Quasi-optimal Data Placement for Secure Multi-tenant Data Federation on the Cloud

Qi Kang[‡†], Ji Liu[§†*], Sijia Yang[‡], Haoyi Xiong[§],
Haozhe An[§], Xingjian Li[§], Zhi Feng[§], Licheng Wang[‡], and Dejing Dou[§]
[†] Equal contribution
[‡]The State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, Beijing, China.
[§] Big Data Lab, Baidu Inc., Beijing, China

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9377953

***Abstract*—As it is difficult to directly share data among different organizations, data federation brings new opportunities to the data-related cooperation among different 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 communication cost or time cost, and do not consider the features of jobs that process the data. In this paper, we propose an approach to enable secure data processing on the Cloud with the data from different organizations. The approach consists of a data federation platform for secure data processing on the Cloud named FedCube and a greedy data placement algorithm that creates a plan to store data on the Cloud in order to achieve multiple objectives based on a cost model. The cost model is composed of two objectives, i.e., reducing both monetary cost and execution time. We present an experimental evaluation by comparing our data placement algorithm with the existing methods based on the data federation platform. The experiments show that our proposed algorithm significantly reduce the total cost (up to 69.8%).**

***Index Terms*—Data federation; Cloud computing; Secure data sharing; Data placement; Multi-objective**

I. Introduction

Data sharing is the first step to 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], [4]. Data federation [5] 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 makes 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 on the Cloud. With the pay-as-you-go model, Cloud computing (Cloud) brings convenience to the organizations to store and process large amount of data. Cloud services bring a large amount of resources at different layers. There are typically three types of services on the Cloud, i.e., Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [6]. For instance, virtual machines, in which data can be processed, are provided at the IaaS layer. A Virtual Machine (VM) is an emulator of a computer, which can be viewed as a computing node in a network [7]. Data storage services are provided at the PaaS layer. 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 payed and released as needed [8]. Dynamic provisioning enables Cloud tenants/users to construct scalable systems with reasonable cost on the Cloud [9]. 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 the access to the data [10]. Encryption is a conventional way to keep the data confidential, such as homomorphic encryption [11], identity-based encryption [12] and authentication mechanism [13]. In addition, the isolation techniques [14], which provide secure execution spaces for different jobs with specific access controls, are also used to control the accessibility to the data on 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 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[1], Microsoft Azure Cloud[2] and Baidu Cloud[3], 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 [15], [16]. However, the job execution frequency is not well exploited while constructing the data placement algorithms for the data storage on the Cloud.

In this paper, we propose a solution to enable data processing on the Cloud for the scientific collaboration among different organizations. It consists of a secure data processing platform named FedCube based on the Cloud, a multi-objective cost model and a data placement algorithm. The platform enables secure data processing with the encrypted data stored on the Cloud for the collaboration among different organizations. The multi-objective cost model consists of monetary cost and the execution time. The data placement algorithm creates a data storage plan based on the cost model in order to reduce both monetary cost and the execution time of jobs. We validate our proposed algorithm based on a simulation and extensive experiments on the platform with a widely used benchmark, i.e., Wordcount, and a real-life data processing application for COVID-19 [17]. 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 II reviews related works. Section III presents the system design of the secure data processing platform. Section IV defines the problem, proposes a cost model and the greedy data placement algorithm based on the cost model. Section V shows the experimental results. Finally, we conclude the paper in Section VI.

II. Related Work

In this section, we present the data placement methods and the security techniques on the Cloud.

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 [18], [19], [20]. In addition, the data dependency among different jobs can be exploited to reduce the time and monetary cost to transfer data [21], [22]. 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 [23], while the authors do not consider the cost to store data on the Cloud. Load balancing algorithms [24] or dynamic provisioning algorithms [25] 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.

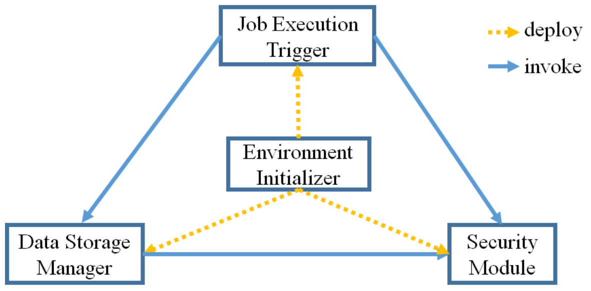


Fig. 1. The functionality architecture of the FedCube platform.

Data security is of much importance to the Cloud users. In order to protect the data security, data accessibility is controlled by attributing different levels of permission to avoid unauthorized or maliciously access to data on the Cloud [26], [27]. In addition, encryption techniques [28] and distributed data storage plan based on data partitioning [29] can be exploited. Federated learning is proposed to train a model while ensuring the data privacy [30], yet it is not applicable to the general data processing among different organizations on the Cloud. In addition, secure separated data processing spaces [14] are proposed to ensure the access control and privacy of data. The separated data processing spaces are disconnected with the public network, which ensures that the confidentiality and the security of data within the local network.

