble de toutes les solutions possibles ont été vérifié. L'algorithme de ExactFed_BDWP est exécuté sous les exigences des propriétés des données intermédiaires et les contraintes de capacités des centres de données sélectionnés $DChome_d$ hébergeant les données intermédiaires ainsi que ceux de la fédération.

$$\lambda_i^j = \begin{cases} 0 & i \text{ et } j \text{ sont traités dans le meme centre de données.} \\ 1 & \text{sinon.} \end{cases} \tag{A.3}$$

3. L'algorithme de ExactFed_BDWP conserve les paires de fichiers de dépendances dans un seul centre de données tout en économisant leurs coûts de transfert, de stockage et de transaction ou d'accès avec les contraintes de dépendances (équation (A.3)). Les données intermédiaires avec tolérance aux dépendances devraient être optimisées en fonction du coût des transactions des requêtes d'E/S émises entre les centres de données fédérés.

4. Le problème est résolu pour chaque instance du *Workflow* lorsqu'une nouvelle paire de fichiers de dépendance est générée à partir d'un centre de données de résidence. Un tel coût est supposé variable selon les caractéristiques de la fédération (les capacités disponibles et les prix dynamiques).

La fonction objective pour l'acheminement et le placement optimal et économique des données intermédiaires peut être exprimée par la minimisation de leurs coûts de transfert et de stockage dans les centres de données fédérés, où la variable de décision $x_{id}^k = 1$ est utilisée pour indiquer que la donnée intermédiaire i (fichier) est placé dans un centre de données et $x_{id}^k = 0$ le cas contraire. L'algorithme de ExactFed_BDWP minimise également le coût de transaction des E/S lorsque les dépendances de données intermédiaires sont routées séparément dans différents centres de données, c'est-à-dire lorsque la deuxième variable de décision $y_{ijdd'}^{kk'} = 1$. La fonction objective (*MinCost*) est soumise à plusieurs inégalités et

contraintes linéaires exprimées par le problème 3.

$$\begin{aligned}\text{Min Cost} &= \sum_{idk}^{d\neq k} x_{id}^{k} \cdot size_i \cdot (OSC_k + ITC_k + OTC_d) \\ &+ \sum_{idk}^{d=k} x_{id}^{k} \cdot size_i \cdot LSC_k \\ &+ \sum_{ijdd'k}^{d\neq k} y_{ijdd'}^{kk'} \cdot Dep_{ij} \cdot \lambda_i^j \cdot IOPC_{i,j}\end{aligned}$$

Subject to :

$\sum_{ik} x_{id}^{k} = IDN_d$	$\forall d = 1, ..., D, d \neq k$
$\sum_{dk} x_{id}^{k} = 1$	$\forall i = 1, ..., ID_i$
$x_{id}^{k} + x_{jd'}^{k} = 2$	$\forall i, j = 1, ..., ID_i, \forall k, d, d' = 1, ..., D$
$x_{id}^{k} + x_{jd'}^{k} \leq 1$	$\forall i, j = 1, ..., ID_i, \forall k, d, d' = 1, ..., D$
$x_{id}^{k} + x_{jd'}^{k'} - y_{ijdd'}^{kk'} \leq 1$	$\forall i, j = 1, ..., ID_i, i \neq j, \forall k, k', d, d' = 1, ..., D$
$\sum_{kk'dd'} y_{ijdd'}^{kk'} \leq \sum_{k} x_{id}^{k}$	$\forall d = 1, ..., D$
$\sum_{id}^{k\neq d} x_{id}^{k} \cdot size_i \leq SCF_k$	$\forall k = 1, ..., D$
$\sum_{id}^{k=d} x_{id}^{k} \cdot size_i \leq SCL_d$	$\forall k = 1, ..., D$
$\sum_{id}^{k\neq d} x_{id}^{k} \cdot size_i \cdot BCF_k \leq DBmax_k$	$\forall k = 1, ..., D.$
$\sum_{id} x_{id}^{k} \cdot size_i \leq SCmax_k$	$\forall k = 1, ..., D$
$\sum_{d} IDN_d = 1$	$\forall i = 1, ..., ID_i$
$Dep_{ij} = Dep_{ji}$	$\forall i, j = 1, ..., ID_i, \forall Dep_{ij} \in DEP$

Problem 3 – Problème de placement de données du *Big Data Workflow* dans un stockage de Cloud fédéré.

A.7 Contribution 3 : Algorithmes scalables

Cette partie aborde la dernière contribution, dont une nouvelle approche différenciée des travaux citées auparavant qui ne tiennent pas en compte les types de dépendance (inter-et intra-*job*). Pour cela, nous formulons d'abord le problème de placement de données intermédiaires de dépendances générées à partir de plusieurs applications de type *Workflow* (une application se réfère à un *job* qui s'exécute en plusieurs tâches) dans des centres de données distribués dans le Cloud. Nous proposons ensuite un modèle d'optimisation pour le problème de placement qui inclut les contraintes du type de dépendance. Le modèle proposé est combiné avec la minimisation des coûts de stockage en appliquant un algorithme exacte et une heuristique (SPL_LP et UNS_GREED_HEUR) tout en réduisant le problème de placement au

