

# Data Placement for Multi-Tenant Data Federation on the Cloud

Ji Liu<sup>1</sup>, Member, IEEE, Lei Mo<sup>2</sup>, Member, IEEE, Sijia Yang, Jingbo Zhou, Shilei Ji, Haoyi Xiong<sup>3</sup>, Senior Member, IEEE, and Dejing Dou<sup>4</sup>, Senior Member, IEEE

**Abstract**—Due to privacy concerns of users and law enforcement in data security and privacy, it becomes more and more difficult to share data among organizations. Data federation brings new opportunities to the data-related cooperation among organizations by providing abstract data interfaces. With the development of cloud computing, organizations store data on the cloud to achieve elasticity and scalability for data processing. The existing data placement approaches generally only consider one aspect, which is either execution time or monetary cost, and do not consider data partitioning for hard constraints. In this paper, we propose an approach to enable data processing on the cloud with the data from different organizations. The approach consists of a data federation platform named **FedCube** and a Lyapunov-based data placement algorithm. **FedCube** enables data processing on the cloud. We use the data placement algorithm to create a plan in order to partition and store data on the cloud so as to achieve multiple objectives while satisfying the constraints based on a multi-objective cost model. The cost model is composed of two objectives, i.e., reducing monetary cost and execution time. We present an experimental evaluation to show our proposed algorithm significantly reduces the total cost (up to 69.8%) compared with existing approaches.

**Index Terms**—Data federation, cloud computing, data sharing, data placement, multi-objective

## 1 INTRODUCTION

DATA sharing is the first step for the data-related collaborations among different organizations [1], for example, joint modeling with data from multi-party. Meanwhile, direct sharing of raw data with collaborators is difficult due to big volume and/or ownership [2], [3]. Data federation [4] virtually aggregates the data from different organizations, which is an appropriate solution to enable data-related collaborations without direct raw data sharing. Based on cloud service, data federation works as an intermediate layer to establish an abstract data interface. It provides a virtual data view, on which the involved organizations can collaboratively store, share and process data.

As high efficiency and low cost make it possible to lease resources, e.g., computing, storage, and network, at a large scale, a growing number of organizations tend to outsource their data onto the cloud. With the pay-as-you-go model, cloud computing (cloud) brings convenience to the organizations to store and process a large amount of data. Cloud services bring a large number of resources at different layers. A Virtual Machine (VM) is an emulator of a computer, which can be viewed as a computing node in a network [5]. Through the data storage services, unlimited data can be stored on the cloud. Cloud providers promise to provide three features, i.e., infinite computing resources available on-demand, dynamic hardware resource provisioning in need, machines and storage paid and released as needed [6]. Dynamic provisioning enables cloud tenants/users to construct scalable systems with reasonable cost on the cloud [7]. With these features, the scientific collaboration on the cloud among different organizations becomes a practical solution.

Despite the advantages of cloud computing, data security issue on the cloud tends to be serious. When the data is stored on the cloud, it is crucial to keep confidentiality. Only the authorized tenants/users should have access to the data [8]. Encryption is a conventional way to keep the data confidential, such as identity-based encryption [9]. In addition, the isolation techniques [10], which provide secure execution spaces for different jobs with specific access controls, are also used to control the accessibility to the data on the cloud. A job is composed of a data processing program or a set of data processing programs to be executed on the cloud in order to generate new knowledge from the input data. During the scientific collaboration based on the data stored on the cloud, the combination of encryption algorithms and

- Ji Liu, Haoyi Xiong, and Dejing Dou are with Big Data Lab, Baidu Inc., Beijing 100085, China. E-mail: {jiliuwork, xhyccc}@gmail.com, liuji04@baidu.com.
- Lei Mo is with the School of Automation, Southeast University, Nanjing 211189, China. E-mail: lmo@seu.edu.cn.
- Sijia Yang is with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China. E-mail: anneryang\_1230@126.com.
- Jingbo Zhou is with Business Intelligence Lab, Baidu Inc., Beijing 100085, China. E-mail: zhoujingbo@baidu.com.
- Shilei Ji is with Security Department, Baidu Inc., Beijing 100085, China. E-mail: jishilei@baidu.com.

Manuscript received 7 March 2021; revised 21 November 2021; accepted 14 December 2021. Date of publication 20 December 2021; date of current version 7 June 2023.

This work was supported in part by the National Key R&D Program of China under Grant 2018YFB1402600, in part by the Fundamental Research Funds for the Central Universities under Grant 2242021R10113, in part by Southeast University “Zhishan Scholars” under Grant 2242021R40003, and in part by the Natural Science Foundation of Jiangsu Province under Grant BK20210218. (Corresponding author: Ji Liu.)

Recommended for acceptance by J. Wang.

Digital Object Identifier no. 10.1109/TCC.2021.3136577

isolation techniques can be utilized to keep the confidentiality and security of the data on cloud.

When using the cloud services, tenants/users have to pay for them. For instance, when tenants/users directly store their data on the cloud, they would be charged for the cloud storage service. Widely used cloud service providers, such as Amazon Web Services (AWS) cloud,<sup>1</sup> Microsoft Azure cloud<sup>2</sup> and Baidu cloud,<sup>3</sup> provide different data storage types, e.g., hot data storage, data storage with low frequency, cold data storage, and archive data storage, as data storage services. The cost of data storage on the cloud varies from type to type. In order to reduce the monetary cost to store and to process the data on the cloud, it is necessary to choose a proper data storage type based on a data placement algorithm. However, the job execution frequency is not well exploited while constructing the data placement algorithms for the data storage on the cloud. In addition, existing approaches cannot exploit data partitioning techniques to satisfy multiple constraints.

There are multiple constraints for the data processing on the cloud. For instance, when a user requires that the execution of a job should be within a time period, there is a hard time deadline. When a user has a budget limit for the data processing, there is a hard monetary budget for the execution of jobs. In addition, when the system is stable, the jobs can be continuously executed. Otherwise, there may be storage errors during the execution when the accumulated stored data exceed the storage capacity. Thus, the system stability is critical as well.

In this paper, we propose a solution to enable data processing on the cloud for scientific collaboration among different organizations. The solution consists of a secure data processing platform named **FedCube**, a multi-objective cost model, and a Lyapunov-based data placement algorithm. The main contributions of this paper are:

- The **FedCube** platform. We propose a cloud platform, i.e., **FedCube**. **FedCube** enables secure data processing with the encrypted data stored on the cloud for collaboration among different organizations.
- A data placement problem formulation. We formulate the data placement problem based on a multi-objective cost model and constraints. The multi-objective cost model consists of monetary cost and execution time. The constraints include hard execution time deadline, hard monetary budget, and system stability constraint.
- A Lyapunov-based data placement algorithm. We use the algorithm to create a data storage plan based on the cost model in order to reduce both monetary cost and the execution time of jobs with the consideration of constraints while exploiting data partitioning techniques.
- An extensive experimental evaluation based on a simulation and a widely used benchmark, i.e., Wordcount, and a real-life data processing application for COVID-19 [11]. The simulation and the experiments

are carried out based on a widely used cloud, i.e., Baidu cloud.

The rest of the paper is organized as follows. Section 2 reviews related works. Section 3 presents the system design of the secure data processing platform. Section 4 presents the data placement system model, proposes a cost model, shows the hard constraints, and defines the problem. Section 5 proposes the Lyapunov-based data placement algorithm based on the cost model. Section 6 shows the experimental results. Finally, we conclude the paper in Section 7.

## 2 RELATED WORK

Lyapunov optimization is widely used to optimize the system while ensuring system stability. For instance, Lyapunov optimization is exploited to gain profit [12], to ensure the Quality of Service [13] and the time average sensing utility [14]. However, the aforementioned work focuses on a single objective besides the system stability and does not consider the task or data partitioning for satisfying multiple constraints. In this paper, we combine the Lyapunov optimization with multiple objectives for data placement.

Data placement is critical to both the monetary cost and the execution time of jobs. In order to reduce the execution time, data transfer can be reduced based on graph partitioning algorithm [15]. In addition, the data dependency among different jobs can be exploited to reduce the time and monetary cost to transfer data [16]. However, these methods only consider one objective, i.e., reducing execution time. They cannot be applied to place the data in different storage types on the cloud. A weighted function of multiple costs can be used to achieve multiple objectives, which can generate a Pareto optimal solution [17], while the authors do not consider the cost to store data on the cloud or hard constraints. Load balancing algorithms [18] or dynamic provisioning algorithms [19] are proposed to generate an optimal provisioning plan in order to minimize the monetary cost while they do not consider the data storage types on the cloud. The storage type of the best performance can be selected to store data [20] while the economic storage type can be selected [21]. However, these two methods cannot address multiple objectives. In this paper, we propose an algorithm to achieve multiple objectives by placing data into various data storage types while satisfying hard constraints.

In order to handle a multi-objective problem, there are basically two types of solutions, i.e., *a priori* and *a posteriori* [17], [22]. In this paper, we use an *a priori* method, where the preference information is provided by the users, and then the best solution is produced. Our approach is based on a multi-objective scheduling algorithm focusing on minimizing a weighted sum of objectives. The advantage of such approach is that the scheduling is automatically guided by predetermined weights. In contrast, *a posteriori* methods produce a Pareto front of solutions without predetermined preference information [22]. Each produced solution is better than the others with respect to at least one objective, and users need to choose one from the produced solutions, which corresponds to user interference. In this paper, we assume that users can determine the value for the weight of each objective. *a priori* methods can enable us to produce optimal or near-optimal solutions without user interference

1. <https://aws.amazon.com/>

2. <https://azure.microsoft.com/en-us/>

3. <https://cloud.baidu.com/>

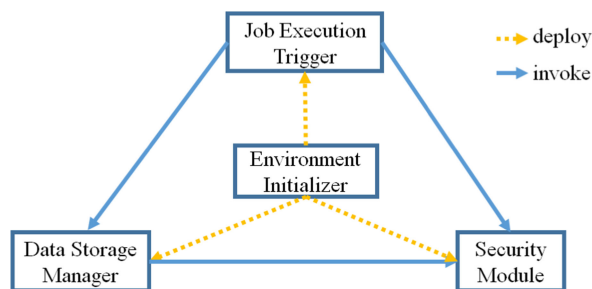


Fig. 1. The functionality architecture of the **FedCube** platform.

at run-time. Finally, when the weight of each objective is positive, the minimum of the weighted cost function is already a Pareto optimal solution [23], [24] and our proposed approach can generate a Pareto optimal or near-optimal solution with the predefined weights. Thus, we do not consider *a posteriori* methods in this paper.

Data security is of much importance to the cloud users. In order to protect data security, data accessibility is controlled by attributing different levels of permission to avoid unauthorized or malicious access to data on the cloud [25]. In addition, encryption techniques [26], [27] and distributed data storage plan based on data partitioning [28], [29], [30] can be exploited. Federated learning is proposed to train a model while ensuring data privacy [31], yet it is not applicable to the general data processing among different organizations on the cloud. In addition, secure separated data processing spaces [10] are proposed to ensure the access control and privacy of data. The separated data processing spaces are disconnected from the public network, which ensures that the confidentiality and the security of data within the local network. Our proposed platform, i.e., **FedCube**, not only provides different data access controls for different tenants/users but also exploits the secure separated data processing spaces to ensure the security and the confidentiality of the data.

### 3 SYSTEM DESIGN

In this section, we propose a secure data processing platform named **FedCube**. First, we explain the architecture of the platform. Then, we present the life cycle of users' accounts and jobs to be executed.

#### 3.1 Architecture

The **FedCube** platform is a data federation platform that provides tenants/users with secure data processing service on the cloud. Tenants/users can upload their data onto the platform and execute the self-written programs on Baidu cloud. In addition, tenants/users can leverage the data from other organizations for their own data processing jobs, as long as they get permission from the data owners. We illustrate the architecture of the platform and explain the functionalities of each module in this section. As shown in Fig. 1, the functional architecture of the platform consists of four modules:

##### 3.1.1 Environment\_INITIALIZER

The environment initializer creates the user account and its execution space on the coordinator node. The created user account is used for the user's security configuration, e.g.,

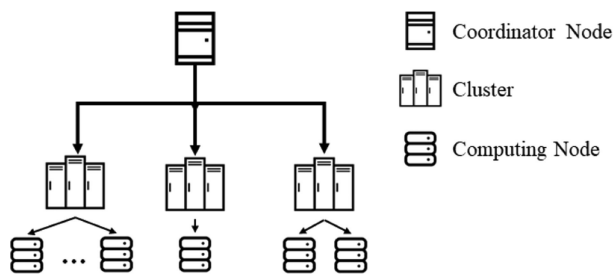


Fig. 2. Infrastructure architecture of the **FedCube** platform.

the access permission to certain data from another user. The user account is also associated with secure execution spaces for the execution of submitted jobs in the cluster. The secure execution space is a working space without a connection to any public network, which can ensure the confidentiality and the security of the data within the local network.