In this paper, we propose an approach that enables secure data processing while reducing the monetary cost and the execution time at the same time. The approach consists of a secure data processing platform, a multi-objective cost model and a greedy data placement algorithm. The platform not only provides different data access controls for different tenants/users but also exploits the secure separated data processing spaces to keep the data secure. The cost model is a weighted function of monetary cost and execution time. The data placement algorithm generates a storage plan to store data on the Cloud with the objectives of reducing the monetary cost and the execution time of jobs.

III. System Design

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

A. 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 the 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

[1]https://aws.amazon.com/
[2]https://azure.microsoft.com/en-us/
[3]https://cloud.baidu.com/

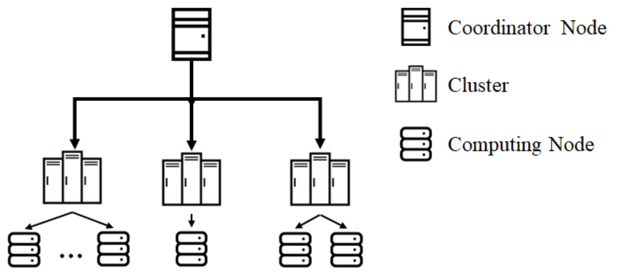


Fig. 2. Infrastructure architecture of the FedCube platform.

Figure 1, the functionality architecture of the platform consists of four modules:

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 user's security configuration, e.g., 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 the connection to any public network, e.g., Internet, which can ensure the confidentiality and the security of the data within the local network.

As shown in Figure 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 local network on the Cloud. Each computing node is created based on the image [31] 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 [31].

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 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 a job, which can be useful for the following execution in order to reduce useless repetitive execution [32].

3) Job Execution Trigger: The job execution trigger starts the execution of 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 III-A1). Then, 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 associate to 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 previous execution.

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 [33]. 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, the data confidentiality and security is ensured by the data interface defined by the data owner while ensuring efficient cooperation among different organizations.

B. 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.

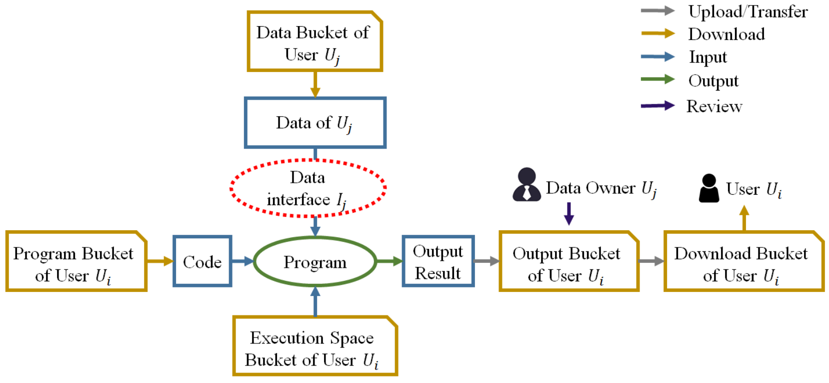

Upload/Transfer
Download
Input
Output
Review
Data Bucket of User U_j
Data of U_j
Data interface I_j
Data Owner U_j
User U_i
Program Bucket of User U_i
Code
Program
Output Result
Output Bucket of User U_i
Download Bucket of User U_i
Execution Space Bucket of User U_i

Fig. 3. Job Execution Workflow.

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 Figure 1. For the n collaborators in the above scenario, we create n accounts (U_t 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 III-A2) for each user. Afterwards, 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, tenants/users can carry out data processing 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 Figure 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, she also gets the necessary information, e.g., the mock data, to access to 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 III-B2), 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 account are removed from the platform by the environment initializer module.

2) Job Life Cycle: The job life cycle consists of four phases, i.e., initialization, data synchronization, job execution and finalization.

Initialization Phase. The initialization phase [34] 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 the 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 [35], [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 [34] 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. As shown in Figure 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], [38], the data storage manager uploads the encrypted intermediate of the job. Afterwards, the environment initializer module removes 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.

IV. Multi-Objective Data Placement

In this section, we first define the problem. Then, we propose a cost model based on two costs, i.e., monetary cost and execution time. Afterwards, we propose a greedy data placement algorithm to generate a plan to store the data with the minimum cost.

A. 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 total cost, which consists of the monetary cost and the execution time of jobs, on the Cloud. The data placement problem is a typical NP-hard problem [39]. A job can be executed multiple times because the user-defined codes are updated or the parameters are updated [40]. As shown in Table I, 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 higher price and higher data access speed while the data access frequency depends on the execution frequency of job execution. In addition, the total cost to execute a job once differs according to the data storage types. Thus, the problem we address in this paper is how to find an optimal storage plan of all the data sets in order to reduce the total cost to execute the jobs with different execution frequencies based on a cost model (see details in Section IV-B).