problème de MCMF (acronyme de Minimum Cost Multiple-Sources Multicommodity Flow) respectivement pour les dépendances intra-et inter-*job*. Étant donné que l'acheminement des données intermédiaires à partir de la nature des dépendances intra-*job* peut être divisé et placé dans de différents centres de données, le problème est appelé MCMF avec variable fractionnelle (*splittable*) qui est traité par l'algorithme de SPL_LP. Cela permet de réduire le coût potentiel du mouvement des données intermédiaires entre plusieurs centres de données comme le cas dans l'algorithme de ExactFed_BDWP. L'algorithme de SPL_LP traite le problème de placement des dépendances entre un ensemble de fichiers non synchronisé pouvant être fractionnés durant leur acheminement et leur placement, cela diffère complètement de l'approche dans ExactFed_BDWP (paire de fichiers non divisible). Par ailleurs, l'algorithme de UNS_GREED_HEUR concerne le problème de placement de données intermédiaires à partir des dépendances inter-*job*. Nous formulons ce problème comme des demandes atomiques ne pouvant pas être fractionnées, cependant, comme la plupart de ces problèmes sont NP-Difficile, il est compliqué d'obtenir une solution optimale basée sur des méthodes exactes. Les approches de *greedy* semblent être simples et efficaces qui ont fait leur preuve pour le problème de flux non divisible (*unsplittable flow*) [Kol03, Kry05, BBA07, CCGK07, PRF11], et sont faciles à mettre en œuvre et évoluent linéairement pour de très grandes instances. Ainsi, l'utilisation des concepts de *greedy* donne une bonne solution approximative à notre problème de placement de données intermédiaires. Les résultats expérimentaux prouvent que les algorithmes proposés sont très prometteurs en termes de minimisation de coûts de stockage, ainsi qu'en montrant que, même avec des conditions differentes, le rapport coût de l'algorithme de UNS_GREED_HEUR est proche de la solution fractionnaire optimale de SPL_LP.

A.7.1 Algorithmes de SPL_LP

L'algorithme de SPL_LP est un modèle exacte LP (acronyme de Linear Program) relaxé par des variables réelles et par l'inclusion de conditions valides exprimées sous forme de contraintes ou d'inégalités. À travers les contraintes du problème, le placement de données intermédiaire dans un graphe orienté $G = (DC \cup A, E)$ à l'intervalle de temps t concerne l'acheminement et le placement des quantités de données intermédiaires de dépendances nommées $\phi^M(t) \in \Phi^M$. Celles-ci sont considérée comme des flux de commodités continus d'une composante de dépendance nommée m générée dans plusieurs centres de données sources, qui seront stockées dans un ou plusieurs centres de données de destinations tout en économisant leurs coûts de transfert, de stockage et de mouvement (ou communication).

Soit $x^m_{i,j}(t) \in \mathbb{R}$, une quantité réel de donnée intermédiaire d'une composante de dépendance m reflétant le flux de dépendance de données intermédiaires transféré à partir d'un

centre de données source dc_i à l'intervalle de temps t vers un centre de données de destination dc_j à l'intervalle de temps $t+1$ sur un lien de transmission nommé $e_{i,j}$ dans le graphe G. Afin de tenir compte de la quantité de dépendances des données intermédiaires migrantes entre les centres de données de destinations dc_j et $dc_{j'}$, nous injectons une autre variable réel nommée $x^m_{j,j'}(t) \in \mathbb{R}$. Le but de la résolution du problème de placement des données du *Big Data Workflow* comme un problème de la variante fractionnaire de MCMF est de minimiser la fonction objective (équation (A.4)) sous les contraintes peuvant être formulées comme un modèle LP optimisé par rapport aux flux $x^M_{i,j}(t)$.

$$Minimize \ \ (C(w_{i,j}) + C(s_j) + C(w_{j,j'})) \tag{A.4}$$

Selon cette formulation, le problème 4 est un modèle LP et est donc polynomial. Cependant, l'optimisation est effectuée par rapport aux flux $x^m_{i,j}(t)$ qui est borné et contraint en raison de la quantité de dépendances de données intermédiaires $\phi^m_{a^R_i}(t)$ générée par une seule tâche nommée a^r_i (quantité atomique), cela converge l'algorithme LP en un temps non polynomial par rapport à l'augmentation de la taille de cette quantité sur de grandes instances.

Le volume de ces quantités atomiques rend le problème 4 encore plus difficile à résoudre, puisque le fractionnement des flux $x^m_{i,j}(t)$ devient marginal. Étant donné que les données d'une composante de dépendance ne peuvent pas être traitées séparément avant que d'autre données de cette même composante générée par des tâches voisines ne soient matérialisées, le problème de la version non fractionnelle (*unsplittable*) est encore plus difficile à résoudre qui est NP-Difficile [Asa00]. En raison de tout cela, nous introduisons une heuristique pour traiter la variante non fractionnelle (*unsplittable*) pour des instances à très grande échelle aves un délai très raisonnable.

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

Subject to :

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

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

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

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

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

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

$$\sum_{i \in DC} x^m_{i,j}(t) \leq \phi^m_i(t) \;\; \forall j, m, t$$

$$\sum_{m \in M} |\phi^m_i(t)| \cdot x^m_{i,j}(t) \leq s^{avail}_{i,j}(t) \;\; \forall i, j, t$$

$$\sum_{m \in M} \sum_{t \in T} |\phi^m_i(t)| \cdot x^m_{i,j}(t) \leq S_j \;\; \forall i, j$$

$$\sum_{i \in DC} x^m_{s_{source},i} = \sum_{j \in DC} x^m_{j,s_{sink}} \;\; \forall m \in M$$

$$C(w_{i,j}) = \sum_{i \in DC} \sum_{j \in DC} \sum_{m \in M} \sum_{t \in T} |\phi^m_i(t)| \cdot x^m_{i,j}(t) \cdot w_\phi \cdot c_{w_\phi}$$

$$C(s_j) = \sum_{i \in DC} \sum_{j \in DC} \sum_{m \in M} \sum_{t \in T} |\phi^m_i(t)| \cdot x^m_{i,j}(t) \cdot c_{s_j}$$

$$C(w_{j,j'}) = \sum_{i \in DC} \sum_{\substack{j,j' \in DC \\ j \neq j'}} \sum_{m \in M} \sum_{t \in T} |\phi^m(t) - \phi^m_{a^r_i}(t)| \cdot x^m_{j,j'}(t) \cdot w_\phi \cdot c_{w_\phi}$$

$$x^m_{i,j}(t), x^m_{j,j'}(t) : \quad \text{continuous}$$

$$\phi^m_{a^r_i}(t) : \quad \text{discrete}$$

Problem 4 – Problème de placement des données intermédiaires fractionnelles du *Big Data Workflow*

