Cross-cloud MapReduce for Big Data

Peng Li, *Member, IEEE,* Song Guo, *Senior Member, IEEE,* Shui Yu, *Member, IEEE,* and Weihua Zhuang, *Fellow, IEEE*

Abstract—MapReduce plays a critical role as a leading framework for big data analytics. In this paper, we consider a geo-distributed cloud architecture that provides MapReduce services based on the big data collected from end users all over the world. Existing work handles MapReduce jobs by a traditional computation-centric approach that all input data distributed in multiple clouds are aggregated to a virtual cluster that resides in a single cloud. Its poor efficiency and high cost for big data support motivate us to propose a novel data-centric architecture with three key techniques, namely, cross-cloud virtual cluster, data-centric job placement, and network coding based traffic routing. Our design leads to an optimization framework with the objective of minimizing both computation and transmission cost for running a set of MapReduce jobs in geo-distributed clouds. We further design a parallel algorithm by decomposing the original large-scale problem into several distributively solvable subproblems that are coordinated by a high-level master problem. Finally, we conduct real-world experiments and extensive simulations to show that our proposal significantly outperforms the existing works.

Index Terms—big data, MapReduce, cloud, optimization, parallel algorithm

1 Introduction

The dramatic growth of data volume in recent years imposes an emerging issue of processing and analyzing a massive amount of data. As a leading framework for big data analytics that is pioneered by Google and popularized by the open-source Hadoop [1], MapReduce [2] is leveraged by a large number of enterprises to parallelize their data processing on distributed computing systems. It decomposes a job into a number of parallel map tasks, followed by reduce tasks that merge all intermediate results generated by map tasks to produce the final results.

MapReduce jobs are usually executed on clusters of commodity PCs, which require large investment in hardware and management. Since a cluster must be provisioned for peak usage to avoid overload, it is underutilized on average. Thus, cloud becomes a promising platform for MapReduce jobs because of its flexibility and pay-as-you-go business model. For each MapReduce job, a virtual cluster is created by employing a number of virtual machines (VMs). The size of the cluster can be dynamically adjusted according to job requirements. However, the services provided by an individual cloud provider are typically limited to certain geographic regions, making it impossible to process data from all over the globe. To truly fulfill the promise of cloud computing for big data application, an emerging scheme is to store and process data in a geographically distributed cloud environment, where multiple clouds locate at different places in the world and they are connected by inter-cloud networks [3].

In this paper, we consider such big data applications in geo-distributed clouds. Each cloud collects data from its geographic region and stores them into its storage infrastructure, e.g., Amazon S3. To execute a big data application, a number of MapReduce jobs are created, each of which targets on a set of input data that are often distributed in multiple clouds. Existing work [4] handles this situation by aggregating these data to a virtual cluster that is created at a single cloud. However, this data aggregation scheme would lead to poor efficiency and high cost for big data workload as explained as follows.

First, the amount of data needed to be aggregated can be so large that the resulting data traffic would occupy excessive bandwidth of inter-cloud network. The situation would become aggravated when multiple jobs exist in the system. The inter-cloud network is a critical infrastructure for providers of online services such as Amazon, Google, and Microsoft. When more services rely on low-latency and high-bandwidth inter-cloud communication for good user experience, the inter-cloud network becomes a scarce resource that should be efficiently utilized. Loading big data remotely will incur significant communication cost, and also affect the performance of all services sharing the bandwidth of inter-cloud networks.

Second, a large cluster is usually preferred by MapReduce jobs for big data processing. For example, the Hadoop cluster of Facebook includes 1200 8-core machines and 800 16-core machines [5]. Each machine has 12TB local storage. On the other hand, the size of the virtual MapReduce cluster that can be supported by a single cloud is limited due to

- *P. Li and S. Guo are with the School of Computer Science and Engineering, The University of Aizu, Japan. E-mail: {pengli, sguo}@u-aizu.ac.jp*
- *S. Yu is with the School of Information Technology, Deakin University, Australia. E-mail: syu@deakin.edu.au.*
- *W. Zhuang is with the School of Electrical and Computer Engineering, University of Waterloo, Canada. E-mail: wzhuang@uwaterloo.ca.*

resource or cost constraints. Therefore, the traditional data aggregation scheme cannot handle the big data jobs with demand exceeding the residual resources at any cloud.