B. Cost Model

Inspired by [23], 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 are summarized in Table II.

The total cost to execute a set of jobs with a data placement plan is defined as follows:

$$\mathrm{Cost}(Jobs, Plan) = \sum_{j \in Jobs} \mathrm{Cost}(j,t) \tag{1}$$

, where $Jobs$ represents a set of jobs and $Plan$ represents a data placement plan of $Jobs$. t is the data storage type of Job j defined by $Plan$. 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.

The total cost to execute a job per unit of time is defined by:

$$\begin{aligned}\mathrm{Cost}(j,t) =& (w_t * \mathrm{Time_n}(j,t) \\ &+ w_m * \mathrm{Money_n}(j,t)) * \mathrm{f}(j)\end{aligned} \tag{2}$$

where $\mathrm{Time_n}(j,t)$ and $\mathrm{Money_n}(j,t)$ are the normalized time cost and monetary cost; $\mathrm{Time_n}(j,t)$ and $\mathrm{Money_n}(j,t)$ are defined by formula 3 and 7; j represents a job and t represents the storage plan, i.e., the mapping between the storage type and the data; w_t and w_m represents the importance of the monetary cost and execution time of the job. w_t and w_m should be positive values, defined by the user. $\mathrm{f}(j)$ 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.

1) Time Cost: The normalized time cost is defined by the following formula:

$$\mathrm{Time_n}(j,t) = \frac{\mathrm{Time}(j,t)}{DesiredTime} + Penalty \tag{3}$$

with

$$Penalty = \begin{cases} \beta * (\mathrm{Time}(j,t) - DesiredTime), \\ \qquad \text{if } \mathrm{Time}(j,t) > DesiredTime \\ 0, \qquad \text{otherwise} \end{cases} \tag{4}$$

where $\mathrm{Time}(j,t)$ represents the total execution time of the job and $DesiredTime$ represents the estimated execution time of this job, defined by the user. The desired execution time could be larger or smaller than the real execution time $\mathrm{Time}(j,t)$. β with $\beta > 1$ is the parameter for penalty, which increases normalized time cost when $\mathrm{Time}(j,t)$ is longer than $DesiredTime$. The total execution time consists of three parts, which can be defined by:

$$\begin{aligned}\mathrm{Time}(j,t) =& \mathrm{InitializationTime}(j) \\ &+ \mathrm{DataTransferTime}(j,t) \\ &+ \mathrm{ExecutionTime}(j)\end{aligned} \tag{5}$$

where $\mathrm{InitializationTime}(j)$ represents the time to initialize the computing nodes for Job j; $\mathrm{DataTransferTime}(j,t)$ represents the time to transfer the data from the Cloud to computing nodes; $\mathrm{ExecutionTime}(j)$ represents the execution time of Job j. The initialization of the computing nodes for Job j 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 j, e.g., $n * averageInitializationTime$ with n representing the number of computing nodes and $averageInitializationTime$ 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 j and the data storage type t. According to Amdahl's law [41], the

TABLE I
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	$\geq$ 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

TABLE II
DESCRIPTION OF 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

Parameter	Meaning	Origin
DesiredTime	The estimated execution time to execute a job	UD
DesiredMoney	The estimated monetary cost to execute a job	UD
β	The parameter for penalty to increase normalized time	UD
InitializationTime	The time to initialize the computing nodes	Measure
averageInitializationTime	The average time to initialize a computing node	Measure
DataTransferTime	The time to transfer the data from the Cloud to computing nodes	Measure
ComputingSpeedPerCPU	The average computing performance of each computing node	Measure
workload	The workload of a job	Measure
ExecutionTime	The execution time of Job	Execution
TimeQuantum	The time quantum used to charge tenants/users in the Cloud	Cloud
α	The percentage of the workload that can be executed in parallel	Cloud

execution time of Job j can be estimated by the following formula [23]:

$$\mathrm{ExecutionTime}(j) = \frac{(\frac{\alpha}{n} + (1+\alpha)) * \mathrm{workload}(j)}{ComputingSpeedPerCPU} \quad (6)$$

where α represents the percentage of the workload that can be executed in parallel; n is the number of computing nodes, which is configured by users; $\mathrm{workload}(j)$ represents the workload of a job which can be measured by the number of FLOP (FLoat-point Operations) [42]. $ComputingSpeedPerCPU$ is the average computing performance of each computing node, which can be measured by the number of FLOPS (FLoating-point Operations Per Second).