A.7.2 Approche de *greedy*

Le placement de dépendances de données intermédiaires du type intra-*job* est comparé aux solutions de SPL_LP et de ExactFed_BDWP afin de remédier aux temps d'exécution de ces deux algorithmes qui traitent cependant deux types de dépendances different. Cela nécessite le placement de la quantité de dépendances de données intermédiaires dans un seul centre de données de destination. Pour traiter ce cas, une solution naïve de *greedy* considère une commodité entière d'une composante de dépendance m générée à partir de différentes sources comme une source de flux unique, contrairement à l'algorithme de SPL_LP qui tolère de multiple flux fractionnés à partir d'une composante de dépendance m lors de la résolution du problème. Sous la solution non fractionnelle (*unsplittable*), une composante n'est jamais divisée, c'est-à-dire est fractionnée le long de plus d'un chemin pendant le routage et le placement de ces données intérmediaires. En outre, l'approche de *greedy* applique une procédure de routine dans un graphe spécifique G_p (dérivé de G) et suppose que la demande

minimale est inférieure ou égale à la capacité maximale des nœuds (centre de données de destination) dans G_p [Asa00]. Ainsi, ce dernier impliquant moins de connexion et la recherche locale de l'optimum sur un graphe spécifique qui réduit l'espace de recherche accélère l'exécution du temps de la solution de *greedy*. En tant que tel, un *framework* d'optimisation de *greedy* est proposé pour construire cette approche. Nous développons ensuite l'algorithme de UNS_GREED_HEUR basé sur le *framework* d'optimisation proposé. Dans ce qui suit, nous résumons l'enchainement de construction de l'approche de *greedy*.

1. **Framework d'optimisation de greedy** : L'idée fondamentale derrière le *framework* proposé est de réduire le problème de placement dans le cas de dependance inter-*job* à un problème de MCMF avec des valeurs de flux non fractionnelles dont plusieurs sources de composantes de dépendance dans un graphe de flux dirigé $G_p = (DC_p \cup A_p; E_p; u; c)$. Nous traitons ainsi les paramètres de G_p : c avec une fonction de coût dans $E \rightarrow R$, et u une fonction de capacité dans : $E \rightarrow R$.

   La première partie de la construction du graphe de *greedy* G_p concerne l'affectation des flux d'entrée des sources multiples. Dans cette phase, l'idée est de mapper toutes les données intermédiaires à leurs sources de résidences et aux tâches qui les a généré, puis mapper chacune à un nœud virtuel qui représente une composante réunissant les données de dépendance de plusieurs sources. Tous ces nœuds sont mappés à une super source.

   La deuxième partie de la construction du *framework* concerne l'identification des liens potentiels pour l'acheminement des dépendances de données intermédiaires sur les destinations des centres de données. Afin de mapper chaque nœud virtuel d'une composante de dépendance, nous avons considéré les capacités de chaque destination pouvant accueillir une ou plusieurs composantes de dépendance. Une hypothèse évidente de non-goulot d'étranglement qui a été faite dans cette variante, est qu'un nœud de destination d'un centre de données virtuel dc_{j_p} dans G_p a suffisamment de capacité pour satisfaire individuellement au minimum une composante de dépendance ϕ^m. Ainsi, dans le graphe d'origine G, les centres de données de destination qui n'ont pas la capacité de stockage disponible pour chaque composant de dépendance sont exclus pour la construction de G_p.

   Nous avons ensuite mappé chaque nœud virtuel d'une composante de dépendance aux nœud virtuel de destination pouvant les accueillir. Ensuite pour chaque lien d'acheminement, nous avons assigné une fonction de capacité et une fonction de coût qui calculent les capacités résiduelles à chaque intervalle de temps t, ainsi que le coût correspondant pour l'acheminement et le stockage de chacune des composantes. Tous les nœuds virtuels de destination sont mappés au nœud puit final.

2. **Algorithme de UNS_GREED_HEUR :** Comme l'heuristique donne des solutions de placement séquentiel, il n'y a pas de problème de congestion sur les différents liens d'acheminement partagés par les composantes de dépendance, donc l'algorithme prend en charge le placement des composantes entièrement à leur destination. Les étapes de fonctionnement de UNS_GREED_HEUR est résumé ci-dessous :
   (a) La première étape de l'algorithme est le trie des composantes de dépendance selon leurs tailles respectives. Toutes les composantes sont injectées dans une liste de vecteur nommée L_ϕ dans un ordre décroissant.
   (b) L'algorithme sélectionne la première composante dans la liste L_ϕ, et trouve le chemin qui satisfait la contrainte de conservation des flux depuis le nœud source vers le nœud puit.
   (c) Pour chaque solution d'acheminement de flux d'une composante, l'algorithme calcule le chemin le plus court nommé ShP_Φ à partir du nœud super source vers le nœud puit en fonction du coût de stockage minimal (fonction définie). Une fois le chemin trouvé, les flux de la composante sont supprimés itérativement dans la liste, et l'algorithme calcule les capacités résiduelles de chaque lien du graphe G_p. Ainsi, la composante acheminée est supprimée dans la liste, et l'algorithme répète la sous-procédure de l'étape (b) jusqu'à ce que toutes les valeurs des flux soient balayées.
   (d) Enfin, l'algorithme répète la sous-procédure de l'étape (c) jusqu'à $L_{\ phi} \longleftarrow \varnothing$, et vérifie les valeurs des flux les plus importantes itérativement. Ensuite, il rétablit les chemins les plus courts trouvés avec les coûts optimaux et indique pour chaque chemin la paire de composante acheminé et sa destination correspondante dans le graphe d'origine G.