As shown in Fig. 2, multiple clusters can be dynamically created by the environment initializer module when the execution of jobs is triggered. Each cluster consists of several computing nodes, i.e., VMs on the cloud. The coordinator node coordinates the execution among different clusters for all users. The user has access to the platform through the coordinator node, which is connected to the public Internet. The computing nodes in each cluster are only interconnected with the coordinator node through the local network on the cloud. Each computing node is created based on the image [32] indicated by the user, which contains necessary tools for the execution of her jobs. An image is a serialized copy of the entire state of a VM stored on the cloud [32].

##### 3.1.2 Data Storage Manager

The data storage manager creates a data storage account and storage buckets on the cloud for a user. A storage bucket is a separated storage space to store the data with its own permission strategy. The data storage account is used to transfer data between the platform and the user's devices, e.g., computer. Each account is associated with five buckets, i.e., user data bucket, user program bucket, output data bucket, download data bucket, and execution space bucket. Each account has an independent Authorization Key (AK) and Secret Key (SK), with which the tenants/users can send or retrieve the data stored in the buckets. In addition, the access permission strategy varies from bucket to bucket. For instance, the user has read and write permission on the user data bucket and the user program bucket while she only has the read permission on the download data bucket. A user can store data in the user data bucket while she can submit self-written codes to the user program bucket. The tenants/users do not have read or write permission on the output data bucket and the execution space bucket. After the execution of the program generated based on the submitted codes, the output data is stored in the output data bucket. After the confidentiality review of the output data, the output data is transferred to the download data bucket. The review is carried out by the owner of the input data of the job in order to avoid the risk that the raw data or sensitive information appears in the output data of the job. The execution space bucket is used to cache intermediate data of

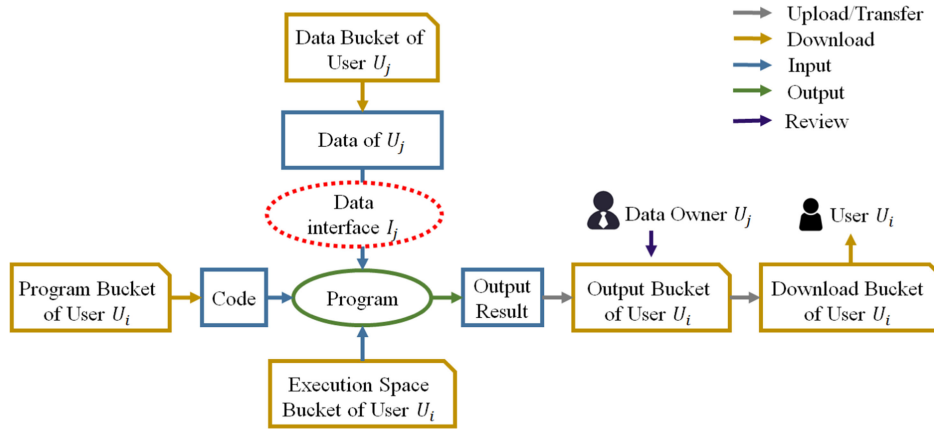


Fig. 3. Job execution workflow.

a job, which can be useful for the following execution in order to reduce useless repetitive execution [33].

### 3.1.3 Job Execution Trigger

The job execution trigger starts the execution of the job in a cluster. A user can upload the user-written codes onto the platform through a web portal. Then, she can start the execution of the program using the job execution trigger. Once the execution of the program is triggered, a cluster is created, deployed, and configured (see details in Section 3.1.1). Afterward, the execution of the job is performed in the computing nodes of the cluster. When several jobs start simultaneously in the same cluster, the job execution trigger creates the same number of execution spaces as that of jobs in order to enable parallel execution without conflict. When the input data of a program consists of the data from other data owners, the corresponding data interfaces are used in order to avoid direct raw data sharing. Let us take two tenants/users as example: User  $U_1$  and User  $U_2$ . A data interface ( $I_1$ ) is defined by the data owner (User  $U_1$ ), which is associated with the data ( $D_1$ ) on the platform. When User  $U_2$  gets the permission to use  $D_1$ , the program generated based on the submitted codes of User  $U_2$  can process the data  $D_1$  using the Interface  $I_1$ . The intermediate data stored in the execution bucket can also be used when the job needs the results of the previous execution.

### 3.1.4 Security Module

In the platform, we use four mechanisms to ensure the security of the data. The first mechanism is to encrypt the data before storing it on the cloud. The encryption is based on the Rijndael encryption algorithm [34]. The second mechanism is to separate the computing nodes from the public network, e.g., Internet, which ensures that no data communication is allowed between the clusters and outside devices, e.g., servers, on the cloud. The third mechanism is a uniformed data access control. When a user applies for the permission of the data owned by another user, a data access interface is provided by the data owner instead of direct raw data sharing. The last mechanism is the audition of the codes and output data by data owners, which ensures that no data is leaked from output data. Through these mechanisms, data confidentiality and security are ensured by the data interface defined by the data owner while ensuring efficient cooperation among different organizations.

## 3.2 Life Cycle

In order to present the interactions among users, the platform, and the job execution on the platform, we present the account life cycle and job life cycle. The life cycle describes the state transition of a user account or a job on the platform. We assume that there are  $n$  scientific collaborators. Each collaborator has private data, which requires keeping confidentiality and security. Through the life cycle, we present how  $n$  collaborators process the data on the platform.

### 3.2.1 Account Life Cycle

The account life cycle consists of three phases, i.e., account creation, data processing, and account cleanup. First, the account related to the user of the platform is created. Then, the user can process the data on the platform. Finally, when the user no longer needs the platform, the data related to the account is removed.

*Account Creation Phase.* When a new user needs to use the platform, we create an account and configure the platform using the environment initializer module as shown in Fig. 1. For the  $n$  collaborators in the above scenario, we create  $n$  accounts ( $U_i$  with  $t$  representing the number of the collaborator) for each scientific collaborator on the platform. First, the job execution trigger is deployed for each user in the coordinator node. Then, the data storage manager creates a storage account and five storage buckets (see details in Section 3.1.2) for each user. Afterward, the environment initializer deploys the security module for each user. The security module contains the encryption and decryption information for each user. Please note that the encryption and decryption information is different for different users.

*Data Processing Phase.* After the account creation, data processing jobs can be carried out on the platform. Before processing the data, each user uploads her own data and the data interface file to the user data bucket. As shown in Fig. 3, if User  $U_i$  needs to exploit the data from another Users  $U_j$ , User  $U_i$  can apply for the permission. Once User  $U_i$  gets the permission from User  $U_j$ , the user also gets the necessary information, e.g., the mock data, to access the data using the corresponding data interface. The mock data contains the data schema of the raw data and some randomly generated examples, while the raw data is never shared with the users. User  $U_i$  may use the data from several other tenants/users at the same time. Then, User  $U_i$  can



submit the codes to process data. In order to process data, User  $U_i$  triggers the execution of a job related to the submitted codes (see details in Section 3.2.2), which corresponds to the execution of the job ( $j_i$  with  $i$  representing the number of the execution) on the platform. During the execution of a job, the intermediate data generated from different execution of the job can be directly used. After the execution and the review of the output data, user  $U_i$  can download the output data of Job  $j_i$  from the user download bucket.

*Account Cleanup Phase.* When the user no longer needs the platform, the corresponding data, storage buckets, and accounts are removed from the platform by the environment initializer module.

*Initialization Phase.*

### 3.2.2 Job Life Cycle

The job life cycle consists of four phases, i.e., initialization, data synchronization, job execution, and finalization.

The initialization phase [35] is to prepare the environment to execute a job on the platform. The preparation contains three steps: provisioning, deployment, and configuration. First, VMs are provisioned to the job as computing nodes. There are two cases where existing VMs can be provisioned to the job. The first case is that there are enough live computing nodes on the platform corresponding to the execution of the same or the other jobs of the same user. The second case is that there are enough live computing nodes for the programs of other tenants/users, and all the related tenants/users allow sharing computing nodes. Otherwise, the environment initializer module dynamically creates new VMs as computing nodes, which contain necessary tools for the execution of the job. Then, in order to execute the job, a proper execution space is deployed on the allocated VMs. In order to enable data access, the execution space is configured in each node. For instance, the AK and SK files are transferred into the computing nodes in order to enable data synchronization.

*Data Synchronization Phase.* During the data synchronization phase [36], the data storage module synchronizes the data or data interfaces stored on the cloud. In addition, the scripts or the files corresponding to the submitted codes are also transferred to the execution space created in the initialization phase.

*Job Execution Phase.* The execution phase [35] is the period to execute jobs in the execution space of corresponding VMs. The execution frequency of each job can be dynamically monitored by the platform to compute the cost of data storage. The data, including newly generated intermediate data, is dynamically placed with appropriate storage types with small cost according to the method presented in Section 5. As shown in Fig. 3, after synchronizing the data from buckets, the program corresponding to the submitted codes processes the input data. The execution can be performed in a single computing node or multiple computing nodes in order to reduce the overall execution time. After the execution, the output data is transferred to the output bucket of the user. Once the data is reviewed and approved by the data owners of the input data, it is encrypted by the security module and is transferred to the download bucket to be accessed by the user.

*Finalization Phase.* In the finalization phase [37], the data storage manager uploads the encrypted intermediate of the

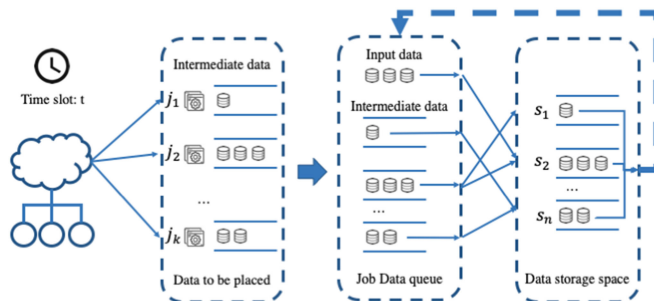


Fig. 4. System model for data placement.

job. Afterward, the environment initializer module removes the corresponding execution space(s). If a node does not contain any execution space, the node is released, i.e., removed, by the environment initializer, in order to reduce the monetary cost to rent the corresponding VMs.

## 4 MULTI-OBJECTIVE COST MODEL AND PROBLEM FORMULATION

In this section, we first present the system model for data placement. Then, we propose a cost model based on two costs, i.e., monetary cost and execution time. Afterward, we present the data placement constraints, i.e., hard execution time constraints and the hard monetary budget constraints. Finally, we define the problem to address in the paper.

### 4.1 Data Placement System Model

The system model for data placement is shown in Fig. 4. In the **FedCube** platform, we assume that the execution of jobs generates intermediate data at time slot  $t$ , which may be used as input data in the following time slots, e.g.,  $t + x$  with  $x > 0$ . Then, the intermediate data should be placed with other input data. Each job has a queue to store the generated intermediate data, and we consider  $N$  data storage spaces, which correspond to  $N$  storage types with diverse data access speeds and diverse prices to store data. Each data set can be placed to one or multiple data storage types. In order to place a data set to multiple storage types, a data set can be partitioned into several chunks, and each chunk is placed to a data storage type. We assume that the valid time of data set  $d_i$  placed at storage type  $s_j$  is  $T_{max(i,j)}$ . If data set  $d_i$  is not accessed by any job within  $T_{max(i,j)}$ , the data set will be removed from the storage space of the platform. When there is a data set generated during the execution of a job or when a job is executed, all the input data is placed again. When the input data is being replaced, its original corresponding storage type is kept until a newly placed storage type is associated.

### 4.2 Cost Model

Inspired by [17], we propose a multi-objective cost model. The cost model is composed of monetary cost and time cost (i.e., the execution time of a job). In order to find a storage plan, we need a cost model to estimate the cost of storing the input data for the execution of jobs. The cost model is generally implemented in the data storage module and under a specific execution environment. In the case of this paper, the execution environment is the **FedCube** platform.