2) Monetary Cost: Normalized monetary cost is defined by the following formula:

$$\mathrm{Money_n}(j,t) = \frac{\mathrm{Money}(j,t)}{DesiredMoney} \quad (7)$$

where $\mathrm{Money}(j,t)$ is the financial cost to rent VMs as computing nodes on the Cloud. $DesiredMoney$ represents the estimate execution money of this job, which can be configured by the user. $DesiredMoney$ can be bigger or smaller than the real monetary cost $\mathrm{Money}(j,t)$. $\mathrm{Money}(j,t)$ can be estimated based on the following formula:

$$\begin{aligned}\mathrm{Money}(j,t) = &\ \mathrm{ExecutionMoney}(j,t)\\ &+ \mathrm{DataStorageMoney}(j,t)\\ &+ \mathrm{DataAccessMoney}(j,t)\end{aligned} \quad (8)$$

where $\mathrm{ExecutionMoney}(j,t)$ represents the monetary cost to use the computing nodes to execute the job; $\mathrm{DataStorageMoney}(j)$ represents the monetary cost to store the data on the Cloud; $\mathrm{DataAccessMoney}(j,t)$ represents the monetary cost to access to the data. $\mathrm{ExecutionMoney}(j,t)$ can be estimated by the following formula:

$$\begin{aligned}\mathrm{ExecutionMoney}(j,t) = &\ \mathrm{VMPrice}(j) * n\\ &* \lceil \frac{\mathrm{Time}(j,t) - \mathrm{InitializationTime}(j)}{TimeQuantum} \rceil\end{aligned} \quad (9)$$

where $\mathrm{VMPrice}(j)$ represents the average cost of each computing node of Job j; n represents the number of computing nodes to execute the job; $TimeQuantum$ represents the time quantum used to charge tenants/users in the Cloud; $\mathrm{Time}(j,t)$ and $\mathrm{InitializationTime}(j)$ are defined in formula 5.

We allocate the storage monetary cost of a data set to the jobs based on the workload. $\mathrm{DataStorageMoney}(j,t)$ is defined by the following formula:

$$\begin{aligned}&\mathrm{DataStorageMoney}(j,t) =\\ &\quad (\sum_{i\in \mathrm{dataset}(j)} [(\mathrm{workload}(j) * \mathrm{f}(j))*\\ &\quad \frac{\mathrm{StoragePrice}(t) * \mathrm{size}(i)}{\sum\limits_{k\in \mathrm{job}(i)} \mathrm{workload}(k) * \mathrm{f}(k)}])/\mathrm{f}(j)\end{aligned} \quad (10)$$

where $\mathrm{workload}(j)$ represents the workload of job j; $\mathrm{dataset}(j)$ represents the data sets that job j uses; $\mathrm{job}(i)$ represents the jobs that takes data i as input data; $\mathrm{StoragePrice}(t)$ represents the monetary cost to store the data with the storage type t on the Cloud; $\mathrm{size}(j)$ represents the size of the input data i.

$\mathrm{DataAccessMoney}(j,t)$ is defined by the following formula:

$$\mathrm{DataAccessMoney}(j,t) = \mathrm{ReadPrice}(t) * \mathrm{size}(j) \quad (11)$$

where $\mathrm{ReadPrice}(t)$ represents the monetary cost to read data from the Cloud; $\mathrm{size}(j)$ represents the size of the input data of Job j.

Algorithm 1 Greedy data placement

```
Input:
    $D$: A set of data; $J$: A set of job related to Data $d$ in $D$; $t$: Storage Type; $StorageTypeList$: The list of Storage Types $t$.
Output:
    $S$: The storage type of each data $d$ in data set $D$ with the minimum cost.
    $\text{cost}_{\text{min}}$: The minimum cost of each data $d$ in data set $D$.
1: for each Data $d$ in $D$ do
2:     S = $\emptyset$
3:     $\text{cost}_{\text{min}} \leftarrow +\infty$
4:     for each Storage type $t$ in $StorageTypeList$ do
5:         $\text{cost}_{\text{tmp}} = 0$
6:         for each Job $j$ in $J$ do
7:             $\text{cost}_{\text{tmp}} = \text{cost}_{\text{tmp}} + \text{Cost}(j, t)$ {According to Formula 2}
8:         end for
9:         if $\text{cost}_{\text{tmp}} < \text{cost}_{\text{min}}$ then
10:            $\text{cost}_{\text{min}} \leftarrow \text{cost}_{\text{tmp}}$
11:            S $\leftarrow S \cup t$
12:        end if
13:    end for
14: end for
```

C. Greedy Data Placement Algorithm

Based on the multi-objective cost model, we propose a greedy data placement algorithm to reduce the total cost to execute a set of jobs on the FedCube platform as shown in Algorithm 1. In the algorithm, for each Data d, we choose the storage type that minimizes the total cost to execute a set of jobs of different execution frequencies. Lines 4 - 14 choose the storage type for a set of jobs J related to Data d. For this set of job J, lines 6 - 8 calculate the total cost of all related jobs with the current storage type using the cost model presented in Section IV-B. Lines 9 - 10 compare the cost of J with cost_{tmp} and chooses the small one as cost_{min}. Line 11 gets the current storage type with the smaller cost. After the calculation of each storage type of all related jobs, the algorithm chooses the storage type with the minimum total cost for this data set d. Then, a data storage plan is generated for a set of data sets D.