3. **Complexité :** En résumé, la complexité moyenne identifiée de l'heuristique proposée pour trouver des solutions au problème de placement des dépendances de données intermédiaires (cas de inter-*job*) à partir des sources multiples est de $O(2M^2 + |DC|^2 \times (M^2 + 1) + M + |DC|)$ dans le pire des cas.

A.8 Conclusion & perspectives

L'objectif des travaux de recherche présentés dans cette thèse est de divulguer et de résoudre un énorme problème de niche pour la gestion de stockage des données intermédiaires des applications de *Workflow* dans un environnement de Cloud. En effet, l'infrastructure ou

les fournisseurs de Cloud sont aujourd'hui les solutions par défaut pour gérer, déployer et traiter les applications ainsi que leurs données complexes dans un environnement distribué tel que le Cloud. Plus précisément, cette thèse traite le problème consistant à placer des données intermédiaires résultantes des traitements des applications basées sur un *Workflow* en considérant ses multiples facettes, niveaux et exigences afin de fournir non seulement une solution spécifique, mais aussi une approche générique et complète pour la gestion des données du *Big Data Workflow*. Pour faire face à cette problématique, nous avons proposé une étude de cas pour comprendre les accès des données intermédiaires des application de MapReduce. Puis, nous avons entamé la résolution du problème de placement des données intermédiaires à travers différentes approches en amont, et trois stratégies de placement de flux de données ont été proposées tout en utilisant des modèles théoriques pour des scénarios différents :

1. Modèle et algorithme de prédiction à base de Markov pour la caractérisation du comportement des accès des fichiers *spill* des applications de MapReduce-Hadoop. Cette approche prédit les E/S interférées des données intermédiaires par les accès concurrents de ces applications. Le modèle de prédiction ansi que l'algorithme ont été évalué à l'aide de *micro-benchmarks* de type MapReduce représentant sur un environnement de cluster Hadoop. En outre, l'approche proposée fait acte d'introduction à la problématique de gestion du gros volume de données intermédiaires.
2. Modèle et algorithme pour le problème de placement du Big Data *Workflow* en considérant un environnement de stockage fédéré et coopératif multi-Cloud ainsi que les besoins des applications de données de *Workflow*. Cette approche traite des dépendances entre des paires de fichiers générées dans des centres de données de résidence. L'approche proposée bénéficie des avantages qu'offre un environnement fédéré comme l'internalisation/l'externalisation des ressources de stockage et réseau pour la gestion efficace et économique des données intermédiaires partagées par une communauté scientifique et/ou industriels. La validation de l'algorithme montre son efficacité pour un nombre d'instances pratique.
3. Modèles et algorithmes qui permettent de traiter different aspects à travers une solution exacte relaxé permettant de résoudre une classe de données d'un *Workflow* spécifique (dépendance intra-*job*), et une heuristique qui résout une autre variante (dépendance inter-*job*). Celle ci améliore en plusieurs ordres de grandeur l'execution des deux algorithmes exactes dans une infrastructure de Cloud incluant des centaines de centres de données distribués en un temps de convergence très pratique.

Dans ce manuscrit, nous avons présenté plusieurs variantes de notre problème de placement par des hypothèses restrictives sur les données d'un *Workflow* dans sa version générale. Néanmoins, nous pourrions nous intéresser à d'autre contraintes et exigences du problème.

En effet, nous pourrions améliorer l'algorithme de ExactFed_BDWP à travers la réduction des paramètres d'entrées de cet algorithme. Également, d'évoluer le modèle de fédération par un nouveau schéma de tarification qui capte mieux les exigences des applications de *Workflow* tel que le plan de réservation des ressources et leur contrainte de temps. De même, nous pourrions aussi s'intéresser à rendre le fonctionnement des approches proposées en mode *online* en ajoutant des prédicteurs pour les arrivées (dépendances des données intermédiaires) ainsi qu'à la gestion des tâches dans un environnement dynamique en évoquant la contrainte de temps.

Appendix B

Publications

International Journal

— Ikken, S., Renault, É., Tari, A. & Kechadi, M. T. Efficient Intermediate Data Placement for Improving Workflow of Big Data Processing in Cloud Datacentres. Submission in journal of Computer and System Sciences-Elsevier.

— Barkat, A., Diniz dos Santos, A. & Ikken, S. Open Source Solutions for Building IaaS Clouds. SCALABLE COMPUTING. PRACTICE AND EXPERIENCE, 16(2), 187-204. 2015 .

International Conference

— Ikken, S., Renault, É., Barkat, A., Tari, A. & Kechadi, M. T. Cost-Efficient Big Intermediate Data Placement in a Collaborative Cloud Storage Environment. In 2017 IEEE 19th International Conference on High Performance Computing and Communications "HPCC17". December 2017.

— Ikken, S., Renault, É., Barkat, A., Tari, A. & Kechadi, M. Efficient Intermediate Data Placement in Federated Cloud Data Centers Storage. In International Conference on Mobile, Secure and Programmable Networking (pp. 1-15) "MSPN2016" Springer International Publishing. June 2016.

— Ikken, S., Renault, É., Tari, A. & Kechadi, M. T. Toward Scheduling I/O Request of MapReduce Tasks Based on Markov Model. In International Conference on Mobile, Secure and Programmable Networking (pp. 78-89) "MSPN2015". Springer International Publishing. June 2015.

National Conference

— Ikken, S., Renault, É. & Kechadi, M. T. Intermediate Data I/O Interference Prediction from Co-scheduled Tasks in MapReduce-Hadoop Processing. SUCESS 2017.

— Ikken, S., Tari, A. & Kechadi, M. T. Organisation Sémantique des Métadonnées pour le Data Mining Haute Performance dans un Système de Stockage de Cloud Computing. In proceeding of Colloque of Optimisation et les Systèmes d'Information "COSI'2013". June 2013.

Bibliography

[ACC+14] Danilo Ardagna, Giuliano Casale, Michele Ciavotta, Juan F Pérez, and Weikun Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, 5(1):11, 2014.