Third, constraining a virtual cluster within one cloud fails to exploit the diversity of distributed cloud environment. The cost of maintaining a virtual cluster at different clouds depends on their distinct local electricity prices. For example, the electricity grid of some regions in U.S. is managed by a Regional Transmission Organization (RTO) that operates wholesale electricity market according to power supply and demand, such that electricity price may vary on an hourly or 15-min basis.

The crux of above issues is that MapReduce jobs are handled using a traditional computation-centric method. In this paper, we propose a novel big data-centric architecture that supports MapReduce in a geo-distributed cloud environment. In particular, the following three key techniques are exploited to reduce the cost of running a set of MapReduce jobs.

(1) Cross-cloud virtual cluster: As shown in Table 1, the map tasks of some jobs accept large data as input, but generate only a small amount of intermediate data as output. For this kind of jobs, instead of aggregating data to a virtual cluster at a single cloud for centralized processing, we maintain a cross-cloud virtual cluster by placing map tasks on the clouds where input data reside. In such a way, reduce tasks only need to fetch small intermediate data over the inter-cloud network, leading to a reduced bandwidth cost. Moreover, the computation cost can be also optimized by exploiting cloud diversity at a fine-grained task level.

(2) Data-centric job placement: Existing solutions deal with jobs independently without considering the correlation among their input data. Motivated by the inter-job data characteristic that some common data are often shared by related jobs, we propose a data-centric job placement over distributed clouds such that the duplicate data loading through inter-cloud network will be significantly reduced.

(3) Network coding (NC) based traffic routing: When a common block needs to be disseminated to multiple jobs across clouds, a multicast session over the inter-cloud network is formed. By addressing the challenges of destination uncertainty and routing, we propose an NC-based scheme to achieve maximum multicast flow.

The main contributions of this paper are summarized as follows.

- We propose a novel big data-centric architecture for multiple MapReduce jobs in geo-distributed cloud environment. The intra-job and inter-job data characteristics are exploited by cross-cloud virtual cluster and data-centric job placement, respectively. To optimize the data dissemination among clouds, we propose an NC-based traffic routing scheme to achieve maximum flow.
- Second, we prove that the cost minimization problem for a set of MapReduce jobs is NP-complete. To solve this problem, we propose an optimization framework by jointly considering cross-cloud virtual cluster, data-centric job placement, and NC-based traffic routing.
- Third, to deal with the large-scale problem with big data, we design a parallel algorithm by decomposing the original large-scale problem into several distributively solvable subproblems that are coordinated by a high-level master problem.
- Finally, we conduct both real-world experiments on Amazon EC2 and extensive simulations to evaluate the performance of our proposal. The results show that our proposal outperforms existing work significantly.

The rest of this paper is organized as follows. We introduce some necessary preliminaries and important related work in Section 2. Section 4 presents the system model. Three key design issues are presented in Section 5. In Section 6, we prove the NP-hardness of the cost minimization problem and propose an optimization framework to solve this problem. A parallel algorithm design is presented in Section 7, and the performance evaluation results are given in Section 8. Finally, we conclude our paper in Section 9.

2 Preliminaries

2.1 MapReduce

MapReduce is a software framework for big data processing on large clusters consisting of hundreds or thousands of machines. Users submit a data processing request, referred to as a job in MapReduce, by specifying a map and a reduce function. When a job is executed, two types of tasks, i.e., map and reduce, are created. The input data are divided into independent chunks that are processed by map tasks in parallel. The generated intermediate results in forms of key-value pairs may be shuffled and sorted by the framework, and then are fetched by reduce tasks to produce final results. Note that some MapReduce jobs may have only map tasks. As an open source implementation of MapReduce, Hadoop uses a master-slave architecture to organize both computation and storage resources in a cluster. In the master node, a JobTracker is responsible for data and task assignment, and a Namenode keeps the directory tree of all files in the Hadoop Distributed File System (HDFS). In each slave node, a TaskTracker and a Datanode deal with task execution and data storage, respectively.