The origin of parameters mentioned in this section is

TABLE 1  
Description of Parameters

Abbreviation	Parameter	Meaning	Origin
DT	DesiredTime	The estimated execution time to execute a job	UD
DM	DesiredMoney	The estimated monetary cost to execute a job	UD
TDL	TimeDeadline	The hard execution time deadline to execute a job	UD
MB	MoneyBudget	The hard monetary budget to execute a job	UD
AIT	averageInitializationTime	The average time to initialize a computing node	Measure
CSP	ComputingSpeedPerCPU	The average computing performance of each computing node	Measure
WL	workload	The workload of a job	Measure
$\alpha_k$	$\alpha_k$	The percentage of the workload of Job $k$ that can be executed in parallel	Measure
speed	speed	The data transfer speed for a type of data storage	cloud
SP	StoragePrice	The monetary cost to store data with a storage type	cloud
RP	ReadPrice	The monetary cost to read data from a cloud storage service	cloud
VMP	VMPPrice	The monetary cost to use a VM	cloud

“Abbreviation” represents the abbreviation of the parameters. “Origin” represents where the value of the parameter comes from. UD: that the parameter value is defined by users; Measure: that the parameter value is estimated by the user with the job in a cloud environment; Execution: measured during the execution of job in cloud; cloud: the parameter value is obtained from the cloud provider.

summarized in Table 1. We assume that there are  $K$  jobs on the platform.

The total cost to execute a set of jobs with a data placement plan at time slot  $t$  is defined as the sum of the total cost of all the jobs

$$\text{TotalCost}(Plan[t]) = \sum_{k=1}^K \text{Cost}(job_k, Plan[t]), \quad (1)$$

where  $Plan[t]$  represents a data placement plan of the data sets related to the set of jobs at time slot  $t$ , and  $j_k$  represents the  $k^{th}$  job. In the rest of this paper, the total cost represents the normalized cost to execute a set of jobs with a data placement plan per time unit.  $Plan[t]$  is a matrix of data placement variables, which can be expressed by the following formula:

$$Plan[t] = \begin{bmatrix} p_{0,0}[t] & p_{0,1}[t] & \dots & p_{0,n}[t] \\ p_{1,0}[t] & p_{1,1}[t] & \dots & p_{1,n}[t] \\ \dots & \dots & \ddots & \dots \\ p_{m,0}[t] & p_{m,1}[t] & \dots & p_{m,n}[t] \end{bmatrix}, \quad (2)$$

where  $p_{i,j}[t]$  represents that data set  $d_i$  is placed to storage type  $s_j$ ,  $m$  represents the number of input and intermediate data sets, and  $n$  represents the number of storage types. When  $p_{i,j}[t] = 0$ , data set  $d_i$  is not placed to storage type  $s_j$ ; when  $p_{i,j}[t] = 1$ , data set  $d_i$  is directly placed to storage type  $s_j$ ; When  $0 \leq p_{i,j}[t] \leq 1$ , data set  $d_i$  is partitioned and the part corresponding to  $p_{i,j}[t]$  is placed to storage type  $s_j$ .

The total cost to execute a job is defined by

$$\text{Cost}(job_k, Plan[t]) = w_m \cdot M_n(job_k, Plan[t]) \cdot f(job_k) + w_t \cdot T_n(job_k, Plan[t]), \quad (3)$$

where  $T_n(job_k, Plan[t])$  and  $M_n(job_k, Plan[t])$  are the normalized time cost and monetary cost, respectively, and they can be defined by Formulas (4) and (9);  $job_k$  represents the  $k^{th}$  job and  $Plan[t]$  represents the data placement plan at time slot  $t$ ;  $w_t$  and  $w_m$  represents the importance of the execution time and the monetary cost of the job. Defined by the user,  $w_t$  and  $w_m$  should be positive values that meet the constraints:  $0 \leq w_t \leq 1$ ,  $0 \leq w_m \leq 1$ ,  $w_t + w_m = 1$ .  $f(job_k)$

represents the average frequency of the job execution, which can be dynamically measured according to the history execution before the job execution, e.g., daily, monthly, quarterly and yearly. Since the time cost and monetary cost are normalized, neither of them has a unit. Please note that the time cost refers to the execution time of Jobs once while the hard monetary budget is related to the budget per time period, e.g., a day or a month.

#### 4.2.1 Time Cost

The normalized time cost is defined by the following formula:

$$T_n(job_k, Plan[t]) = \frac{T(job_k, Plan[t])}{DT_k}, \quad (4)$$

where  $T(j, Plan[t])$  represents the total execution time of the job and  $DT_k$  represents the expected execution time (set by the user) of Job  $job_k$ . Please note that the execution time  $T(j, Plan[t])$  represents the time to execute  $job_k$  once. The desired execution time could be larger or smaller than the real execution time  $\text{Time}(j, Plan[t])$  while it should be larger than a limit defined by the **FedCube** platform, e.g.,  $1/20 * \text{SET}_k$  with  $\text{SET}_k$  representing the sequential execution time of Job  $job_k$  with one computing node, in order to avoid the unfairness among users. The total execution time consists of three parts, which are defined by

$$T(job_k, Plan[t]) = \text{InitT}(job_k) + \text{DTT}(job_k, Plan[t]) + \text{ET}(job_k), \quad (5)$$

where  $\text{InitT}(job_k)$  represents the Time to Initialize the computing nodes for Job  $job_k$ ;  $\text{DTT}(job_k, Plan[t])$  represents the Time to Transfer the Data from the cloud storage service to computing nodes;  $\text{ET}(job_k)$  represents the Execution Time of Job  $job_k$ . The initialization of the computing nodes for Job  $job_k$  can be specified by the user or realized by the platform, which is out of the scope of this paper while the time can be calculated based on Job  $job_k$ , e.g.,  $n_k \cdot \text{AIT}$  with  $n_k$  representing the number of computing nodes and  $\text{AIT}$  representing the average time to initialize a computing node. The data transfer time can be calculated based on the size of the

input data of Job  $job_k$  and the data placement plan as follows:

$$DTT(job_k, Plan[t]) = \sum_{j=1}^N \sum_{i \in \text{data}_k} \frac{\text{size}(d_i)}{\text{speed}_j} \cdot p_{i,j}[t], \quad (6)$$

where  $\text{speed}_j$  represents the speed to transfer data from data storage type  $j$  to computing nodes. As explained in Section 4.1,  $N$  represents the total number of storage types on the **FedCube** platform. According to the Amdahl's law [38], the execution time of Job  $j$  can be estimated by the following formula [17] when the impact of data communication can be ignored with  $n$  computing nodes

$$ET(job_k) = \frac{[\alpha_k/n + (1-\alpha_k)] \cdot WL(job_k)}{CSP}, \quad (7)$$

where  $\alpha_k$ <sup>4</sup> represents the percentage of the workload that can be executed in parallel;  $n$  is the number of computing nodes, which is configured by users while it should be less than a limit, e.g., 20, defined by the **FedCube** platform;  $WL(j)$  represents the workload of a job which can be measured by the number of FLOP (FLOat-point Operations) [39].  $CSP$  is the average computing performance of each computing node, which can be measured by the number of FLOPS (FLOating-point Operations Per Second).

#### 4.2.2 Monetary Cost

Normalized monetary cost is defined by the following formula:

$$M_n(job_k, Plan[t]) = \frac{M(job_k, Plan[t])}{DM_k}, \quad (9)$$

where  $Money(job_k, Plan[t])$  is the financial cost to rent VMs as computing nodes on the cloud. Please note that the monetary cost  $Money(job_k, Plan[t])$  represents the total monetary cost to execute  $job_k$  within a time period, e.g., a month.  $DM_k$  represents the expected execution monetary cost of Job  $job_k$ , which can be configured by the user while it should be larger than or equal to a limit defined by the **FedCube** platform, i.e.,  $SMC_k$  with  $SMC_k$  representing the monetary cost of Job  $job_k$  with one computing node, in order to avoid the unfairness among users.  $DM_k$  can be bigger or smaller than the real monetary cost  $Money(job_k, t)$ .  $Money(job_k, Plan[t])$  can be estimated based on the following formula:

$$\begin{aligned} M(job_k, Plan[t]) &= EM(job_k, Plan[t]) \\ &+ DSM_k(job_k, Plan[t]) \\ &+ DAM_k(job_k, Plan[t]), \end{aligned} \quad (10)$$

where  $EM(job_k, Plan[t])$  represents the Monetary cost to use the computing nodes to Execute the job;  $DSM(job_k, Plan[t])$  represents the Monetary cost to Store the Data on the cloud

4.  $\alpha_k$  can be obtained by measuring the execution time of executing the job  $k$  twice with different numbers of computing nodes [17]. For instance, assume that we have  $t_1$  for  $m_1$  computing nodes and  $t_2$  for  $m_2$  computing nodes,

$$\alpha_k = \frac{m_2 * m_1 * (t_2 - t_1)}{m_2 * m_1 * (t_2 - t_1) + m_1 * t_1 - m_2 * t_2} \quad (8)$$

storage service;  $DAM(job_k, Plan[t])$  represents the Monetary cost to Access to the Data.  $EM(job_k, Plan[t])$  can be estimated by the following formula:

$$EM(job_k, Plan[t]) = VMP(job_k) \cdot n_k \cdot [T(job_k, Plan[t]) - \text{Init}T(job_k)], \quad (11)$$

where  $VMP(job_k)$  represents the average monetary cost of a VM for the execution of Job  $j$ ;  $n_k$  represents the number of computing nodes to execute the job;  $T(job_k, Plan[t])$  and  $\text{Init}T(job_k)$  are defined in Formula (5).

We allocate the storage monetary cost of a data set to the jobs based on the workload.  $DSM(j, Plan[t])$  is defined by the following formula:

$$DSM(job_k, Plan[t]) = \frac{WL(job_k)}{\sum_{l=1}^K (WL(job_l) \cdot f(job_l))} \cdot \sum_{j=1}^N \sum_{i \in \text{data}_k} (SP_j \cdot \text{size}(d_i) \cdot p_{i,j}[t]), \quad (12)$$

where  $WL(job_k)$  represents the workload of job  $job_k$ ;  $\text{dataset}(j)$  represents the data sets that job  $j$  uses;  $job(i)$  represents the jobs that takes data  $i$  as input data;  $SP(s_i)$  represents the monetary cost to store the data with the storage type  $s_i$ , which is defined in the data placement plan  $plan[t]$ , on the cloud;  $\text{size}(d_i)$  represents the size of the input data  $d_i$ .

$DAM(job_k, Plan[t])$  is defined by the following formula:

$$\begin{aligned} \text{DataAccessMoney}(job_k, Plan[t]) \\ = \sum_{j=1}^N \sum_{i \in \text{data}_k} (RP_j \cdot \text{size}(d_i) \cdot p_{i,j}[t]), \end{aligned} \quad (13)$$

where  $RP_j$  represents the monetary cost to read data  $d_i$  from the cloud storage service;  $\text{size}(d_i)$  represents the size of the input data  $d_i$  of Job  $job_k$ .

### 4.3 Data Placement Constraints

In this section, we present the constraints of data placement. First, we present the hard execution time and monetary budget constraints for each job. Then, we present the system stability constraint based on Lyapunov optimization.

We assume that there are hard time deadline and hard monetary budget for each job, which can be formulated as follows:

$$T(job_k, Plan[t]) \leq TDL_k, \quad \forall k \in [0, K], \quad (14)$$

$$M(job_k, Plan[t]) \leq MB_k, \quad \forall k \in [0, K], \quad (15)$$

where  $T(job_k, Plan[t])$  and  $M(job_k, Plan[t])$  are defined in Formulas (5) and (10) respectively,  $TDL_k$  represents the hard execution time deadline,  $MB_k$  represents the hard monetary cost Budget, and  $Jobs$  represents the set of jobs in the system.