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 n^m. The complexity of our proposed algorithm is $m * k * n$, which is much smaller than n^m when $n^{m-1} > m*k$ (a general case).

V. Experimentation

In this section, we first present the simulation to compare the execution time of our proposed greedy 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

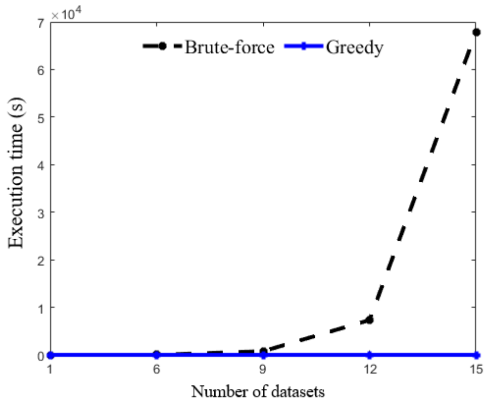


Fig. 4. Execution time of Greedy and Brute-force

four storage methods: greedy, brute-force and standard storage and archive storage. 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 standard storage method [43] uses standard storage to store data all the time while archive [44] storage uses archive storage to store data all the time. We take the standard storage method as a baseline. 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 [45], and a real-life data processing program for the correlation analysis of COVID-19 [17] (COVID-19-Correlation), which is selected from recent work related to COVID-19 [17], [46], [47]. In the experimentation, we consider five execution frequencies (daily, semimonthly, monthly, quarterly and yearly) for Wordcount and COVID-19-Correlation.

A. 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 G 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 $DesiredTime$, w_t. The data sets include DBLP XML files [48] and some data sets from Baidu. The DBLP XML file contains the meta data, e.g., the name of authors, publishers, of computer-based English articles. The comparison experiment results shown in Figure 4.

Figure 4 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 the brute-force method 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 our greedy algorithm remains within 0.0001s.

Figure 5 presents the comparison among four methods: greedy, brute-force, standard storage and archive storage. Our greedy algorithm corresponds to the same total cost as that of

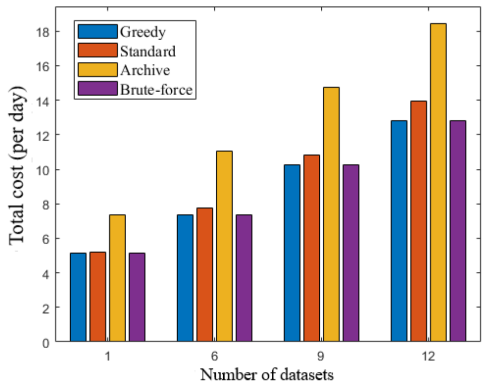


Fig. 5. Comparison among four methods

the brute-force method, which is up to 8.2% and 30.6% smaller than that of the standard storage method and the archive storage method, 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.

B. Wordcount

Hadoop [49] is a framework for parallel big data processing on a cluster of commodity servers. Hadoop contains two components, i.e., HDFS [50] 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 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, $< word, 1 >$ is generated for each work in the input data. Then, the number of $< word, 1 >$ 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 G as the input data.

We assume that the Wordcount job is executed at five different frequencies: daily, semi-monthly, monthly, quarterly and yearly. We set $DesiredTime$ as 1200 seconds and $DesiredMoney$ as 1 dollar.

Figure 6 shows that our proposed algorithm, i.e., the greedy data placement algorithm, significantly outperforms the baseline approach. When the frequency is daily, the total cost corresponding to different approaches are shown in Figure 6(a). When ω_t is 0 and ω_m is 1, the greedy algorithm can reduce the total cost by 8.6% compared with the archive method. Compared with the archive method, the greedy algorithm can reduce the total cost by 25.1% and 42.2% when ω_t is 0.5 and 0.9 respectively. When the frequency is quarterly, Greedy can reduce the total cost by 23.8% compared with the archive method when the ω_t is 0 as Figure 6(b) shows. The greedy algorithm can generate an optimal storage plan, which significantly outperforms (the total cost is 37.1% smaller) the archive method when ω_t is 0.5. The total cost can be reduced by 48.6% using the greedy algorithm compared with the standard method when ω_t is 0.9 and ω_m is 0.1. Figure 6(c) presents the efficiency of our proposed algorithm when the frequency is yearly. Compared with the standard method, our algorithm can reduce the total cost by 60.4% and 27.4% when ω_t is 0.5 and 0.9 respectively. The experiment shows that the advantage of our algorithm is more significant when ω_t is 0, where the total cost can be reduced by 69.8% compared with the archive method.