2.2 Network coding

The data dissemination in traditional networks follows a simple store-and-forward principle, in which each node first stores the received data packets and

TABLE 1
Data characteristics of different jobs

Type	Job	Input data	Intermediate data
I	data filtering and reduction (e.g., grep)	662MB	5.6MB
II	data statistics (e.g., wordcount)	662MB	905MB
III	data transformation (e.g.,sort)	1GB	1 GB

then forwards them to next-hop nodes. Network coding, which is first presented in the pioneering paper [6], breaks such assumption by allowing intermediate nodes to combine the received packets together before forwarding them if they can be successfully decoded at the destinations. Network coding can be categorized into two types: inter-session network coding and intra-session network coding. In the former, coding operations are allowed among packets belonging to different sessions or flows. The latter restricts coding operations to packets belonging to the same session, which is used in our design in this paper.

3 Related work

3.1 MapReduce on clusters

There exists a large body of work for efficient resource allocation and job scheduling for MapReduce on clusters [7], [8], [9], [10]. It has been well recognized that the data locality plays a critical role in the performance of MapReduce. Zaharia et al. [11] propose a fair job scheduler for MapReduce at Facebook by taking data locality and interdependence between map and reduce tasks into consideration. Hindman et al. [12] propose Mesos, a platform that can achieve data locality for multiple jobs sharing a commodity cluster. Want et al. [13] focus on making a balance between data locality and load balancing to simultaneously maximize throughput and minimize delay. For this purpose, they propose a new queuing architecture and propose a map task scheduling algorithm. The most recent work [14] studies the problem of reducing the fetching cost for reduce tasks and formulates a stochastic optimization framework to improve data locality for reduce tasks.

3.2 MapReduce and virtualization

There is a growing trend to deploy MapReduce in virtuaulized environment. Amazon Elastic MapReduce [15] is a publicly offered web service that runs on virtualized cloud platform. Gunarathne et al. [16] propose a novel MapReduce architecture that successfully leverages the scalable Azure infrastructure provided by Microsoft to provide an efficient, on-demand alternative to traditional MapReduce clusters. Chohan et al. [17] use Amazon Spot Instances to improve the runtime of MapReduce jobs and propose new techniques to reduce the adverse effects caused by spot instance termination due to budget constraints during the jobs. Purlieus [18] is proposed as a MapReduce resource allocation system aiming at enhancing the performance of MapReduce jobs in the cloud. Heuristic algorithms are designed to improve locality during both map and reduce phases. Kang et al. [19] propose a MapReduce Group Scheduler (MGS) to improve the efficiency and fairness of existing virtual machine manager scheduler by taking characteristics of MapReduce workloads into consideration. They implement MGS by modifying the famous Xen hypervisor and evaluate the performance on Hadoop. Sharma et al. [20] consider a hybrid compute cluster consisting of native and virtual servers, and propose a 2-phase hierarchical scheduler called HybridMR for effective resource management.

To deal with the MapReduce jobs with geo-distributed data, several recent efforts have been made to build a virtual cluster cross multiple clouds. Cardosa et al. [21] have shown that a single-cluster MapReduce architecture may not suitable for situations where data and computational resources are widely distributed. They consider and compare two alternatives, i.e., a global MapReduce architecture, and multi-layer MapReduce architecture. Iordache et al. [22] have proposed Resilin, a novel EMR (Elastic MapReduce) API compatible system that allows users to perform MapReduce computations across a wide range of resources from private, community, and public clouds. Although the basic idea of above work is similar with our proposal, they mainly focus on performance measurement or API development for a single MapReduce job, ignoring the inter-job and intra-job characteristics as we exploit in this paper. Moreover, our proposed NC-based traffic routing and the corresponding global performance optimization framework have never been considered by previous work. Mandal et al. [23] have implemented and evaluated a Hadoop cluster across multiple clouds. However, they mainly focus on experimental measurement, and do not study performance optimization. BStream [24] has been proposed as a cross-cloud MapReduce framework that couples stream-processing in the external cloud and Hadoop workloads in the internal cloud. We consider a more general model with multiple clouds (more than two) and the associated traffic routing problem among multiple clouds is studied in our paper by the first time.