[ADJ+10] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, and Harbinder Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *NSDI*, volume 10, pages 28–0, 2010.

[AJB11] Sandip Agarwala, Divyesh Jadav, and Luis A Bathen. icostale: adaptive cost optimization for storage clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 436–443. IEEE, 2011.

[AKO08] Parag Agrawal, Daniel Kifer, and Christopher Olston. Scheduling shared scans of large data files. *Proceedings of the VLDB Endowment*, 1(1):958–969, 2008.

[AS14] VP Anuradha and D Sumathi. A survey on resource allocation strategies in cloud computing. In *Information Communication and Embedded Systems (ICICES), 2014 International Conference on*, pages 1–7. IEEE, 2014.

[Asa00] Yasuhito Asano. Experimental evaluation of approximation algorithms for the minimum cost multiple-source unsplittable flow problem. In *ICALP Satellite Workshops*, pages 111–122, 2000.

[BBA07] Meriema Belaidouni and Walid Ben-Ameur. On the minimum cost multiple-source unsplittable flow problem. *RAIRO-Operations Research*, 41(3):253–273, 2007.

[BCA12] Tekin Bicer, David Chiu, and Gagan Agrawal. Time and cost sensitive data-intensive computing on hybrid clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 636–643. IEEE, 2012.

[BF05] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys (CSUR)*, 37(1):1–28, 2005.

[BHK+91] Mary G Baker, John H Hartman, Michael D Kupfer, Ken W Shirriff, and John K Ousterhout. Measurements of a distributed file system. In *ACM SIGOPS Operating Systems Review*, volume 25, pages 198–212. ACM, 1991.

[BKT13] David Bermbach, Tobias Kurze, and Stefan Tai. Cloud federation: Effects of federated compute resources on quality of service and cost. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pages 31–37. IEEE, 2013.

[BR01] Ivan D Baev and Rajmohan Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 661–670. Society for Industrial and Applied Mathematics, 2001.

[BYV08] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 5–13. Ieee, 2008.

[CBHTE10] Rory Carmichael, Patrick Braga-Henebry, Douglas Thain, and Scott Emrich. Biocompute: towards a collaborative workspace for data intensive bio-science. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 489–498. ACM, 2010.

[CCGK07] Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47(1):53–78, 2007.

[CPL16] Wuhui Chen, Incheon Paik, and Zhenni Li. Tology-aware optimal data placement algorithm for network traffic optimization. *IEEE Transactions on Computers*, 65(8):2603–2617, 2016.

[Dai05] Weizhen Dai. *A Query-based Approach to Workflow Process Dependency Analysis*. University of Waterloo, 2005.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DJ12] Maciej Drwal and Jerzy Józefczyk. Decentralized approximation algorithm for data placement problem in content delivery networks. In *Doctoral Conference on Computing, Electrical and Industrial Systems*, pages 85–92. Springer, 2012.

[DJL+13] Sebastian Dippl, Michael C Jaeger, Achim Luhn, Alexandra Shulman-Peleg, and Gil Vernik. Towards federation and interoperability of cloud storage systems. In *Data Intensive Storage Services for Cloud Environments*, pages 60–71. IGI Global, 2013.

[DJN+10] Tim Dörnemann, Ernst Juhnke, Thomas Noll, Dominik Seiler, and Bernd Freisleben. Data flow driven scheduling of bpel workflows using cloud resources. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 196–203. IEEE, 2010.

[DSL+08] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: the montage example. In *Proceedings*

of the 2008 ACM/IEEE conference on Supercomputing, page 50. IEEE Press, 2008.

[dSM15] Telmo da Silva Morais. Survey on frameworks for distributed computing: Hadoop, spark and storm. In *Proceedings of the 10th Doctoral Symposium in Informatics Engineering-DSIE*, volume 15, 2015.

[DSVK11] Christina Delimitrou, Sriram Sankar, Kushagra Vaid, and Christos Kozyrakis. Accurate modeling and generation of storage i/o for datacenter workloads. *Proc. of EXERT, CA*, 2011.

[EA12] Iman Elghandour and Ashraf Aboulnaga. Restore: reusing results of mapreduce jobs. *Proceedings of the VLDB Endowment*, 5(6):586–597, 2012.

[EMKL15] Mahdi Ebrahimi, Aravind Mohan, Andrey Kashlev, and Shiyong Lu. Bdap: a big data placement strategy for cloud-based scientific workflows. In *Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on*, pages 105–114. IEEE, 2015.

[FB13] Wei Fan and Albert Bifet. Mining big data: current status, and forecast to the future. *ACM sIGKDD Explorations Newsletter*, 14(2):1–5, 2013.

[GB14] Nikolay Grozev and Rajkumar Buyya. Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, 44(3):369–390, 2014.

[GGY+13] Sven Groot, Kazuo Goda, Daisaku Yokoyama, Miyuki Nakano, and Masaru Kitsuregawa. Modeling i/o interference for data intensive distributed applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 343–350. ACM, 2013.

[GHKS13] Lukasz Golab, Marios Hadjieleftheriou, Howard Karloff, and Barna Saha. Distributed data placement via graph partitioning. *arXiv preprint arXiv:1312.0285*, 2013.

[GLJ17] Yang Gao, Keqiu Li, and Yingwei Jin. Compact, popularity-aware and adaptive hybrid data placement schemes for heterogeneous cloud storage. *IEEE Access*, 5:1306–1318, 2017.

[Gob13] MaryAnne M. Gobble. Big data: The next big thing in innovation. *IRI Research-Technology Management,*, 56(1):64–67, 2013.

[Gro12] Sven Groot. Modeling i/o interference in data intensive map-reduce applications. In *Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on*, pages 206–209. IEEE, 2012.

[HBT+13] Jun He, John Bent, Aaron Torres, Gary Grider, Garth Gibson, Carlos Maltzahn, and Xian-He Sun. I/o acceleration with pattern detection. In *Proceedings of the*

22nd international symposium on High-Performance Parallel and Distributed Computing, pages 25–36. ACM, 2013.