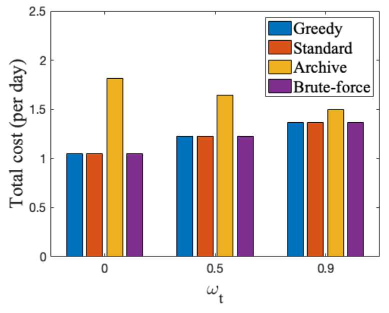
Figure 6 presents that our algorithm can reduce the total cost by up to 60.4% compared with the standard method and up to 69.8% compared with the archive method. As the execution frequency of job decreases, the advantage of our algorithm becomes significant. The comparison of Figure 6(a), 6(b) and 6(c) indicates that as the user's tolerance for time, i.e., ω_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.

C. COVID-19

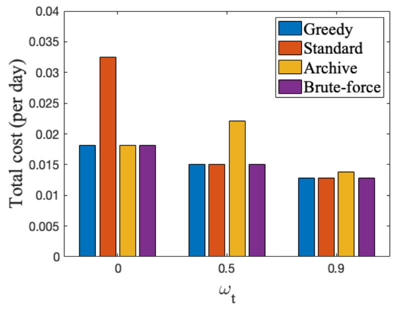
Since the coronavirus disease (COVID-19) has become a global emergency, we reproduced the data processing program for the correlation among COVID-19-related search activities, human mobility and the number of confirmed cases in Mainland China presented in [17]. The data involved in [17] includes the 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. Afterwards, the correlation between any two features is calculated for each city. The experimental results are shown in Figure 7. We set $DesiredTime$ as 600 seconds and $DesiredMoney$ as 0.5 dollar.

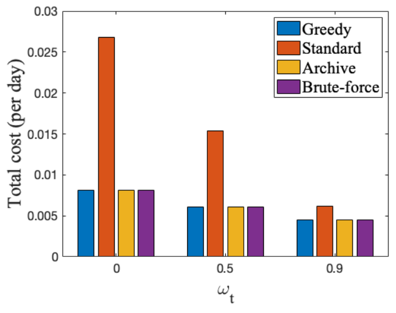
Figure 7 shows that our proposed algorithm, i.e., the greedy data placement algorithm, significantly outperforms the baseline approach (up to 63%) when the size of the input data of the job is smaller that of Wordcount. When the frequency is daily, the total costs of different approaches are shown in Figure 7(a). When ω_t is 0 and ω_m is 1, our algorithm can reduce the total cost by 30.5% compared with the archive



(a) Daily

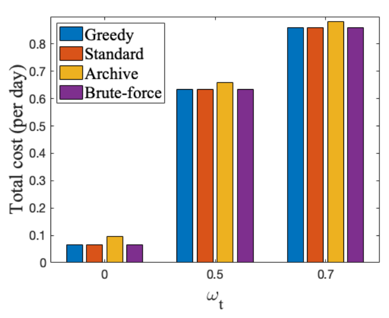


(b) Quarterly

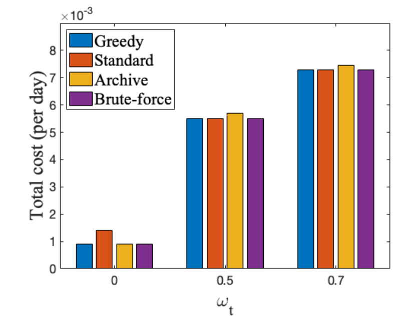


(c) Yearly

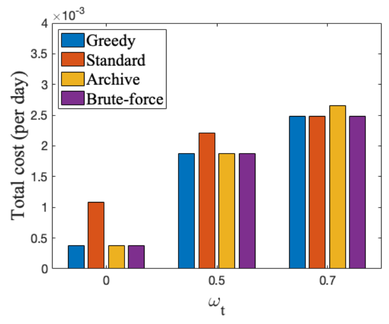
Fig. 6. Total cost of Wordcount



(a) Daily



(b) Quarterly



(c) Yearly

Fig. 7. Total cost of COVID-19-Correlation.

method. When ω_t increases to 0.5, our algorithm can reduce the total cost by 2.6% compared with the archive method. When the importance of time, i.e. ω_t, increases to 0.7, our algorithm can outperform the archive method, the total cost can be reduced by 2.2%. Figure 7(b) 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 the standard method. With the increase of ω_t, our algorithm can reduce the total cost by 3.7% and 1.9% compared with the archive method when ω_t is 0.5 and 0.7 respectively. When the frequency is yearly, the execution results are shown in Figure 7(c). The most significant result is that our algorithm can reduce the total cost by 63% compared with the standard method when ω_t is 0 and ω_m is 1. When ω_t is 0.5 and 0.7, our algorithm can reduce the total cost by 18.2% and 7.7% compared with the standard method and the archive method, respectively.

Our proposed algorithm, i.e., the greedy data placement algorithm, significantly outperforms the standard method (up to 63%) and the archive method (up to 37.1%), when the frequency of the job execution is high and when the size of the input data of the job is big. In addition, we have the same findings as presented in Section V-B.