4 System model

We consider a geo-distributed cloud architecture consisting of several clouds located in different geographical regions. It provides a platform for global applications that persistently collects data from end users spread across the world, while offering a set of services, such as searching, sorting, and data mining, on these data. This geo-distributed cloud environment can be modeled by a direct graph $G(N, A)$, where set N and A denote cloud sites and dedicate inter-cloud links, respectively. Each cloud provides both storage and computation infrastructures. The data collected from corresponding regions are stored in the storage clouds persistently, e.g., Amazon S3. The computation cloud contains a collection of interconnected and virtualised servers, e.g., Amazon EC2.

The stored input data are organized as multiple blocks, i.e., $B_i = \{b_i^1, b_i^2, ..., b_i^{|B_i|}\}$. The size of block $b_i^u \in B_i$ is denoted by $|b_i^u|$. Given a set of MapReduce jobs V of different types, each job $v \in V$ is assigned a virtual cluster with a number of virtual machines (VMs) in computation clouds. The input data of each job may be distributed in multiple clouds. Before its execution, the input data of job v are loaded from storage clouds to the distributed file system of the virtual cluster, e.g., HDFS. After processing, the outputs of each job are stored back to storage clouds and then the corresponding virtual cluster is destroyed by releasing all associated VMs. In a real Hadoop cluster, data are stored in HDFS (Hadoop Distributed File System) that replicates each data blocks into several copies for fault tolerant. Our system model and formulation do not conflict with HDFS because although multiple copies of each data block are stored in HDFS, only one copy will be loaded for computation and be transfered over the inter-cloud network if remote loading is needed.

A cloud usually provides several types of VMs to meet various requirements. For example, Amazon EC2 [25] provides standard VM instances that are different in number of virtual cores, memory size, and local storage. Each combination has a different price. Even for the same type of VMs, the charge varies from cloud to cloud due to different local electricity price and maintenance cost. To evaluate the computation cost of job v at cloud i, we define p_i^v and q_i^v as the cost of processing a unit data element by map and reduce tasks, respectively. Note that we have $p_i^v \neq q_i^v$ in general because map and reduce run different functions. Similarly, the cost of transmitting an unit data element from cloud i to j is denoted by a_{ij}. Based on the system model, we study a cost minimization problem that is defined as follows.

Definition 1: The problem of Cost Minimization for Cross-cloud MapReduce (CMCM): given a number of geo-distributed clouds, and a set of MapReduce jobs, the CMCM problem seeks for a job placement and data routing scheme across the clouds such that the total cost of computation and transmission is minimized.

5 Design Issues

In this section, we present our three key techniques: cross-cloud virtual cluster, data-centric job placement, and NC-based traffic routing, for the CMCM problem.

5.1 Cross-cloud virtual cluster

The main idea of cross-cloud virtual cluster comes from the fact that the input data of some MapReduce jobs may exist in multiple clouds. For example, a healthcare application collects the health records of users of all ages from different regions. To analyze the health condition of people with a specified age from all regions, the input data are distributed in multiple clouds. To deal with such an application using traditional computation-centric approach, a virtual cluster at a single cloud is created and then the required data distributed at other clouds are loaded to the distributed file system of this cluster. For a big data application, the size of input data is often large for its MapReduce jobs. Thus, delivering input data across clouds can occupy lots of bandwidth and incur significant delay for the whole MapReduce job.