[HGC+13] Tim Hegeman, Bogdan Ghit, Mihai Capota, Jan Hidders, Dick Epema, and Alexandru Iosup. The btworld use case for big data analytics: Description, mapreduce logical workflow, and empirical evaluation. In *Big Data, 2013 IEEE International Conference on*, pages 622–630. IEEE, 2013.

[HK10] Christina N Hoefer and Georgios Karagiannis. Taxonomy of cloud computing services. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 1345–1350. IEEE, 2010.

[HLST11] Michael Hogan, Fang Liu, Annie Sokol, and Jin Tong. Nist cloud computing standards roadmap. *NIST Special Publication*, 35, 2011.

[HPL13] Doan B Hoang, M Hoang Phung, and Elaine Lawrence. A collaborative task planning and development environment on the cloud/grid. In *Networks (ICON), 2013 19th IEEE International Conference on*, pages 1–6. IEEE, 2013.

[HSS+10] Thomas A Henzinger, Anmol V Singh, Vasu Singh, Thomas Wies, and Damien Zufferey. Flexprice: Flexible provisioning of resources in a cloud environment. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 83–90. IEEE, 2010.

[JDB14] Ward Jaradat, Alan Dearle, and Adam Barker. Workflow partitioning and deployment on the cloud using orchestra. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 251–260. IEEE Computer Society, 2014.

[JLS+11] Jiahui Jin, Junzhou Luo, Aibo Song, Fang Dong, and Runqun Xiong. Bar: An efficient data locality driven task scheduling algorithm for cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 295–304. IEEE, 2011.

[Kan09] Duckjin Kang. Accurate blktrace: ablktrace. 2009.

[KHCG09] Steven Y Ko, Imranul Hoque, Brian Cho, and Indranil Gupta. On availability of intermediate data in cloud computations. In *HotOS*, 2009.

[Kho] Tushar Khot. Markov chain learning on file access patterns with noisy data.

[KJH+14] Seong-Hwan Kim, Kyung-No Joo, Yun-Gi Ha, Gyu-Beom Choi, and Chan-Hyun Youn. A phased workflow scheduling scheme with task division policy in cloud broker. In *International Conference on Cloud Computing*, pages 76–86. Springer, 2014.

[KKC15] Sewoog Kim, Dongwoo Kang, and Jongmoo Choi. I/o characteristics and implications of big data processing on virtualized environments. *Appl. Math*, 9(2L):591–598, 2015.

[KL14] Andrey Kashlev and Shiyong Lu. A system architecture for running big data workflows in the cloud. In *Services Computing (SCC), 2014 IEEE International Conference on*, pages 51–58. IEEE, 2014.

[Kol03] Petr Kolman. A note on the greedy algorithm for the unsplittable flow problem. *Information Processing Letters*, 88(3):101–105, 2003.

[KR09] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[Kry05] Piotr Krysta. Greedy approximation via duality for packing, combinatorial auctions and routing. In *International Symposium on Mathematical Foundations of Computer Science*, pages 615–627. Springer, 2005.

[LCLZ14] Yin Lu, Yong Chen, Rob Latham, and Yu Zhuang. Revealing applications' access pattern in collective i/o for cache management. In *Proceedings of the 28th ACM international conference on Supercomputing*, pages 181–190. ACM, 2014.

[LD11] Xin Liu and Anwitaman 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*, pages 1052–1061. IEEE, 2011.

[LDU+] Abel W Lin, Lu Dai, Khim Ung, Steven T Peltier, and Mark H Ellisman. The telescience project: Transparent grid access for scientific communities. *Special Issue of Concurrency and Computation: Practice and Experience–Science Gateways at GGF14 (Submitted)*.

[LHHH14] Chia-Wei Lee, Kuang-Yu Hsieh, Sun-Yuan Hsieh, and Hung-Chang Hsiao. A dynamic data placement strategy for hadoop in heterogeneous environments. *Big Data Research*, 1:14–22, 2014.

[LNW+11] Xiao Liu, Zhiwei Ni, Zhangjun Wu, Dong Yuan, Jinjun Chen, and Yun Yang. A novel general framework for automatic and cost-effective handling of recoverable temporal violations in scientific workflow systems. *Journal of Systems and Software*, 84(3):492–509, 2011.

[LSD+14] Cheng Li, Philip Shilane, Fred Douglis, Darren Sawyer, and Hyong Shim. Assert (! defined (sequential i/o)). In *HotStorage*, 2014.

[LSWL16] Lihui Liu, Junping Song, Haibo Wang, and Pin Lv. Brps: A big data placement strategy for data intensive applications. In *Data Mining Workshops (ICDMW), 2016 IEEE 16th International Conference on*, pages 813–820. IEEE, 2016.

[MND+13] Marc X Makkes, Canh Ngo, Yuri Demchenko, Rudolf Strijkers, Robert Meijer, Cees de Laat, et al. *Defining intercloud federation framework for multi-provider cloud services integration*. IARIA, 2013.

[MR97] Tara M Madhyastha and Daniel A Reed. Input/output access pattern classification using hidden markov models. In *Proceedings of the fifth workshop on I/O in parallel and distributed systems*, pages 57–67. ACM, 1997.

[MSS16] Artan Mazrekaj, Isak Shabani, and Besmir Sejdiu. Pricing schemes in cloud computing: An overview. *International Journal of Advanced Computer Science and Applications*, 7(2):80–86, 2016.

[MT10] Marian Mihailescu and Yong Meng Teo. Dynamic resource pricing on federated clouds. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 513–517. IEEE Computer Society, 2010.

[MTK+15a] Ioannis Mytilinis, Dimitrios Tsoumakos, Verena Kantere, Anastassios Nanos, and Nectarios Koziris. I/o performance modeling for big data applications over cloud infrastructures. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pages 201–206. IEEE, 2015.