VI. Conclusion

When organizations outsource their data on the Cloud, it is critical to choose a proper data placement strategy to reduce the 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 greedy data placement algorithm. The cost model consists of monetary cost and execution time. The greedy algorithm generates a data placement plan with the minimum total cost based on the cost model. We carried out extensive experiments to validate our proposed algorithm. The experimental results indicate that out proposed algorithm outperforms the baseline approach up to 69.8%. Experiments show that our algorithm can generate the same optimal solution as the brute-force method with a short execution time.

VII. Acknowledgement

This work is supported in part by National Key R&D Program of China (No. 2018YFB1402600).

References

[1] I. Greif and S. K. Sarin, "Data sharing in group work," *ACM Transactions on Office Information Systems*, vol. 5, no. 2, pp. 187–211, 1987.

[2] J. Liu, L. Pineda-Morales, E. Pacitti, A. Costan, P. Valduriez, G. Antoniu, and M. Mattoso, "Efficient scheduling of scientific workflows using hot metadata in a multisite cloud," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 10, pp. 1940–1953, 2019.

[3] P. Voigt and A. von dem Bussche, *The EU General Data Protection Regulation (GDPR): A Practical Guide*, 1st ed. Springer Publishing Company, Incorporated, 2017.

[4] A. Qi, G. Shao, and W. Zheng, "Assessing china's cybersecurity law," *Computer Law & Security Review*, vol. 34, no. 6, pp. 1342 – 1354, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0267364918303157

[5] E. Valentijn, K. Begeman, A. Belikov, D. Boxhoorn *et al.*, "Target and (astro-)wise technologies data federations and its applications," in *Astroinformatics*, ser. Proceedings of the International Astronomical Union, vol. 12, no. S325, 2016, pp. 333–340.

[6] S. Srirama, O. Batrashev, and E. Vainikko, "Scicloud: scientific computing on the cloud," in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 579–580.

[7] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, “Parallelization of Scientific Workflows in the Cloud,” INRIA, Research Report RR-8565, 2014. [Online]. Available: https://hal.inria.fr/hal-01024101

[8] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, “Above the clouds: A berkeley view of cloud computing,” *Department Electrical Engineering and Computer Sciences, University of California, Berkeley*, 2009.

[9] N. Kratzke, “A brief history of cloud application architectures,” *Applied Sciences*, vol. 8, 2018.

[10] 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.

[11] M. Tebaa, S. El Hajji, and A. El Ghazi, “Homomorphic encryption applied to the cloud computing security,” in *Proceedings of the World Congress on Engineering*, vol. 1, no. 2012, 2012, pp. 4–6.

[12] X. Boyen and B. Waters, “Anonymous hierarchical identity-based encryption (without random oracles),” in *Annual International Cryptology Conference*, 2006, pp. 290–307.

[13] X. Yu and Q. Wen, “A view about cloud data security from data life cycle,” in *2010 international conference on computational intelligence and software engineering*, 2010, pp. 1–4.

[14] A. Guabtni, F. Charoy, and C. Godart, “Customizable isolation in transactional workflow,” in *Interoperability of Enterprise Software and Applications*, 2006, pp. 197–202.

[15] A. Kaur, P. Gupta, M. Singh, and A. Nayyar, “Data placement in era of cloud computing: a survey, taxonomy and open research issues,” *Scalable Computing: Practice and Experience*, vol. 20, no. 2, pp. 377–398, 2019.

[16] L. Singh and J. Malhotra, “A survey on data placement strategies for cloud based scientific workflows,” *International Journal of Computer Applications*, vol. 141, no. 6, pp. 30–33, 2016.

[17] H. Xiong, J. Liu, J. Huang, S. Huang, H. An, Q. Kang, Y. Li, D. Dou, and H. Wang, “Understanding the collective responses of populations to the covid-19 pandemic in mainland china,” *medRxiv*, 2020.

[18] Ü. V. Çatalyürek, K. Kaya, and B. Uçar, “Integrated data placement and task assignment for scientific workflows in clouds,” in *Proceedings of the fourth international workshop on Data-intensive distributed computing*, 2011, pp. 45–54.

[19] L. Golab, M. Hadjieleftheriou, H. Karloff, and B. Saha, “Distributed data placement to minimize communication costs via graph partitioning,” in *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, 2014, pp. 1–12.

[20] L. Guo, Z. He, S. Zhao, N. Zhang, J. Wang, and C. Jiang, “Multi-objective optimization for data placement strategy in cloud computing,” in *International Conference on Information Computing and Applications*, 2012, pp. 119–126.

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

[22] 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 *21st IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2017, pp. 433–438.

[23] J. Liu, E. Pacitti, P. Valduriez, D. de Oliveira, and M. Mattoso, “Multi-objective scheduling of scientific workflows in multisite clouds,” *Future Generation Computer Systems*, vol. 63, pp. 76–95, 2016.

[24] M. Kumar and S. C. Sharma, “Dynamic load balancing algorithm for balancing the workload among virtual machine in cloud computing,” *Procedia computer science*, vol. 115, pp. 322–329, 2017.