For storage spaces, as shown in the right part of Fig. 4, we use  $S_j(t)$  to denote the set of data sets placed in the data storage space of Type  $j$ . Therefore, the dynamic set is defined as follows:

$$S_j(t+1) = \max[S_j(t) - r_j(t), 0] + \sum_{i=1}^M p_{i,j}[t], \quad (16)$$

TABLE 2

The Monetary Cost to Store Data on the Cloud With Different Storage Types, i.e., Standard, Low frequency, Cold and Achieve

	Standard	Low frequency	Cold	Achieve
Expected data access frequency	frequently	< once per month	< once per year	≥ three years
Cost to store data (Dollar/GB/month)	0.0155	0.0113	0.0045	0.015
Cost to read data (Dollar/GB)	N/A	0.0042	0.0085	0.12

where  $r_j(t)$  represents the data to be removed because of time limit and  $\sum_{i=1}^m p_{i,j}[t]$  represents the newly placed data sets to data storage type  $s_j$ .

For jobs shown in the middle part of Fig. 4, we use  $J_i$  to denote the set of data sets generated from the execution of Job  $i$ . We have the following job data storage set defined as follows:

$$J_k(t+1) = \max \left[ J_k(t) - \sum_{j=1}^N \sum_{i \in \text{data}_k} p_{i,j}[t], 0 \right] + G_k[t], \quad (17)$$

where  $\text{data}_k$  represents the set of input data sets of Job  $k$ , and  $G_k[t]$  represents the newly generated intermediate data of Job  $k$ .

We exploit the Lyapunov optimization technique [40] by considering both the set of data sets placed in the data storage spaces and the job data storage sets. Let  $D(t) = (S_j(t), J_i(t), j \in \{1, \dots, n\}, i \in \{1, \dots, k\}, t \in \{1, 2, \dots\})$  denote all the data sets in time slot  $t$ . We have the following constraint in order to ensure the stability of the system:

$$\bar{D} \triangleq \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \left( \sum_{j=1}^N \mathbb{E}\{S_j(t)\} + \sum_{k=1}^K \mathbb{E}\{J_k(t)\} \right) < \infty. \quad (18)$$

#### 4.4 Problem Definition

The problem we address in the paper is a data placement problem, i.e., how to choose a storage type to store the data in order to reduce the expected total cost, which consists of the monetary cost and the execution time of jobs, while satisfying constraints, on the cloud. A job can be executed multiple times because the user-defined codes are updated, or the parameters are updated [41]. As shown in Table 2, different data storage types of storage services on the cloud correspond to different prices. The storage type with higher expected data access frequency, e.g., Standard, has a higher price and higher data access speed. The total cost to execute a job once differs with different data placement plans. Thus, the problem we address in this paper is how to find an optimal data placement plan of all the data sets in order to reduce the expected total cost to execute the jobs with different execution frequencies based on a cost model. We define the expected total cost as

$$\overline{\text{Cost}}(\text{Jobs}, \text{Plan}[t]) = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\text{Cost}(\text{Jobs}, \text{Plan}[t])\}. \quad (19)$$

Then, the problem addressed in this paper can be formulated as follows:

$$\begin{aligned} & \min \overline{\text{Cost}}(\text{Jobs}, \text{Plan}[t]) \\ & \text{s.t.} \begin{cases} p_{i,j}[t] \in [0, 1] \\ \sum_{j=1}^N p_{i,j}[t] = 1, \\ \text{Formulas (14), (15), and (18)}. \end{cases} \end{aligned} \quad (20)$$

where  $\text{Jobs}$  represents the set of jobs in the system.

The data placement problem is a typical NP-hard problem [42]. Let us separate the data placement variables into two parts, i.e.,  $p'_{i,j}$  and  $p''_{i,j}$ .  $p'_{i,j}$  is a continuous variable between 0 and 1, which represents the partitioning of a data set.  $p''_{i,j}$  is a 0-1 integer, which is the scheduling decision. Then, we have  $p_{i,j} = p'_{i,j} * p''_{i,j}$ . Then, the problem defined in Formula (19) is a Mixed Integer Linear Programming (MILP) problem, which is a proven NP-hard problem [43], [44], [45]. In this case, the exhaustive search for an optimal solution for  $p''_{i,j}$  increases exponentially and the complexity is  $\mathcal{O}(N^M)$ , which cannot be solved within a polynomial time, with  $N$  representing the number of storage types and  $M$  represents the number of data sets.

## 5 NEAR-OPTIMAL DATA PLACEMENT

In this section, we present a near-optimal data placement approach based on Lyapunov optimization. Lyapunov optimization is widely used to achieve optimization objectives while ensuring the system stability [40], [46]. In order to exploit Lyapunov optimization techniques, we first construct a Lyapunov function and propose a Lyapunov-based algorithm (LNODP) to perform the data placement while ensuring system stability. Then, we propose a greedy approach to perform the near-optimal data placement while satisfying hard deadlines. The greedy approach consists of three algorithms, i.e., near-optimal data planning (NOD Planning), near-optimal data placement, and data placement (NOD Placement) with partitioning (NOD Partitioning). LNODP exploits NOD Planning to generate a near-optimal data placement plan; NOD Planning takes advantage of NOD Placement to choose optimal data storage type when the hard constraints can be satisfied, and NOD Placement uses NOD partitioning to generate a plan to partition the data in order to satisfy hard constraints when one data storage type does not work.

### 5.1 Lyapunov Optimization Based Data Placement

We define a Lyapunov function  $L(t)$  as follows:

$$L(t) \triangleq \frac{1}{2} \left( \sum_{j=1}^N [S_j(t)]^2 + \sum_{k=1}^K [J_k(t)]^2 \right). \quad (21)$$

This function represents the data sets to be placed. Then, we can define the derivative of the Lyapunov function as follows:



$$\frac{\Delta L(t)}{\Delta t} \triangleq \mathbb{E}\{L(t + \Delta t) - L(t) | D(t)\}. \quad (22)$$

We use the expectation to address the randomness of the intermediate data generated by the execution of jobs and the data placement actions. As to solve the problem defined in Formula (20) requires the global information the **FedCube** system, which is hard to predict or gather, we transform the problem defined in Formula (20) to the following objective function, which is a greedy conversion with limited local information:

$$\min \left( \frac{\Delta L(t)}{\Delta t} + \omega \cdot \mathbb{E}\{Cost | D(t)\} \right) \\ \text{s.t.} \begin{cases} T(job_k, Plan[t]) < TDL_k, \forall k \in [1, K] \\ M(job_k, Plan[t]) < MB_k, \forall k \in [1, K] \\ p_{i,j}[t] \in [0, 1], \end{cases} \quad (23)$$

where  $T(job_k, Plan[t])$  and  $M(job_k, Plan[t])$  are defined in Formulas (5) and (10) respectively,  $TDL_k$  represents the hard execution time deadline,  $MB_k$  represents the hard monetary cost Budget, and the parameter  $\omega \geq 0$  represents the importance of the expected total cost compared with the stability of the system.

**Theorem 1.** *The objective function has the following upper bound when  $\Delta t = 1$ :*

$$\begin{aligned} & \Delta L(t) + \omega \mathbb{E}\{Cost(Jobs, Plan[t]) | D(t)\} \\ & \leq L + \omega \sum_{k=1}^K C_k \\ & + \mathbb{E} \left\{ \sum_{j=1}^N S_j(t) r_j(t) | D(t) \right\} - \mathbb{E} \left\{ \sum_{k=1}^K J_k(t) G_k[t] | D(t) \right\} \\ & + \mathbb{E} \left\{ \sum_{j=1}^N \sum_{k=1}^K \sum_{i \in data_k} (J_k(t) - S_j(t) + \omega C'_{i,j}) p_{i,j}[t] | D(t) \right\}, \quad (24) \end{aligned}$$

with  $L$  defined in Formula (28),  $C$  defined in Formula (30), and  $C'$  defined in Formula (31).

**Proof.** First, we focus on the data stored in the job data queue with the assumption that  $\sum_{i=1}^M p_{i,j}[t] \leq d_{\max}$  and  $r_j(t) \leq r_{\max}$

$$\begin{aligned} & S_j^2(t+1) - S_j^2(t) \\ & = \left( \max[S_j(t) - r_j(t), 0] + \sum_{i=1}^M p_{i,j}[t] \right)^2 - S_j^2(t) \\ & \leq \left( \sum_{i=1}^M p_{i,j}[t] \right)^2 + (r_j(t))^2 - 2S_j(t) \left( r_j(t) - \sum_{i=1}^M p_{i,j}[t] \right) \\ & \leq (d_{\max})^2 + (r_{\max})^2 - 2S_j(t) \left( r_j(t) - \sum_{i=1}^M p_{i,j}[t] \right). \quad (25) \end{aligned}$$

Then, we have the similar results for the data storage spaces with the assumption that  $\sum_{i \in data_k, j=1}^N p_{i,j}[t] \leq data_{\max}$ , where  $data_{\max}$  represents the maximum number of data sets for any job, and  $G_k[t] \leq G_k^{\max}$

$$\begin{aligned} & J_k^2(t+1) - J_k^2(t) \\ & = \left( \max[J_k(t) - \sum_{i \in data_k, j=1}^N p_{i,j}[t], 0] + G_k[t] \right)^2 - J_k^2(t) \\ & \leq (G_k[t])^2 + \left( \sum_{i \in data_k, j=1}^N p_{i,j}[t] \right)^2 \\ & \quad - 2 \cdot J_k(t) \left( \sum_{i \in data_k, j=1}^N p_{i,j}[t] - G_k[t] \right) \\ & \leq (G_k^{\max})^2 + (data_{\max})^2 \\ & \quad - 2 \cdot J_k(t) \left( \sum_{i \in data_k, j=1}^N p_{i,j}[t] - G_k[t] \right) \quad (26) \end{aligned}$$

With Formulas (25) and (26), we have

$$\begin{aligned} & \Delta \{L(t) | D(t)\} \\ & \leq L + \mathbb{E} \left\{ \sum_{j=1}^N S_j(t) (r_j(t) - \sum_{i=1}^M p_{i,j}[t]) | D(t) \right\} \\ & \quad + \mathbb{E} \left\{ \sum_{k=1}^K J_k(t) \left( \sum_{i \in data_k, j=1}^N p_{i,j}[t] - G_k[t] \right) | D(t) \right\}, \quad (27) \end{aligned}$$

$$L = \frac{N}{2} \cdot [(d_{\max})^2 + (r_{\max})^2] + \frac{K}{2} \cdot [(G_k^{\max})^2 + (data_{\max})^2]. \quad (28)$$

The cost model presented in Section 4.2 can be rewritten as

$$\begin{aligned} & \mathbb{E}\{cost(Jobs, Plan[t]) | D(t)\} \\ & = \sum_{k=1}^K C_k + \mathbb{E} \left\{ \sum_{j=1}^N \sum_{k=1}^K \sum_{i \in data_k} C'_{i,j,k} \cdot p_{i,j}[t] | D(t) \right\}, \quad (29) \end{aligned}$$

$$\begin{aligned} C_k = & \left( \frac{\omega_t \cdot n_k \cdot AIT}{DT_k} + \left( \frac{\omega_t}{DT_k} + \frac{\omega_m \cdot VMP(job_k) \cdot n_k}{DM_k} \right) \right. \\ & \left. \cdot \frac{(\frac{\alpha_k}{n} + (1 + \alpha_k)) * WL(job_k)}{CSP} \right) \cdot f(job_k), \quad (30) \end{aligned}$$

$$\begin{aligned} C'_{i,j,k} = & \left( \frac{\omega_t}{speed_j \cdot DT_k} + \frac{\omega_m \cdot VMP(job_k) \cdot n_k}{speed_j \cdot DM_k} + \frac{\omega_m \cdot RP_j}{DM_k} \right. \\ & \left. + \frac{\omega_m \cdot WL(job_k) \cdot SP_j}{\sum_{l=1}^K (WL(job_k) \cdot f(job_k)) \cdot DM_k} \right) \\ & \cdot size(d_i) \cdot f(job_k) \quad (31) \end{aligned}$$

Finally, we can take the expectation and add the total cost, i.e.,  $cost(Plan[t])$  to both sides of Formula (27) and hence Theorem 1 is proven.  $\square$

In order to solve the problem defined in Formula (20), we minimize the upper bound of Theorem 24. As the status of Time slot  $t$  can be observed in the system, we only need to minimize the following element:

$$\begin{aligned} & \mathbb{E} \left\{ \sum_{j=1}^N \sum_{k=1}^K \sum_{i \in \text{data}_k} (J_k(t) - S_j(t) + \omega C'_{i,j,k}) p_{i,j}[t] | D(t) \right\} \\ &= \mathbb{E} \left\{ \sum_{j=1}^N \sum_{i=1}^M C'_{i,j} p_{i,j}[t] | D(t) \right\}, \end{aligned} \quad (32)$$

with  $C'_{i,j}$  defined as

$$C'_{i,j} = \sum_{k \in \text{Jobs}_i} (J_k(t) + \omega C'_{i,j,k}) - S_j(t), \quad (33)$$

where  $\text{Jobs}_i$  represents the set of jobs that process Data set  $d_i$ .