[MTK+15b] Ioannis Mytilinis, Dimitrios Tsoumakos, Verena Kantere, Anastassios Nanos, and Nectarios Koziris. I/o performance modeling for big data applications over cloud infrastructures. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pages 201–206. IEEE, 2015.

[MVML12] Rafael Moreno-Vozmediano, Rubén S Montero, and Ignacio M Llorente. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer*, 45(12):65–72, 2012.

[Net15] Cisco Visual Networking. Cisco global cloud index: Forecast and methodology, 2015-2020. *White paper*, 2015.

[Noo] Omar-Qais Noorshams. Modeling and prediction of i/o performance in virtualized environments.

[NPC14] Catalin Negru, Florin Pop, and Valentin Cristea. Cost optimization for data storage in public clouds: a user perspective. In *Proceedings of 13th International Conference on Informatics in Economy*, 2014.

[NPM+10a] Tomasz Nykiel, Michalis Potamias, Chaitanya Mishra, George Kollios, and Nick Koudas. Mrshare: sharing across multiple queries in mapreduce. *Proceedings of the VLDB Endowment*, 3(1-2):494–505, 2010.

[NPM+10b] Tomasz Nykiel, Michalis Potamias, Chaitanya Mishra, George Kollios, and Nick Koudas. Mrshare: sharing across multiple queries in mapreduce. *Proceedings of the VLDB Endowment*, 3(1-2):494–505, 2010.

[OR02] James Oly and Daniel A Reed. Markov model prediction of i/o requests for scientific applications. In *Proceedings of the 16th international conference on Supercomputing*, pages 147–155. ACM, 2002.

[Pet14] Dana Petcu. Consuming resources and services from multiple clouds. *Journal of Grid Computing*, 12(2):321–345, 2014.

[PRF11] Hamed Pirsiavash, Deva Ramanan, and Charless C Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1201–1208. IEEE, 2011.

[PSS10] Pranav Pathak, Mehedi Sarwar, and Sohum Sohoni. Markov prediction scheme for cache prefetching. In *Proceeding of 2nd Annual Conference on Theoretical and Applied Computer Science. November 5*, page 14, 2010.

[RAH12] Arkaitz Ruiz-Alvarez and Marty Humphrey. A model and decision procedure for data storage in cloud computing. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 572–579. IEEE Computer Society, 2012.

[Rat11] J Rath. Data center strategies. simplifying high-stakes, mission critical decisions in a complex industry. 2011.

[RB15] B Kezia Rani and A Vinaya Babu. Scheduling of big data application workflows in cloud and inter-cloud environments. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 2862–2864. IEEE, 2015.

[Ree04] Daniel A Reed. *Scalable Input/Output: achieving system balance*. MIT Press, 2004.

[Rei11] Andrew Reichman. File storage costs less in the cloud than in-house. *Forrester Research, Cambridge, MA*, 2011.

[RHZ15] Salma Rebai, Makhlouf Hadji, and Djamal Zeghlache. Improving profit through cloud federation. In *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*, pages 732–739. IEEE, 2015.

[RLZW16] Xiaoqi Ren, Palma London, Juba Ziani, and Adam Wierman. Joint data purchasing and data placement in a geo-distributed data market. *arXiv preprint arXiv:1604.02533*, 2016.

[RM15] B Rajasekar and SK Manigandan. An efficient resource allocation strategies in cloud computing. *International Journal of Innovative Research in Computer and Communication Engineering*, 3(2):1239–1244, 2015.

[SG01] Ron Sun and C Lee Giles. Sequence learning: from recognition and prediction to sequential decision making. *IEEE Intelligent Systems*, 16(4):67–70, 2001.

[SHW+16] Jie Song, HongYan He, Zhi Wang, Ge Yu, and Jean-Marc Pierson. Modulo based data placement algorithm for energy consumption optimization of mapreduce system. *Journal of Grid Computing*, pages 1–16, 2016.

[SMM+16] Georgios Skourletopoulos, Constandinos X Mavromoustakis, George Mastorakis, Jordi Mongay Batalla, and John N Sahalos. An evaluation of cloud-based mobile services with limited capacity: a linear approach. *Soft Computing*, pages 1–8, 2016.

[SRJ+16] Qian Sun, Melissa Romanus, Tong Jin, Hongfeng Yu, Peer-Timo Bremer, Steve Petruzza, Scott Klasky, and Manish Parashar. In-staging data placement for asynchronous coupling of task-based scientific workflows. In *Proceedings of the Second Internationsl Workshop on Extreme Scale Programming Models and Middleware*, pages 2–9. IEEE Press, 2016.

[STLM07] Bo Sheng, Chiu C Tan, Qun Li, and Weizhen Mao. An approximation algorithm for data storage placement in sensor networks. In *Wireless Algorithms, Systems and Applications, 2007. WASA 2007. International Conference on*, pages 71–78. IEEE, 2007.

[Str10] P Stryer. Understanding data centers and cloud computing. In *Global Knowledge Instructor*, (2010).

[STT09] Hadas Shachnai, Gal Tamir, and Tami Tamir. Minimal cost reconfiguration of data placement in storage area network. In *International Workshop on Approximation and Online Algorithms*, pages 229–241. Springer, 2009.

[TCTB11] Adel Nadjaran Toosi, Rodrigo N Calheiros, Ruppa K Thulasiram, and Rajkumar Buyya. Resource provisioning policies to increase iaas provider's profit in a federated cloud environment. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 279–287. IEEE, 2011.

[TLA+16] Zhuo Tang, Min Liu, Almoalmi Ammar, Kenli Li, and Keqin Li. An optimized mapreduce workflow scheduling algorithm for heterogeneous computing. *The Journal of Supercomputing*, 72(6):2059–2079, 2016.