We reconsider the problem from a data-centric perspective by analyzing the data characteristic of MapReduce jobs. We find that the volumes of input and output data of map tasks differ significantly for various types of workloads, according to which we categorize MapReduce jobs into three types as shown in Table 1. The jobs of type I, i.e., data filtering (e.g., grep), have a large amount of input data but their map tasks generate less intermediate data, while the jobs of type II, i.e., data reduction (e.g., wordcount), are fed with small input data but produce a large size of intermediate results. The jobs, e.g., sort, that have similar volumes of input and output data of map tasks are included in type III, i.e., data transformation. We also collect Hadoop traces of about 16000 jobs on a 600-machine cluster at Facebook [26] and show the distribution of ratio of the intermediate data size to the input data size in Fig. 1, where the ratios are expressed in $\ln$ form. There are about 70% of jobs whose ratio is less than 1 (or 0 in the $\ln$ scale), i.e., their input data are larger than their corresponding intermediate data.

Preceding observations motivate us to create a cross-cloud virtual cluster that allows map tasks to process the data in each cloud locally and then to aggregate the intermediate results. Its basic idea can be illustrated using the example in Fig. 2, where the input data of a MapReduce job are equally distributed in two clouds. For clarity, we use the Hadoop trace of a typical data aggregate job from [27], whose input data and intermediate data size is illustrated in the figure. The communication cost between two clouds

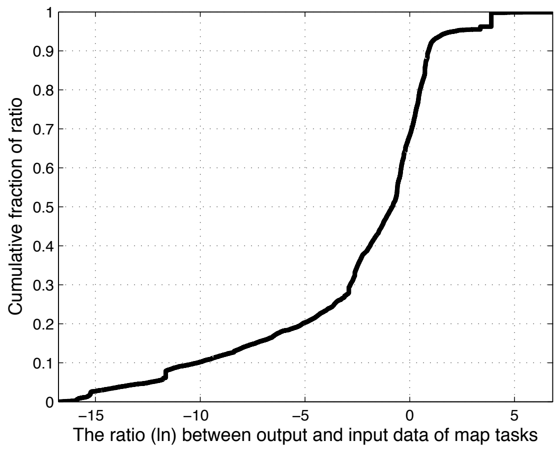

Fig. 1. The distribution of ratio between the amounts of intermediate results and input data.

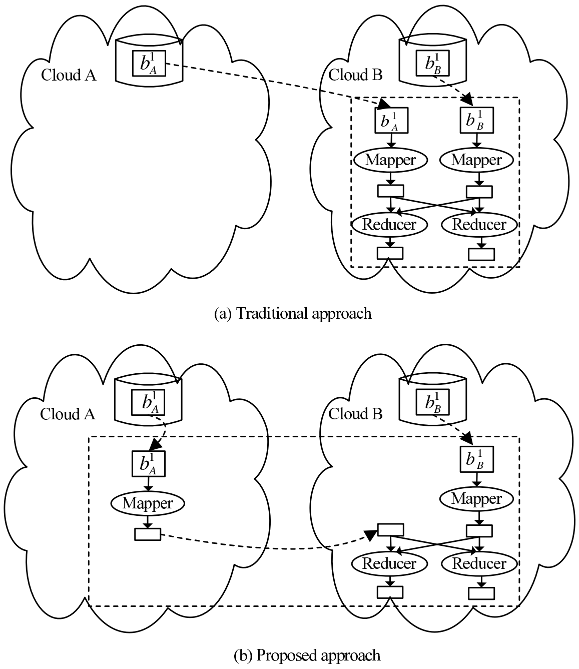

Fig. 2. Illustration of cross-cloud virtual cluster.

is 0.01$/GB [25], and the processing cost of map and reduce tasks is 3.59×10^{-4}\$/GB and 6.25×10^{-3}\$/GB, respectively, at both clouds [4], [27]. If a virtual cluster is created in cloud B using traditional computation-centric approach, the data stored in cloud A need to be loaded to B for MapReduce processing as shown in Fig. 2(a), and the total cost is \$2.575. When a cross-cloud virtual cluster is created, as shown in Fig. 2(b), the data in cloud A can be processed by local map tasks, such that only a small amount of generated intermediate results are aggregated with a reduced transmission cost, leading to a reduced total cost \$0.3631.