---

### Algorithm 1. Lyapunov-Based Near-Optimal Data Placement

---

**Input:**  $D$ : A set of data sets;  
 $T$ : Maximum number of iterations;  
 $T'$ : Maximum number of iterations for generating data placement plans;  
 $\text{Plan}[t]$ : data placement plan in Time slot  $t$ .  
**Output:**  $\text{Plan}[t+1]$ : data placement plan in Time slot  $t+1$ .  
1:  $D \leftarrow \text{sort}(D)$   
2: **for**  $t \in T$  **do**  
3:   **while** not all data sets  $\in D$  are placed and  $\text{iter} < T'$  **do**  
4:      $\text{Plan}^*[t] \leftarrow \text{NearOptimalDataPlanning}(\text{Plan}[t])$   
5:     **for** each Data set  $d_i$  in  $D$  **do**  
6:       **for**  $j \in N$  **do**  
7:           $p_{i,j}^*[t] \leftarrow \text{getPlan}(\text{Plan}^*[t], i, j)$   
8:          **if**  $C'_{i,j} \leq 0$  and  $p_{i,j}^*[t] \neq 0$  **then**  
9:            Set  $p_{i,j}[t+1] = p_{i,j}^*[t]$   
10:          **else**  
11:            Set  $p_{i,j}[t+1] = 0$   
12:          **end if**  
13:       **end for**  
14:     **end for**  
15:     **end while**  
16:      $\text{iter} \leftarrow \text{iter} + 1$   
17: **end for**  
18: Update  $J_k(t)$  and  $S_j(t)$

---

We design a Lyapunov-based approach to minimize Formula (32), as shown in Algorithm 1. First, we sort the data sets based on  $C'_{i,j}$  in descent order in order to minimize the cost of the data set corresponding to high costs first (Line 1). Then, for each data set, we use Algorithm 2 to find an optimal data placement plan (Line 4). For each combination of  $\{i, j\}$  (Line 5), if  $C'_{i,j} \leq 0$  (Line 8), we will update  $p_{i,j}[t+1] = p_{i,j}^*[t]$  (Line 9), otherwise, we will set  $p_{i,j}[t+1] = 0$  (Line 11). Please note that the data set is placed with a data placement plan that meets reasonable constraints based on Algorithm 2 when  $C'_{i,j} \leq 0$ . Otherwise, the placement plan remains idle and will be set with a proper data placement plan in later time intervals. When the data placement plan of a data set is idle, the execution of related jobs is postponed until the placement plan is set in order to meet the constraints. Please note that when the data in the system exceed the capacity of the system or the given constraints are not reasonable, the algorithm may not generate a proper data placement plan that meets all the constraints.

Authorized licensed use limited to: University of Nantes. Downloaded on July 06, 2023 at 12:20:23 UTC from IEEE Xplore. Restrictions apply.

## 5.2 Near-Optimal Data Placement Algorithm

Based on the multi-objective cost model, we propose a greedy algorithm to generate a near-optimal data placement plan while reducing the expected total cost to execute a set of jobs on the FedCube platform as shown in Algorithm 2. In the algorithm, for each Data set  $d$ , we first calculate the total cost based on the cost model (Line 2). Then, we generate a near-optimal data placement plan by replacing Data set  $d$  while keeping the other data sets based on Algorithm 3 (Line 3). Afterward, if the new data placement plan can reduce the total cost according to the cost model, we update the data placement plan if the new data placement plan corresponds to a smaller total cost (Lines 4 - 5).

---

### Algorithm 2. Near-Optimal Data Planning

---

**Input:**  $D$ : A set of data sets;  
 $\text{Plan}$ : data placement plan in Time slot  $t$ .  
**Output:**  $\text{Plan}^*$ : The near-optimal data placement plan of each data  $d$  in data set  $D$  with the minimum cost.  
1: **for** each Data  $d$  in  $D$  **do**  
2:    $\text{cost}_{\text{before}} \leftarrow \text{calculateCost}(\text{Plan})$   $\triangleright$  According to Formula (1)  
3:    $\text{Plan}' \leftarrow \text{getNearOptimalPlacement}(d, \text{Plan})$   
4:   **if**  $\text{Cost}(\text{Plan}') < \text{Cost}(\text{Plan})$  **then**  
5:      $\text{Plan}^* \leftarrow \text{Plan}'$   
6:   **end if**  
7: **end for**

---



---

### Algorithm 3. Near-Optimal Data Placement

---

**Input:**  $d$ : A data set;  
 $\text{Jobs}$ : A set of jobs that process Data set  $d$ ;  
 $\text{StorageTypeList}$ : The list of storage types;  
 $\text{Plan}$ : a data placement plan.  
**Output:**  $\text{Plan}^*$ : The near-optimal data placement plan of Data set  $d$ .  
1:  $\text{Plan}^* \leftarrow \text{Plan}$   
2:  $j^* = \text{getOptimalType}(\text{Plan}, d, \text{StorageTypeList})$   
3:  $\text{TypesForTimeConstraints} \leftarrow \text{getTypesForTimeConstraint}(\text{Plan}, d, \text{Jobs})$   
4:  $\text{TypesForMonetaryConstraints} \leftarrow \text{getTypesForMonetaryConstraint}(\text{Plan}, d, \text{jobs})$   
5:  $\text{AvailableTypes} \leftarrow \text{TypesForTimeConstraints} \cap \text{TypesForMonetaryConstraints}$   
6: **if**  $j^* \in \text{AvailableTypes}$  **then**  
7:   **For**  $j \in [1, N]$  and  $p_{i,j} \in \text{Plan}^*$ , set  $p_{i,j} = \begin{cases} 1, & j = j^* \\ 0, & j \neq j^* \end{cases}$   
8: **else**  
9:    $\text{Plan}^* \leftarrow \text{dataPlacementWithPartitioning}(d, \text{Plan}, \text{Jobs}, \text{TypesForTimeConstraints}, \text{TypesForMonetaryConstraints})$   
10: **TypesForMonetaryConstraints,**  
11: **endif**

---

Algorithm 3 replaces Data set  $d$  in order to reduce the total cost. First, we choose an optimal data storage type  $j^*$  based on the data placement plan by trying each data storage type in  $\text{storageTypeList}$  (Line 2). Then, we choose a set of possible storage type candidates that meet both the hard time deadline constraint and hard monetary budget constraint (Lines 3 - 4). If the chosen data storage type  $j^*$  is within the set of storage type candidates, we will update the data storage placement. If not, we will exploit Algorithm 4 to place the data set with data partitioning.

Algorithm 4 generates a near-optimal data placement plan with the consideration of data partitioning while meeting the two constraints, i.e., the hard time deadline constraint and the hard monetary budget constraint. First, if any of the set of available data storage type candidates for the hard time deadline constraint or hard monetary budget constraint is an empty set, we consider that the two constraints cannot be met (Lines 2 and 3). If not, first, we choose an optimal type ( $j_1$  for the time constraint and  $j_2$  for the monetary constraint) within the set of candidates for each constraint by trying each storage type (Lines 5 and 6). We define a possible area as the range of parts of the data set to be placed at Type  $j_1$  while meeting both the two constraints. We can calculate the possible area for each job and the intersection of the area for all the related jobs (Lines 7 - 10). Given a related job  $job_k$  of a data set and two data storage types ( $j_1, j_2$ ), we can calculate the possible area based on Formulas (1) - (13), (14) and (15), and the calculated area is:  $\max\{0, a\} \leq p_{i,j_1} \leq \min\{b, 1\}$  when  $c > 0$ , or  $\max\{a, b\} \leq p_{i,j_1} \leq 1$  when  $c < 0$ , with

$$a = \frac{TDL_k - ET(job_k) - n_k \cdot AIT}{size(d)} \cdot \frac{speed_{j_1} \cdot speed_{j_2}}{speed_{j_2} - speed_{j_1}} - \frac{speed_{j_1}}{speed_{j_2} - speed_{j_1}},$$

$$b = \frac{MB_k}{c \cdot size(d)} - \frac{VMP(job_k) \cdot n_k \cdot ET(job_k)}{c \cdot size(d)} - \frac{VMP(job_k) \cdot n_k}{c \cdot speed_{j_2}} - \frac{SP_{j_2}}{c \cdot size(d)} - \frac{RP_{j_2}}{c \cdot size(d)},$$

$$c = VMP(job_k) \cdot n_k \cdot \left( \frac{1}{speed_{j_1}} - \frac{1}{speed_{j_2}} \right) + d \cdot (SP_{j_1} - SP_{j_2}) + (RP_{j_1} - RP_{j_2}),$$

$$d = \frac{WL(job)}{\sum_{l=1}^K (WL(j_l) \cdot f(j_l))},$$

where  $AIT$  represents the average initialization time,  $ET$  represents the execution time, which can be calculated based on Formula (7),  $SP$  represents the storage price,  $RP$  represents the read price. Finally, if the final possible area is an empty set, we consider that the two constraints cannot be met (Lines 11 and 12). If not, we calculate the optimal data partitioning by choosing a boundary of the area that corresponds to a smaller total cost and update the data placement plan (Lines 14 and 16).

### 5.3 Algorithm Analysis

Let us assume that we have  $M$  input data,  $N$  data storage types, and each input data is related to  $K$  jobs on average. Then, the search space for the problem we address is  $\mathcal{O}(N^M)$ , which is the complexity of the brute-force method. The complexity of ActGreedy algorithm [17] is  $\mathcal{O}(M * K * N)$ . Then, the complexity of LNODP is  $\mathcal{O}(T * M * K * N)$  (when there is no need to execute Algorithm 4) or  $\mathcal{O}(T * M * K^2 * N)$  (when Algorithm 4 is executed for each job), which is much smaller than  $\mathcal{O}(N^M)$ .

when  $N^{M-1} > M * K^2 * T$  (this is a general case). Please note that we do not reduce the complexity of the problem but reduce the complexity of the solution. The complexity of Economic and Performance (see details in Section 6) is  $\mathcal{O}(M * M)$ . Although the complexity of LNODP is slightly bigger than that of ActGreedy, Economic, or Performance, it can generate near-optimal data placement plans while satisfying hard constraints.

---

#### Algorithm 4. Data Placement With Partitioning

---

**Input:**  $d$ : A set of data;

$Jobs$ : A set of job that process Data set  $d$ ;

$StorageTypeList$ : The list of storage types;

$TypesForTimeConstraints$ : A set of storage types that only meet the hard execution time constraint;

$TypesForMonetaryConstraints$ : A set of storage types that only meet the hard monetary budget constraint;

$Plan$ : a data placement plan.

**Output:**  $Plan^*$ : The near-optimal data placement plan of each data  $d$ ;

**Feasibility:** If there is a data placement plan that meets the two constraints

```

1:  $Plan^* \leftarrow Plan[d]$ 
2: if  $TypesForTimeConstraints = \emptyset$  or  $TypesForMonetaryConstraints = \emptyset$  then
3:    $Feasibility = False$ 
4: else
5:    $j_1 \leftarrow \text{getOptimalTypeForTimeConstraint}(Plan, d, Jobs, TypesForTimeConstraints)$ 
6:    $j_2 \leftarrow \text{getOptimalTypeForMonetaryConstraint}(Plan, d, Jobs, TypesForMonetaryConstraints)$ 
7:    $possibleArea \leftarrow [0, 1]$ 
8:   for  $j \in [1, N]$  do
9:      $possibleArea \leftarrow possibleArea \cap \text{getArea}(Plan, d, j_1, j_2, Jobs)$ 
10:  end for
11:  if  $possibleArea = \emptyset$  then
12:     $Feasibility = False$ 
13:  else
14:     $p \leftarrow \text{getOptimalPart}(d, plan, Jobs, possibleArea)$ 
15:    For  $j \in [1, N]$  and  $p_{i,j} \in Plan^*$ , set
16:    $p_{i,j} = \begin{cases} p, j = j_1, \\ 1 - p, j = j_2, \\ 0, else \end{cases}$ 
17:  end if
18: end if

```

---

LNODP can generate a near-optimal result while satisfying the hard constraints in most cases. However, there are two cases where LNODP cannot generate a data placement plan to satisfy hard constraints for a job. First, when there is no data storage type to store all the input data of a job while satisfying both the hard time deadline and the hard monetary budget. Second, when there is no combination of two storage types that can satisfy both the hard time deadline and the hard monetary budget. In these two cases, the user should reset the hard constraints of the job in order to use LNODP to generate data placement plans.