[25] N. Tziritas, S. U. Khan, C. Xu, T. Loukopoulos, and S. Lalis, “On minimizing the resource consumption of cloud applications using process migrations,” *Journal of Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1690–1704, 2013.

[26] M. Farsi, M. Ali, R. A. Shah, A. A. Wagan, and R. Kharabsheh, “Cloud computing and data security threats taxonomy: A review,” *Journal of Intelligent and Fuzzy Systems*, vol. 38, no. 3, pp. 2517–2527, 2020.

[27] F. M. Awaysheh, M. Alazab, M. Gupta, T. F. Pena, and J. C. Cabaleiro, “Next-generation big data federation access control: A reference model,” *Future Generation Computer Systems*, 2020.

[29] Y. Singh, F. Kandah, and W. Zhang, “A secured cost-effective multi-cloud storage in cloud computing,” in *Computer Communications Workshops (INFOCOM WKSHPS)*, 2011.

[28] G. Anthes, “Security in the cloud,” *Communications of the ACM*, vol. 53, no. 11, pp. 16–18, 2010.

[30] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, ser. Proceedings of Machine Learning Research, vol. 54, 2017, pp. 1273–1282.

[31] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning, “Managing security of virtual machine images in a cloud environment,” in *ACM Workshop on Cloud Computing Security*, 2009, p. 91–96.

[32] 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 *Int. Conf. on Database and Expert Systems Applications (DEXA)*, ser. Lecture Notes in Computer Science, vol. 11707, 2019, pp. 452–466.

[33] T. Jamil, “The rijndael algorithm,” *IEEE Potentials*, vol. 23, no. 2, pp. 36–38, 2004.

[34] S. Bardhan and D. A. Menascé, “The anatomy of mapreduce jobs, scheduling, and performance challenges,” in *39th International Computer Measurement Group Conference*, 2013.

[35] G. Stellner, “Consistent checkpoints of pvm applications,” in *Proceedings of the First European PVM User Group Meeting*, 1994.

[36] D. Yoo and K. M. Sim, “A comparative review of job scheduling for mapreduce,” in *IEEE International Conference on Cloud Computing and Intelligence Systems*, 2011, pp. 353–358.

[37] I. Surjandari, A. Rachman, A. Dhini *et al.*, “The batch sheduling model for dynamic multiitem, multilevel production in an assembly job-shop with parrallel machines.” *International Journal of Technology*, vol. 1, pp. 84–96, 2015.

[38] D. Abdullah, R. Srivastava, and M. Khan, “Testability measurement framework: Design phase perspective,” *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 3, no. 11, pp. 8573–8576, 2014.

[39] X. Ren, P. London, J. Ziani, and A. Wierman, “Datum: Managing data purchasing and data placement in a geo-distributed data market,” *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 893–905, 2018.

[40] S. Chunduri, M. Ghaffari, M. S. Lahijani, A. Srinivasan, and S. Namilae, “Parallel low discrepancy parameter sweep for public health policy,” in *Int. Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2018, pp. 291–300.

[41] X.-H. Sun and Y. Chen, “Reevaluating amdahl’s law in the multicore era,” *Journal of Parallel and Distributed Computing*, vol. 70, no. 2, p. 183–188, 2010.

[42] R. Coutinho, L. Drummond, Y. Frota, D. de Oliveira, and K. Ocana, “Evaluating grasp-based cloud dimensioning for comparative genomics: a practical approach,” in *2014 IEEE International Conference on Cluster Computing*, 2014, pp. 371–379.

[43] M. Darwich, Y. Ismail, T. Darwich, and M. A. Bayoumi, “Cost-efficient storage for on-demand video streaming on cloud,” *CoRR*, vol. abs/2007.03410, 2020. [Online]. Available: https://arxiv.org/abs/2007.03410

[44] R. Black, A. Donnelly, D. Harper, A. Ogus, and A. I. T. Rowstron, “Feeding the pelican: Using archival hard drives for cold storage racks,” in *USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage*, N. Agrawal and S. H. Noh, Eds. USENIX Association, 2016. [Online]. Available: https://www.usenix.org/conference/hotstorage16/workshop-program/presentation/black

[45] T. White, *Hadoop: The Definitive Guide*, 4th ed. O’Reilly Media, Inc., 2015.

[46] J. Liu, T. Huang, H. Xiong, J. Huang, J. Zhou, H. Jiang, G. Yang, H. Wang, and D. Dou, “Analysis of collective response reveals that covid-19-related activities start from the end of 2019 in mainland china,” *medRxiv preprint*, 2020.

[47] J. Liu, X. Wang, H. Xiong, J. Huang, S. Huang, H. An, D. Dou, and H. Wang, “An investigation of containment measures against the covid-19 pandemic in mainland china,” *arXiv preprint arXiv:2007.08254*, 2020.

[48] “Dblp: computer science bibliography,” https://dblp.org/xml/.

[49] “Apache hadoop,” http://hadoop.apache.org/.

[50] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, p. 1–10.