In addition to reducing inter-cloud traffic, cross-cloud virtual cluster promises a fine-grained resource

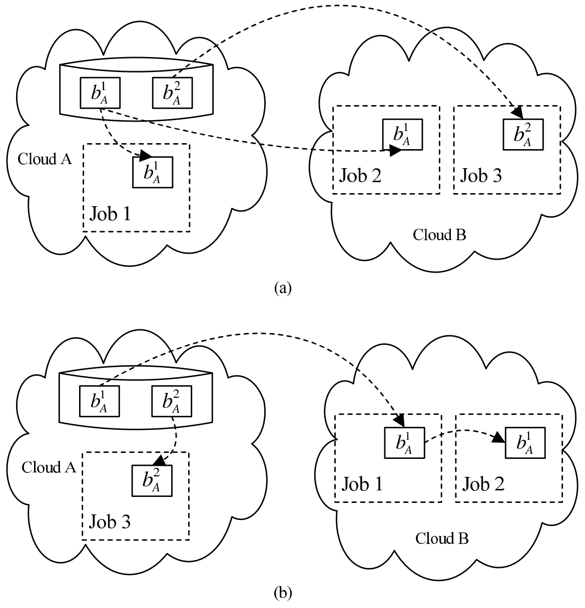

Fig. 3. Illustration of data-centric job placement.

allocation at a task-level instead of a job-level in traditional approach. It enables us to exploit cloud diversity to reduce computation cost by appropriately placing the map and reduce tasks, since each cloud may charge different prices for even the same type of VMs in our geo-distributed cloud architecture. Although the example in Fig. 2 shows that placing map tasks closer to the input data can reduce the inter-cloud data traffic, it doesn't means that it always the best approach with minimum cost when there are multiple jobs with different types. In general cases, we need to find the optimal placement for both map and reduce tasks by jointly considering job types and cloud pricing diversity.

In summary, our cross-cloud approach achieves high flexibility in choosing VM resources for each MapReduce job by exploiting the intra-job data characteristics.

5.2 Data-centric job placement

This section extends our data-centric approach to deal with job placement across clouds using intra-job data characteristics. It is motivated by the observation that several jobs often target on some common data. Consider again the healthcare example, in which each type of disease is analyzed by a different job. The health records of patients with several diseases are simultaneously targeted by multiple jobs. Note that the input data of jobs may completely or partially overlapped. For example, diagnosing pneumonia [28] or appendicitis [29] both require blood test which is used to confirm an infection. In addition, diagnosis of pneumonia still requires a chest X-ray test, while diagnosis of appendicitis requires an abdominal ultra-sound or CT scan for confirmation.

We use the example in Fig. 3 to illustrate our basic idea of data-centric job placement. Suppose two data blocks b_A^1 and b_A^2 are stored in cloud A. Both jobs 1 and 2 need b_A^1 as input data, and job 3 targets on data block b_A^2. We consider a case that the available resources in cloud 1 can support at most one job, while cloud 2 has sufficient resources. One solution, as shown in Fig. 3(a), is to place job 1 on cloud A and the other two on cloud B, resulting in that both data blocks b_A^1 and b_A^2 need to be remotely loaded for jobs 2 and 3, respectively. A more sophisticated solution is to place both jobs 1 and 2 that require the same block b_A^1 on remote cloud B, and to leave job 3 on cloud A, such that only one data block is delivered across clouds as shown in Fig. 3(b).

Combined with cross-cloud virtual cluster, the data-centric job placement has great advantages in reducing transmission and computation cost at both intra- and inter-job levels. On the other hand, it becomes more challenging to determine the optimal placement when multiple common data blocks are shared by a set of jobs, which will be solved in next section.