In order to analyze the worst case guarantee of the LNODP algorithm, we focus on the case when the data is scheduled according to the near-optimal data plan as explained in Line 9

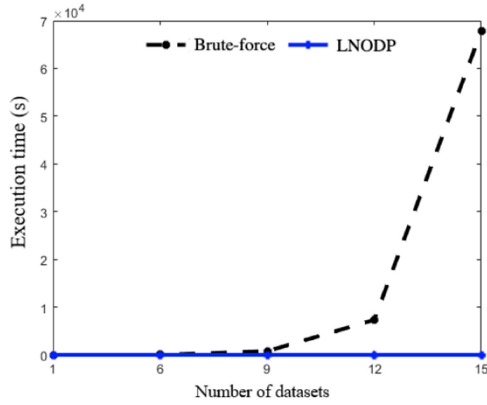


Fig. 5. Execution time of greedy and brute-force.

of Algorithm 1. The case explained in Line 11 of Algorithm 1 is ignored as the data is not scheduled in this case. The problem addressed in Algorithm 2 is a scheduling problem when there is an optimal solution according to Algorithms 3 and 4. When the data can be scheduled without being partitioned, the solution is equal to the solution generated by a greedy algorithm. As the cost function of the combination problem is monotone, Algorithm 4 can generate an optimal combination of the chosen storage types by Algorithm 3. Thus, when the data needs to be partitioned while scheduling, the solution is also equal to the solution generated by a greedy algorithm. As the scheduling problem while minimizing a cost function is a typical submodular problem as explained in [47], the worst case guarantee of the LNODP algorithm becomes the worst case guarantee of a greedy algorithm for a submodular, which is  $\frac{e-1}{e} * f^*$ , where  $e$  is the base of the natural logarithm and  $f^*$  represents the optimal solution [48].

## 6 EXPERIMENTATION

In this section, we first present the simulation to compare the execution time of our proposed Lyapunov-based Near-Optimal Data Placement (LNODP) algorithm and the brute-force method. We consider four storage types, i.e., Standard, Low frequency, Cold, and Archive, in our proposed algorithm. These four storage types are provided by the storage service on the Baidu cloud. Then, we compare the total cost of four storage methods: LNODP, brute-force, Performance [20], and Economic [21]. The brute-force method is to search the minimum cost in the entire searching space, which means that the result of brute-force is the optimal solution. The Performance method [20] uses the storage type that corresponds to the highest data transfer speed. Economic [21] uses the storage type that corresponds to the smallest price to store data. In addition, we compare our algorithm with a simple adapted greedy algorithm, i.e., ActGreedy [17], to show that our algorithm can address multiple hard constraints while ActGreedy only reduces the total cost without considering the hard constraints. Then, we present the comparison of the total cost among the four storage methods using a widely used data processing benchmark, i.e., Wordcount on Hadoop [49], and a real-life data processing program for the correlation analysis of COVID-19 [11] (COVID-19-Correlation), which is selected from recent work related to COVID-19 [11], [50], [51]. In the

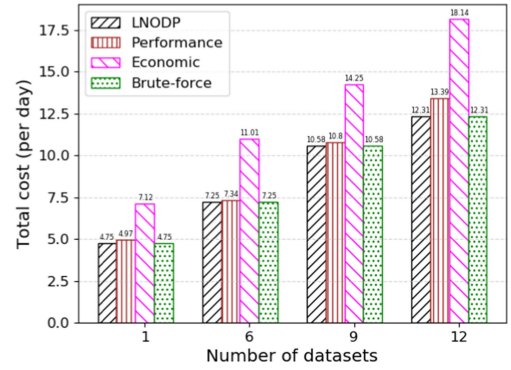


Fig. 6. Comparison among four methods.

experimentation, we consider five execution frequencies (daily, semimonthly, monthly, quarterly, and yearly) for Wordcount and COVID-19-Correlation.

### 6.1 Simulation

In this section, we compare our proposed algorithm with the brute-force method in terms of the execution time and the total cost. We take 15 data sets with the average size being 5.5 GB as the input data of jobs. We execute fifteen jobs to process the input data. Each job is associated with different data sets, including Wordcount, Grep, etc. Each job is with different frequencies and different settings such as  $DT$ ,  $w_t$ . The data sets include DBLP XML files [52] and some data sets from Baidu. The DBLP XML file contains the metadata, e.g., the name of authors, publishers, of computer-based English articles. The comparison experiment results are shown in Fig. 5.

Fig. 5 shows the result of the execution time of different methods. In order to generate a data placement plan for six data sets with fifteen jobs, the execution time of the greedy algorithm is shorter than 0.0001s, while that of LNODP is 0.08s. When the number of the data sets augments, the execution time of the brute-force method increases exponentially. When the number of data sets becomes 15, the execution time of the brute-force method is 67839s, while that of LNODP remains within 0.0001s.

Fig. 6 presents the comparison among four methods: LNODP, brute-force, Performance, and Economic. LNODP corresponds to the same total cost as that of the brute-force method, which is up to 8.2% and 30.6% smaller than that of Performance and Economic, respectively. The simulation experiment shows that the result of our proposed algorithm is as same as the brute-force method, which means the result of our proposed algorithm is the optimal solution in these situations.

### 6.2 Wordcount

Hadoop [53] is a framework for parallel big data processing on a cluster of commodity servers. Hadoop contains two components, i.e., HDFS [54] and MapReduce. HDFS is a distributed file system with a master-slave architecture. MapReduce is a programming model and implementation for parallel data processing in a distributed environment. MapReduce contains two phases, i.e., Map and Reduce. In the Map phase, the input data is processed, and key-value pairs

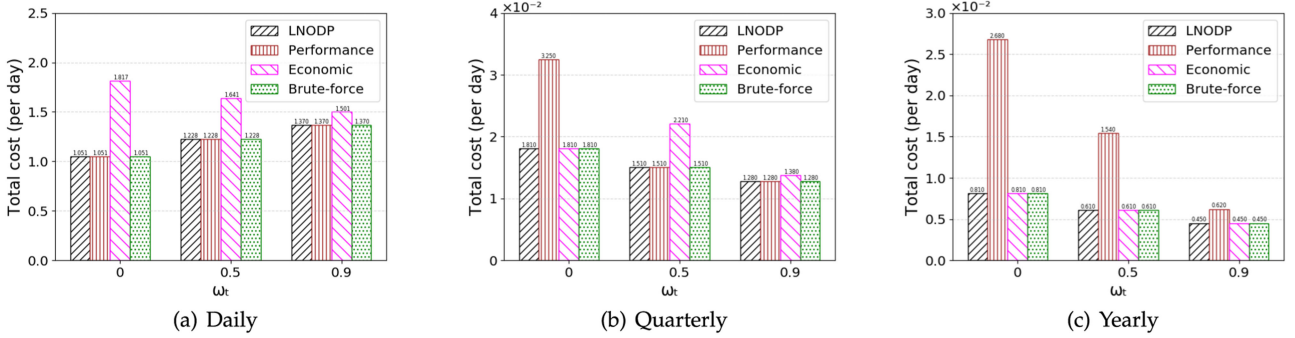


Fig. 7. Total cost of Wordcount.

are generated. In the reduce phase, the key-value pairs of the same Key are processed.

Wordcount is a widely used benchmark, which counts the frequency of each word in the input files. Wordcount contains two steps, i.e., Map and Reduce. In the Map step,  $\langle word, 1 \rangle$  is generated for each work in the input data. Then, the number of  $\langle word, 1 \rangle$  is counted for each work in the Reduce step. Finally, the frequency of each word is calculated and stored in HDFS.

We deploy Hadoop on three computing nodes based on the platform. Each node is a VM with one CPU core and 4 GB RAM. We use DBLP 2019 XML files of 6.04 GB as the input data. We set  $DT$  as 1200 seconds and  $DM$  as 1 dollar.

First, we set the hard time deadline as 2,000 seconds and 10 dollars. Fig. 7 shows that our proposed algorithm, i.e., LNOPD, significantly outperforms the baseline approach. When the frequency is daily, the total cost corresponding to different approaches is shown in Fig. 7a. Compared with Economic, LNOPD can reduce the total cost by 42.2%, 25.2%, and 8.7% when  $\omega_t$  is 0, 0.5, and 0.9 respectively. When the frequency is quarterly, LNOPD can reduce the total cost by 44.3% compared with Performance when the  $\omega_t$  is 0 as Fig. 7b shows. LNOPD can generate an optimal storage plan, which significantly outperforms (the total cost is 31.7% and 7.2% smaller) Economic when  $\omega_t$  is 0.5 and 0.9, respectively. Fig. 7c presents the efficiency of our proposed algorithm when the frequency is yearly. Compared with Performance, our algorithm can reduce the total cost by 69.8%, 60.4% and 27.4% when  $\omega_t$  is 0, 0.5 and 0.9, respectively.

Fig. 7 presents that our algorithm can reduce the total cost by up to 69.8% compared with Performance and up to 42.2% compared with Economic. As the execution frequency of the job decreases, the advantage of our algorithm becomes significant. The comparison of Figs. 7a, 7b and 7c indicates that as the importance of time cost becomes bigger, i.e.,  $\omega_t$ , increases, the advantage of our proposed algorithm becomes significant as well. This experiment also shows that the result of our algorithm can generate the optimal solution as the brute-force method.

Table 3 presents the execution with a strict hard execution time constraint and a hard monetary budget constraint, i.e., 1,420 seconds and 6.5 dollars. The existing methods, e.g., ActGreedy, Performance, Economic, cannot meet both the two constraints, while LNOPD can place the data with data partitioning while satisfying the two hard constraints with small total cost. In addition, we find that the weight of objectives only impacts the total cost, which has no impact on the satisfaction of the constraints.

Authorized licensed use limited to: University of Nantes. Downloaded on July 06, 2023 at 12:20:23 UTC from IEEE Xplore. Restrictions apply.

In addition, the average execution time of LNOPD, Performance, Economic, and Brute-force are  $2.79 \times 10^{-4}$ ,  $4.26 \times 10^{-5}$ ,  $4.14 \times 10^{-5}$ ,  $2.98 \times 10^{-4}$ , respectively. While LNOPD corresponds can generate good data placement plans, the execution time remains quite acceptable.

### 6.3 COVID-19

Since the coronavirus disease (COVID-19) has become a global emergency, we reproduced the data processing program for the correlation among COVID-19-related search activities, human mobility, and the number of confirmed cases in Mainland China presented in [11]. The data involved in [11] includes the number of confirmed cases in each city ( $dataset_c$ ), the volume of COVID-19-related search activities in each city ( $dataset_s$ ), inflows and outflows for each city ( $dataset_m$ ) and the population in each city ( $dataset_p$ ).  $dataset_m$  is the inflow and outflow data of inter-city population with the transitions of the inter-city mobility categorized by the origin and destination pairs.  $dataset_s$  includes the keywords and phrases related to the epidemic from January to March. The total amount of these data sets is 1.134 GB.

The data processing for the COVID-19-related correlation analysis consists of the following three steps. First, the data is selected using a filter operation. Then, a join operator is used to generate the features for each city, i.e., the number of confirmed cases, the inflows, the outflows, the search volumes, the population. Afterward, the correlation between any two features is calculated for each city. The experimental results are shown in Fig. 8. We set  $DT$  as 600 seconds and  $DM$  as 0.5 dollars.

TABLE 3  
Results for Hard Execution Time Constraint and Hard Monetary Budget Constraint

	Constraints		Cost	$\omega_t$
	Time	Monetary		
LNODP	Satisfied (1420.0)	Satisfied (6.5)	0.018	0
ActGreedy	<b>Broken (1465.8)</b>	Satisfied (2.9)	0.0081	
Performance	Satisfied (1405.4)	<b>Broken (9.7)</b>	0.027	
Economic	<b>Broken (1465.8)</b>	Satisfied (2.9)	0.0081	
LNODP	Satisfied (1420.0)	Satisfied (6.5)	0.0053	0.9
ActGreedy	<b>Broken (1465.8)</b>	Satisfied (2.9)	0.0045	
Performance	Satisfied (1405.4)	<b>Broken (9.7)</b>	0.0062	
Economic	<b>Broken (1465.8)</b>	Satisfied (2.9)	0.0045	

Frequency: yearly. Hard time deadline: 1420; hard monetary budget: 6.5. The time unit is second and the monetary unit is yuan.