[TTC15] Prasad Teli, Manoj V Thomas, and K Chandrasekaran. An efficient approach for cost optimization of the movement of big data. *Open Journal of Big Data (OJBD)*, 1(1):4–15, 2015.

[Tud14] Radu Tudoran. *High-performance big data management across cloud data centers*. PhD thesis, ENS Rennes, 2014.

[VON+15] Dan Vesset, CW Olofson, A Nadkarni, A Zaidi, B McDonough, D Schubmehl, S Bond, Sh Kusachi, Q Li, and Ph Carnelley. Idc futurescape: Worldwide big data and analytics 2016 predictions. *International Data Corporation*, 2015.

[VOP11] Ricardo Vilaça, Rui Oliveira, and José Pereira. A correlation-aware data placement strategy for key-value stores. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 214–227. Springer, 2011.

[VSPD+13] Gil Vernik, Alexandra Shulman-Peleg, Sebastian Dippl, Ciro Formisano, Michael C Jaeger, Elliot K Kolodner, and Massimo Villari. Data on-boarding in federated storage clouds. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 244–251. IEEE, 2013.

[WC16] Chase Q Wu and Huiyan Cao. Optimizing the performance of big data workflows in multi-cloud environments under budget constraint. In *Services Computing (SCC), 2016 IEEE International Conference on*, pages 138–145. IEEE, 2016.

[WCAL14] Jianwu Wang, Daniel Crawl, Ilkay Altintas, and Weizhong Li. Big data applications using workflows for data parallel computing. *Computing in Science & Engineering*, 16(4):11–21, 2014.

[WK09] Daniel Warneke and Odej Kao. Nephele: efficient parallel data processing in the cloud. In *Proceedings of the 2nd workshop on many-task computing on grids and supercomputers*, page 8. ACM, 2009.

[WKKB13] Guanying Wang, Aleksandr Khasymski, KR Krish, and Ali R Butt. Towards improving mapreduce task scheduling using online simulation based predictions. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pages 299–306. IEEE, 2013.

[WKQ+08] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[WL15] Dan Wang and Jiangchuan Liu. Optimizing big data processing performance in the public cloud: opportunities and approaches. *IEEE network*, 29(5):31–35, 2015.

[WLN+13] Zhangjun Wu, Xiao Liu, Zhiwei Ni, Dong Yuan, and Yun Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, pages 1–38, 2013.

[WuRILP17] Md Wasi-ur Rahman, Nusrat Sharmin Islam, Xiaoyi Lu, and Dhabaleswar K DK Panda. A comprehensive study of mapreduce over lustre for intermediate data placement and shuffle strategies on hpc clusters. *IEEE Transactions on Parallel and Distributed Systems*, 28(3):633–646, 2017.

[WZDL14] Mingjun Wang, Jinghui Zhang, Fang Dong, and Junzhou Luo. Data placement and task scheduling optimization for data intensive scientific workflow in multiple data centers environment. In *Advanced Cloud and Big Data (CBD), 2014 Second International Conference on*, pages 77–84. IEEE, 2014.

[XL15] Zichuan Xu and Weifa Liang. Operational cost minimization of distributed data centers through the provision of fair request rate allocations while meeting different user slas. *Computer Networks*, 83:59–75, 2015.

[XLX17] Qiufen Xia, Weifa Liang, and Zichuan Xu. The operational cost minimization in distributed clouds via community-aware user data placements of social networks. *Computer Networks*, 112:263–278, 2017.

[XXLZ16] Qiufen Xia, Zichuan Xu, Weifa Liang, and Albert Y Zomaya. Collaboration-and fairness-aware big data management in distributed clouds. *IEEE Transactions on Parallel and Distributed Systems*, 27(7):1941–1953, 2016.

[YLLQ12] Hailong Yang, Zhongzhi Luan, Wenjun Li, and Depei Qian. Mapreduce workload modeling with statistical approach. *Journal of grid computing*, 10(2):279–310, 2012.

[YS12] D Yoo and KM Sim. A locality enhanced scheduling method for multiple mapreduce jobs in a workflow application, 2012.

[YWWL13] Yi Yuan, Haiyang Wang, Dan Wang, and Jiangchuan Liu. On interference-aware provisioning for cloud-based big data processing. In *Quality of Service (IWQoS), 2013 IEEE/ACM 21st International Symposium on*, pages 1–6. IEEE, 2013.

[YYL+12] Dong Yuan, Yun Yang, Xiao Liu, Gaofeng Zhang, and Jinjun Chen. A data dependency based strategy for intermediate data storage in scientific cloud workflow systems. *Concurrency and Computation: Practice and Experience*, 24(9):956–976, 2012.

[YYLC10] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. A data placement strategy in scientific cloud workflows. *Future Generation Computer Systems*, 26(8):1200–1214, 2010.

[ZCLS16] Jinghui Zhang, Jian Chen, Junzhou Luo, and Aibo Song. Efficient location-aware data placement for data-intensive applications in geo-distributed scientific data centers. *Tsinghua Science and Technology*, 21(5):471–481, 2016.

[ZGG15] Deze Zeng, Lin Gu, and Song Guo. Cost minimization for big data processing in geo-distributed data centers. In *Cloud Networking for Big Data*, pages 59–78. Springer, 2015.

[Zha12] Zhuoyao Zhang. Processing data-intensive workflows in the cloud. 2012.

[ZLWC14] Linquan Zhang, Zongpeng Li, Chuan Wu, and Minghua Chen. Online algorithms for uploading deferrable big data to the cloud. In *INFOCOM, 2014 Proceedings IEEE*, pages 2022–2030. IEEE, 2014.

[ZXW16] Qing Zhao, Congcong Xiong, and Peng Wang. Heuristic data placement for data-intensive applications in heterogeneous cloud. *Journal of Electrical and Computer Engineering*, 2016, 2016.

[ZXZ+15] Qing Zhao, Congcong Xiong, Xi Zhao, Ce Yu, and Jian 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*, pages 928–934. IEEE, 2015.