5.3 NC-based traffic routing

When a common data block needs to be disseminated to multiple virtual clusters that may reside in different clouds, the traditional approach that each virtual cluster loads input data independently would cause duplicate data transmissions. To solve this problem, we propose an NC-based scheme that fully exploits the benefits of multicast by adopting random linear network coding. At source, a coded packet a_i' is a linear combination of K native packets $a_j (1 \leq j \leq K)$: $a_i' = \sum_{j=1}^{K} \alpha_{ij} a_j$, where α_{ij} is the coding coefficient chosen from a Galois Field (for example, $GF(2^8)$). At any intermediate node, it is created by randomly mixing the coded packets in the buffer as $a_i'' = \sum_j \beta_{ij} a_j'$, which is still a linear combination of K native packets if β_{ij} is chosen from the same Galois Field. The node receiving a packet successfully first checks whether it is linearly independent with the coded packets already in its buffer. If it is independent, this innovative packet is inserted into its buffer; otherwise, it is discarded.

The following two challenges need to be addressed. First, in contrast to the standard multicast with specific source and destinations, to determine the destination clouds for common data is also a part of the problem. In other words, it is strongly related with the cross-cloud virtual cluster construction and job placement for a set of MapReduce jobs. Although we can handle the problem by optimizing the computation cost and transmission cost separately (i.e., to first conduct job placement that determines destinations of each block, and then to find the best routing to them), the additional benefit of joint optimization will be lost. Second, even destinations of a common block is given, the problem of finding an optimal multicast tree is proven NP-complete.

Fortunately, the NC-based flow routing can achieve the maximum rate for multicast sessions [6]. Furthermore, it facilitates the formulation of a joint optimization of job placement and traffic routing, which will be shown in next section.

6 OPTIMIZATION FRAMEWORK

In this section, we first analyze the hardness of the CMCM problem by proving it NP-hard. Then, we propose an optimization framework to solve the CMCM problem by jointly considering cross-cloud cluster, data-centric job placement and NC-based traffic routing.

6.1 Hardness analysis

We show the NP-hardness of the CMCM problem for a set of MapReduce jobs by reducing the well-known set cover problem that has been proven NP-compete.

Theorem 1: The CMCM problem is NP-hard.

Proof: In order to prove an optimization problem to be NP-hard, we need to show the NP-completeness of its decision form, which is formalized as follows.

The CMCM_D problem

INSTANCE: Given a number of geo-distributed clouds, a set of MapReduce jobs, and a positive constant C.

QUESTION: Is there a job placement and routing scheme such that the total cost is no greater than C?

It is easy to see that the CMCM_D problem is in NP class as the objective function associated with a given solution can be evaluated in a polynomial time. The remaining proof is done by reducing the well-known set cover problem to the CMCM_D problem.

The set cover problem

INSTANCE: Given a finite set $E = \{e_1, e_2, ..., e_m\}$, and a collection of subsets $S = \{s_1, s_2, ..., s_n\}$. QUESTION: Is there a set cover for E, i.e., $S' \subseteq S$, and $|S'| \leq k$, such that every elements in E belongings to at least one of S'?

We now describe the reduction from the set cover problem to an instance of the CMCM_D problem. As shown in Fig. 4, we create a cloud for each subset in S, and a job with only map tasks for each element in E. All jobs target on a single data block that is stored in the cloud S_{n+1}. Each cloud $i (1 \leq i \leq n)$ is connected with the cloud $n+1$ by a direct link with transmission cost of 1 for a data block. If $e_i \in s_j$, the cost of processing a data block at cloud s_j for job e_i is set to 1. Otherwise, the cost is set to $m+k$. Finally, we set $C = m+k$. In the following, we only need to show that the set cover problem has a solution if and only if the resulting instance of CMCM_D problem has a solution that satisfies total cost constraint.

First, we suppose a solution of the set cover problem that all elements can be covered by at most k sets.