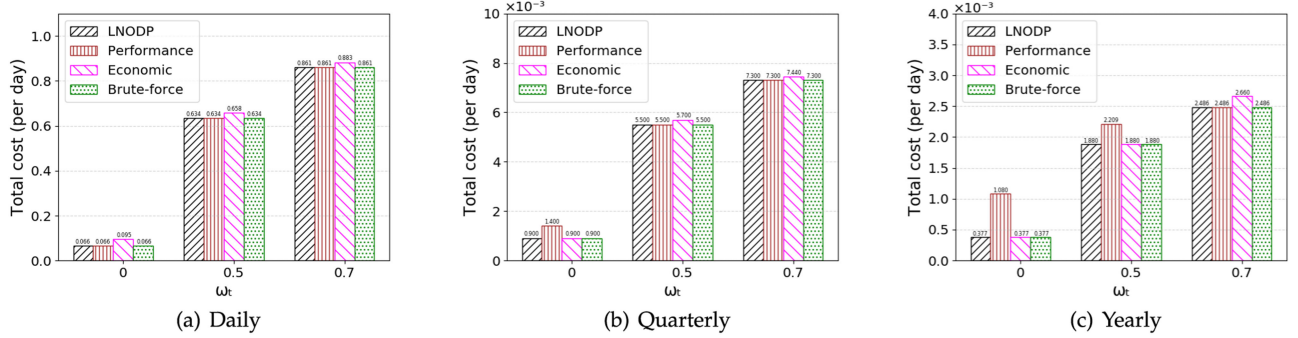


Fig. 8. Total cost of COVID-19-Correlation.

First, we set the hard execution time constraint as 800 seconds and the hard monetary budget constraint as 2 dollars. Fig. 8 shows that our proposed algorithm, i.e., LNOPD, significantly outperforms Performance (up to 65.1%) when the size of the input data of the job is smaller than that of Wordcount. When the frequency is daily, the total costs of different approaches are shown in Fig. 8a. When  $\omega_t$  is 0 and  $\omega_m$  is 1, our algorithm can reduce the total cost by 30.5% compared with Economic. When  $\omega_t$  increases to 0.5, our algorithm can reduce the total cost by 3.6% compared with Economic. When the importance of time, i.e.,  $\omega_t$ , increases to 0.7, our algorithm can outperform Economic, and the total cost can be reduced by 2.5%. Fig. 8b presents the total cost of different approaches when the frequency is quarterly. When the user only considers the importance of money, our algorithm can reduce the total cost by 35.7% compared with Performance. With the increase of  $\omega_t$ , our algorithm can reduce the total cost by 3.5% and 1.9% compared with Economic when  $\omega_t$  is 0.5 and 0.7 respectively. When the frequency is yearly, the execution results are shown in Fig. 8c. The most significant result is that our algorithm can reduce the total cost by 65.1% compared with Performance when  $\omega_t$  is 0 and  $\omega_m$  is 1. When  $\omega_t$  is 0.5 and 0.7, our algorithm can reduce the total cost by 14.9% and 6.5% compared with Performance and Economic, respectively.

From Fig. 8, we find that our proposed algorithm, i.e., LNOPD, significantly outperforms the Performance method (up to 65.1%) and the Economic method (up to 30.5%), when the frequency of the job execution is high and when the size of the input data of the job is big.

TABLE 4

Results for Hard Execution Time Constraint and Hard Monetary Budget Constraint

	Constraints		Cost	$\omega_t$
	Time	Monetary		
LNODP	Satisfied (722.0)	Satisfied (1.8)	0.0050	0
ActGreedy	<b>Broken (732.1)</b>	Satisfied (0.7)	0.0019	
Performance	Satisfied (720.8)	<b>Broken (1.95)</b>	0.00054	
Economic	<b>Broken (732.1)</b>	Satisfied (0.7)	0.0019	
LNODP	Satisfied (722.0)	Satisfied (1.8)	0.0038	0.7
ActGreedy	<b>Broken (732.1)</b>	Satisfied (0.7)	0.0029	
Performance	Satisfied (720.8)	<b>Broken (2.0)</b>	0.0040	
Economic	<b>Broken (732.1)</b>	Satisfied (0.7)	0.0029	

Frequency: yearly. Hard time deadline: 722; hard monetary budget: 1.9. The time unit is second and the monetary unit is yuan.

Authorized licensed use limited to: University of Nantes. Downloaded on July 06, 2023 at 12:20:23 UTC from IEEE Xplore. Restrictions apply.

Table 4 presents the execution with a strict hard execution time constraint and a hard monetary budget constraint, i.e., 722 seconds and 1.9 dollars. The existing methods, e.g., ActGreedy, Performance, Economic, cannot meet both the two constraints. However, LNOPD can place the data with data partitioning while satisfying the two hard constraints with a small total cost. We find that the weight of objectives only impacts the total cost while having no impact on the satisfaction of the constraints.

In addition, the average execution time of LNOPD, Performance, Economic, and Brute-force are  $2.01 \times 10^{-4}$ ,  $2.01 \times 10^{-5}$ ,  $2.01 \times 10^{-5}$ ,  $2.04 \times 10^{-4}$ , respectively. The execution time of LNOPD is quite acceptable.

## 7 CONCLUSION

When organizations outsource their data onto the cloud, it is critical to choose a proper data placement strategy to reduce its expected total cost. In this paper, we proposed a solution to enable data processing on the cloud with the data from different organizations. The approach consists of three parts: a data federation platform with secure data sharing and secure data computing, a multi-objective cost model, and a Lyapunov-based near-optimal data placement algorithm. The cost model consists of monetary cost and execution time. The Lyapunov-based near-optimal algorithm delivers a solution to the problem. We carried out extensive experiments to validate our proposed approach. The experimental results indicate that our proposed algorithm outperforms the baseline approaches up to 69.8% and that our algorithm can generate the same optimal solution as the brute-force method within a short execution time.

## ACKNOWLEDGMENTS

Ji Liu and Lei Mo are equal contribution.

## REFERENCES

- [1] I. Greif and S. K. Sarin, "Data sharing in group work," *ACM Trans. Office Inf. Syst.*, vol. 5, no. 2, pp. 187–211, 1987.
- [2] J. Liu *et al.*, "Efficient scheduling of scientific workflows using hot metadata in a multisite cloud," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 10, pp. 1940–1953, Oct. 2019.
- [3] P. Voigt and A. von dem Bussche, *The EU General Data Protection Regulation (GDPR): A Practical Guide*, 1st ed. Berlin, Germany: Springer Publishing Company, Incorporated, 2017.
- [4] E. Valentijn *et al.*, "Target and (astro-)wise technologies data federations and its applications," in *Proc. Int. Astronomical Union*, 2016, pp. 333–340.
- [5] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "Parallelization of Scientific Workflows in the Cloud," INRIA, Research Rep. RR-8565, 2014.

- [6] A. Fox *et al.*, "Above the clouds: A Berkeley view of cloud computing," Dept. Elect. Eng. and Comput. Sci., California Univ., Berkeley, Tech. Rep. UCB/ECS-2009-28, 2009.
- [7] N. Kratzke, "A brief history of cloud application architectures," *Applied Sci.*, vol. 8, 2018, Art. no. 1368.
- [8] M. M. Moghaddam, M. H. Manshaei, W. Saad, and M. Goudarzi, "On data center demand response: A cloud federation approach," *IEEE Access*, vol. 7, pp. 101 829–101 843, 2019.
- [9] X. Boyen and B. Waters, "Anonymous hierarchical identity-based encryption (without random oracles)," in *Proc. Annu. Int. Cryptol. Conf.*, 2006, pp. 290–307.
- [10] A. Guabtini, F. Charoy, and C. Godart, "Customizable isolation in transactional workflow," in *Proc. Interoperability Enterprise Softw. Appl.*, 2006, pp. 197–202.
- [11] H. Xiong *et al.*, "Understanding the collective responses of populations to the covid-19 pandemic in mainland China," *medRxiv* 2020.04.20.20068676, 2020.
- [12] W. Fang, X. Yao, X. Zhao, J. Yin, and N. Xiong, "A stochastic control approach to maximize profit on service provisioning for mobile cloudlet platforms," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 48, no. 4, pp. 522–534, Apr. 2016.
- [13] D. Y. Zhang and D. Wang, "An integrated top-down and bottom-up task allocation approach in social sensing based edge computing systems," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2019, pp. 766–774.
- [14] X. Wang, R. Jia, X. Tian, and X. Gan, "Dynamic task assignment in crowdsensing with location awareness and location diversity," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2018, pp. 2420–2428.
- [15] L. Golab, M. Hadjieleftheriou, H. Karloff, and B. Saha, "Distributed data placement to minimize communication costs via graph partitioning," in *Proc. Int. Conf. Scientific Stat. Database Manage.*, 2014, pp. 1–12.
- [16] K. Zhao, D. Yuan, Y. Xie, L. Yan, and R. Xu, "An optimized data storage strategy by computational performance and monetary cost with data importance in the cloud," in *Proc. IEEE Int. Conf. Comput. Supported Cooperative Work Des.*, 2017, pp. 433–438.
- [17] J. Liu, E. Pacitti, P. Valduriez, D. de Oliveira, and M. Mattoso, "Multi-objective scheduling of scientific workflows in multisite clouds," *Future Gener. Comput. Syst.*, vol. 63, pp. 76–95, 2016.
- [18] M. Kumar and S. C. Sharma, "Dynamic load balancing algorithm for balancing the workload among virtual machine in cloud computing," *Procedia Comput. Sci.*, vol. 115, pp. 322–329, 2017.
- [19] N. Tziritas, S. U. Khan, C. Xu, T. Loukopoulos, and S. Lalis, "On minimizing the resource consumption of cloud applications using process migrations," *J. Parallel Distrib. Comput.*, vol. 73, no. 12, pp. 1690–1704, 2013.
- [20] M. Darwich, Y. Ismail, T. Darwich, and M. A. Bayoumi, "Cost-efficient storage for on-demand video streaming on cloud," in *Proc. IEEE 6th World Forum on Internet of Things 2020*, pp. 1–4.
- [21] R. Black, A. Donnelly, D. Harper, A. Ogun, and A. I. T. Rowstron, "Feeding the pelican: Using archival hard drives for cold storage racks," in *Proc. USENIX Workshop Hot Topics Storage File Syst., Hot-Storage*, N. Agrawal and S. H. Noh, Eds. USENIX Assoc., 2016, pp. 21–25.
- [22] S. Blagodurov, A. Fedorova, E. Vinnik, T. Dwyer, and F. Hermenier, "Multi-objective job placement in clusters," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2015, pp. 1–12.
- [23] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural Multidisciplinary Optim.*, vol. 26, no. 6, pp. 369–395, 2004.
- [24] L. Zadeh, "Optimality and non-scalar-valued performance criteria," *IEEE Trans. Automat. Control*, vol. 8, no. 1, pp. 59–60, Jan. 1963.
- [25] M. Farsi, M. Ali, R. A. Shah, A. A. Wagan, and R. Kharabsheh, "Cloud computing and data security threats taxonomy: A review," *J. Intell. Fuzzy Syst.*, vol. 38, no. 3, pp. 2517–2527, 2020.
- [26] G. Anthes, "Security in the cloud," *Commun. ACM*, vol. 53, no. 11, pp. 16–18, 2010.
- [27] Z. Feng *et al.*, "Securegbm: Secure multi-party gradient boosting," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, 2019, pp. 1312–1321.
- [28] Y. Singh, F. Kandah, and W. Zhang, "A secured cost-effective multi-cloud storage in cloud computing," in *Proc. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2011, pp. 619–624.
- [29] J. Bian, H. Xiong, Y. Fu, J. Huan, and Z. Guo, "Mp2sda: Multi-party parallelized sparse discriminant learning," *ACM Trans. Knowl. Discovery Data*, vol. 14, no. 3, pp. 1–22, 2020.
- [30] J. Bian, H. Xiong, W. Cheng, W. Hu, Z. Guo, and Y. Fu, "Multi-party sparse discriminant learning," in *Proc. IEEE Int. Conf. Data Mining*, 2017, pp. 745–750.
- [31] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Stat.*, ser. Proceedings of Machine Learning Research, 2017, pp. 1273–1282.
- [32] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning, "Managing security of virtual machine images in a cloud environment," in *Proc. ACM Workshop Cloud Comput. Secur.*, 2009, pp. 91–96.
- [33] G. Heidsieck, D. de Oliveira, E. Pacitti, C. Pradal, F. Tardieu, and P. Valduriez, "Adaptive caching for data-intensive scientific workflows in the cloud," in *Proc. Int. Conf. Database Expert Syst. Appl.*, ser. Lecture Notes in Computer Science, 2019, pp. 452–466.
- [34] T. Jamil, "The rijndael algorithm," *IEEE Potentials*, vol. 23, no. 2, pp. 36–38, Apr./May 2004.
- [35] S. Bardhan and D. A. Menascé, "The anatomy of MapReduce jobs, scheduling, and performance challenges," in *Proc. Int. Comput. Measurement Group Conf.*, 2013, pp. 1–13.
- [36] D. Yoo and K. M. Sim, "A comparative review of job scheduling for MapReduce," in *Proc. IEEE Int. Conf. Cloud Comput. Intell. Syst.*, 2011, pp. 353–358.
- [37] I. Surjandari *et al.*, "The batch scheduling model for dynamic multi-item, multilevel production in an assembly job-shop with parallel machines," *Int. J. Technol.*, vol. 1, pp. 84–96, 2015.
- [38] X.-H. Sun and Y. Chen, "Reevaluating amdahl's law in the multicore era," *J. Parallel Distrib. Comput.*, vol. 70, no. 2, pp. 183–188, 2010.
- [39] R. Coutinho, L. Drummond, Y. Frota, D. de Oliveira, and K. Ocana, "Evaluating grasp-based cloud dimensioning for comparative genomics: A practical approach," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2014, pp. 371–379.
- [40] B. Polyak and P. Shcherbakov, "Lyapunov functions: An optimization theory perspective," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 7456–7461, 2017.
- [41] S. Chunduri, M. Ghaffari, M. S. Lahijani, A. Srinivasan, and S. Namila, "Parallel low discrepancy parameter sweep for public health policy," in *Proc. Int. Symp. Cluster, Cloud Grid Comput.*, 2018, pp. 291–300.
- [42] X. Ren, P. London, J. Ziani, and A. Wierman, "Datum: Managing data purchasing and data placement in a geo-distributed data market," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 893–905, Apr. 2018.
- [43] L. Mo, A. Kritikakou, and O. Sentieys, "Controllable QoS for imprecise computation tasks on dvfs multicores with time and energy constraints," *IEEE J. Emerg. Selected Topics Circuits Syst.*, vol. 8, no. 4, pp. 708–721, Dec. 2018.
- [44] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: A survey," *Surv. Oper. Res. Manage. Sci.*, vol. 17, no. 2, pp. 97–106, 2012.
- [45] J. Hartmanis, "Computers and intractability: A guide to the theory of np-completeness (michael r. Garey and david s. Johnson)," *Siam Rev.*, vol. 24, no. 1, 1982, Art. no. 90.
- [46] Y. Liu *et al.*, "A fair task assignment strategy for minimizing cost in mobile crowdsensing," in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2020, pp. 1–10.
- [47] R. Fokink, T. Lidbetter, and L. A. Végh, "On submodular search and machine scheduling," *Math. Oper. Res.*, vol. 44, no. 4, pp. 1431–1449, 2019.
- [48] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions-I," *Math. Program.*, vol. 14, no. 1, pp. 265–294, 1978.
- [49] T. White, *Hadoop: The Definitive Guide*, 4th ed. Newton, MA, USA: O'Reilly Media, Inc., 2015.
- [50] J. Liu *et al.*, "Analysis of collective response reveals that covid-19-related activities start from the end of 2019 in mainland China," *Arch. Clin. Biomed. Res.*, vol. 5, no. 3, pp. 313–343, 2021.
- [51] J. Liu *et al.*, "An investigation of containment measures against the covid-19 pandemic in mainland China," 2020, *arXiv:2007.08254*.
- [52] "Dblp: Computer science bibliography." Accessed: Feb. 21, 2021. [Online]. Available: <https://dblp.org/xml/>
- [53] "Apache hadoop." Accessed: Feb. 21, 2021. [Online]. Available: <http://hadoop.apache.org/>
- [54] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. IEEE Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–10



**Ji Liu** (Member, IEEE) received the PhD degree from the University of Montpellier in 2016, the master's degree from Telecom SudParis in 2013, and the BsC degree from Xidian University in 2011. He is currently a staff researcher with the Big Data Laboratory, Baidu Research, Beijing, China. He was a software engineer with Murex and postdoctoral research fellow with Inria and LIRMM, University of Montpellier, France. Previously, he was a PhD candidate with the Microsoft Research Inria Joint Centre, Inria Zenith team.

He has authored or coauthored several papers in top international journals and conferences, such as NeurIPS, AAAI, KDD, *Transactions on Knowledge Discovery from Data*, *IEEE Transactions on Intelligent Transportation Systems*, and *IEEE Transactions on Knowledge and Data Engineering*, and is a coauthor of the book *Data-Intensive Workflow Management For Clouds and Data-Intensive and Scalable Computing Environments* published by Morgan Claypool in 2019. His research interests include federated learning, distributed machine learning, scientific workflows, distributed and parallel databases, and distributed system.



**Lei Mo** (Member, IEEE) received the BsC degree from the College of Telecom Engineering and Information Engineering, Lanzhou University of Technology, Lanzhou, China, in 2007, and the PhD degree from College of Automation Science and Engineering, South China University of Technology, Guangzhou, China, in 2013. He is currently an associate professor with the School of Automation, Southeast University, Nanjing, China. From 2013 to 2015, he was a research fellow with the Department of Control Science and

Engineering, Zhejiang University, China. From 2015 to 2017, he was a research fellow with INRIA Nancy–Grand Est, France. From 2017 to 2019, he was a research fellow with INRIA Rennes–Bretagne Atlantique, France. His current research interests include networked estimation and control in wireless sensor and actuator networks, cyber-physical systems, task mapping, and resources allocation in embedded systems. He is currently an associate editor for the *KSII Transactions on Internet and Information Systems*, *International Journal of Ad Hoc and Ubiquitous Computing*, and *Journal of Computer and Journal of Electrical and Electronic Engineering*. He is also a guest editor of the *IEEE Access* and *Journal of Computer Networks and Communications*, and a TPC member for several international conferences.



**Sijia Yang** is currently working toward the PhD degree with the School of Cyber Space Security, Beijing University of Posts and Telecommunications. Her research interests include federated learning and big data analytics.



**Jingbo Zhou** received the BE degree from Shandong University in 2009 and the PhD degree from the National University of Singapore in 2014. He is currently a staff research scientist with Business Intelligent Lab of Baidu Research, working on machine learning problems for both scientific research and business applications, with a focus on spatial temporal data mining and AI for healthcare. He was on the program committees among top conferences in the data mining and artificial intelligence communities, including KDD, AAAI, IJCAI, ACL, CIKM, and SDM.



**Shilei Ji** received the master's degree from Tian-gong University in 2009. He is currently a staff engineer with Baidu, Beijing, China, working on data security and privacy protection for both scientific research and business innovation, with a focus on secure multiparty computation, federated learning, and confidential computing.



**Haoyi Xiong** (Senior Member, IEEE) received the PhD degree in computer science from Telecom SudParis, Université Pierre et Marie Curie, Paris, France, in 2015. From 2016 to 2018, he was an Tenure-Track assistant professor with the Department of Computer Science, Missouri University of Science and Technology, Rolla, MO, USA (formerly known as University of Missouri at Rolla). From 2015 to 2016, he was a postdoctoral research associate with the Department of Systems and Information Engineering, University of

Virginia, Charlottesville, VA, USA. He is currently a principal R&D architect and researcher with Big Data Laboratory, Baidu Research, Beijing, China. He has authored or coauthored more than 60 papers in top computer science conferences and journals., such as ICML, ICLR, UbiComp, JDD, RTSS, AAAI, IJCAI, ICDM, PerCom, *IEEE Transactions on Mobile Computing*, *IEEE Internet of Things Journal*, *IEEE Transactions on Neural Networks and Learning Systems*, *IEEE Transactions on Computers*, *ACM Transactions on Intelligent Systems and Technology*, and *ACM Transactions on Knowledge Discovery from Databases*. His current research interests include automated deep learning (AutoDL), pervasive computing, and Internet of Things. He gave keynote speeches in a series of academic and industrial activities, such as the industrial session of the 19th IEEE International Conference on Data Mining (ICDM'19), and was a poster co-chair for the 2019 IEEE International Conference on Big Data (IEEE Big Data'19). He was the recipient of Best Paper Award from IEEE UIC 2012, Outstanding PhD Thesis Runner Up Award from CNRS SAMOVAR 2015, and the Best Service Award from IEEE UIC 2017. He was one of the co-recipients of the prestigious Science and Technology Advancement Award (First Prize) from Chinese Institute of Electronics 2019, and has been granted the prestigious IEEE TCSC Award for Excellence in Scalable Computing (Early Career Researcher), 2020 by IEEE Computer Society Technical Committee on Scalable Computing (IEEE TCSC).



**Dejing Dou** (Senior Member, IEEE) received the bachelor's degree from Tsinghua University, China, in 1996 and the PhD degree from Yale University in 2004. He is currently the Head of Big Data Lab (BDL) and Business Intelligence Lab (BIL) with Baidu Research. He is also a full professor (on leave) with the Computer and Information Science Department, University of Oregon. He has authored or coauthored more than 150 research papers, some of which appear in prestigious conferences and journals like AAAI, IJCAI, ICML, NeurIPS, ICLR, KDD, ACL, EMNLP, CVPR, ICCV, CIKM, ISWC, *Transactions on Knowledge Discovery from Data*, *Journal of Intelligent Information Systems*, and JoDS, with more than 4500 Google Scholar citations. His research interests include artificial intelligence, data mining, data integration, NLP, and health informatics. His DEXA'15 paper received the Best Paper Award. His KDD'07 paper was nominated for the Best Research Paper Award. His COLING'18 paper was Area Chair Favorites (excellent). He is on the editorial boards of the *Journal on Data Semantics*, *Journal of Intelligent Information Systems*, and PLOS ONE. He is an editor-in-chief of AIMS Electronic Research Archive. He has been serving as program committee members for major international conferences and as program co-chairs for five of them. He has received over five million PI research grants from the NSF and the NIH. He is a senior member of the ACM.

He is currently a principal R&D architect and researcher with Big Data Laboratory, Baidu Research, Beijing, China. He has authored or coauthored more than 60 papers in top computer science conferences and journals., such as ICML, ICLR, UbiComp, JDD, RTSS, AAAI, IJCAI, ICDM, PerCom, *IEEE Transactions on Mobile Computing*, *IEEE Internet of Things Journal*, *IEEE Transactions on Neural Networks and Learning Systems*, *IEEE Transactions on Computers*, *ACM Transactions on Intelligent Systems and Technology*, and *ACM Transactions on Knowledge Discovery from Databases*. His current research interests include automated deep learning (AutoDL), pervasive computing, and Internet of Things. He gave keynote speeches in a series of academic and industrial activities, such as the industrial session of the 19th IEEE International Conference on Data Mining (ICDM'19), and was a poster co-chair for the 2019 IEEE International Conference on Big Data (IEEE Big Data'19). He was the recipient of Best Paper Award from IEEE UIC 2012, Outstanding PhD Thesis Runner Up Award from CNRS SAMOVAR 2015, and the Best Service Award from IEEE UIC 2017. He was one of the co-recipients of the prestigious Science and Technology Advancement Award (First Prize) from Chinese Institute of Electronics 2019, and has been granted the prestigious IEEE TCSC Award for Excellence in Scalable Computing (Early Career Researcher), 2020 by IEEE Computer Society Technical Committee on Scalable Computing (IEEE TCSC).

▷ For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/csdl](http://www.computer.org/csdl).