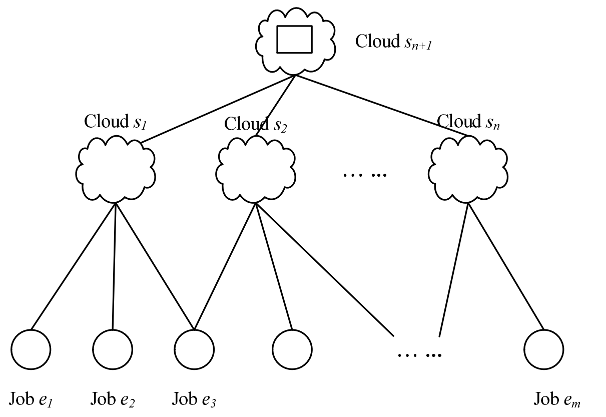

Fig. 4. The constructed CMCM_D instance.

The corresponding solution of the CMCM_D problem is to deliver the data block to the associated clouds, leading to transmission cost k. Since each job will process the data block in its own virtual cluster, the computation cost of all jobs is m. Thus, the total cost is no greater than $C = m + k$.

We then suppose that the CMCM_D problem has a solution with cost no greater than $C = m + k$. Since the job should be executed at the cloud where its processing cost is 1 (otherwise, the total cost will be greater than C), the job placement forms a set cover of elements in E, and total computation cost is m. Meanwhile, the traffic cost should be less than k, which can be achieved only by loading the data block to at most k clouds. Therefore, the corresponding solution in set cover problem guarantees that all elements are covered by at most k subsets. □

6.2 Problem Formulation

6.2.1 Computation cost

An extreme case of cross-cloud virtual cluster is that all data blocks are processed by local map tasks. Unfortunately, it is not always optimal or even feasible due to price diversity and capacity constraints of clouds. To describe remote input data loading, we define a binary variable $x_{ij}^{uv}(i, j \in N, b_i^u \in B_i, v \in V)$ as follows:

$$x_{ij}^{uv} = \begin{cases} 1, & \text{if } b_i^u \text{ is loaded to cloud } j \text{ for job } v, \\ 0, & \text{otherwise.} \end{cases}$$

Let α_i^{uv} denote the distribution of input data for job v:

$$\alpha_i^{uv} = \begin{cases} 1, & \text{if job } v \text{ targets on data block } b_i^u, \\ 0, & \text{otherwise.} \end{cases}$$

If any data block b_i^u is needed by job v, i.e., $\alpha_i^{uv} = 1$, it should be loaded into the corresponding virtual cluster that may spread on multiple clouds, i.e., exact one $x_{ij}^{uv}(j \in N)$ equal to 1. Otherwise, this data block will not be loaded, i.e., $x_{ij}^{uv} = 0, \forall j \in N$. These are summarized by:

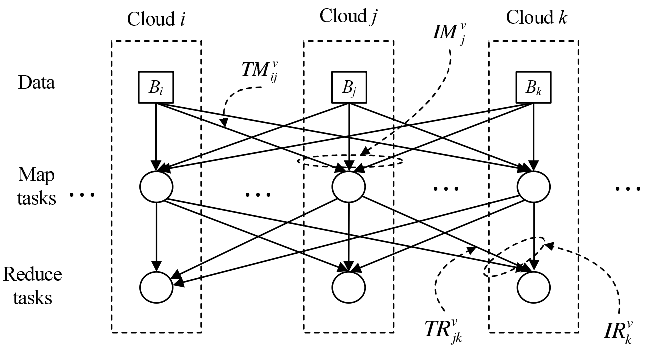

Fig. 5. Illustration of cross-cloud MapReduce.

$$\sum_{j \in N} x_{ij}^{uv} = \alpha_i^{uv}, \forall i \in N, \forall b_i^u \in B_i, \forall v \in V. \tag{1}$$

The amount of input data of job v loaded over link (i, j), denoted by TM_{ij}^v as shown in Fig. 5, can be calculated by:

$$TM_{ij}^v = \sum_{b_i^u \in B_i} |b_i^u| x_{ij}^{uv}, \forall i, j \in N, \forall v \in V. \tag{2}$$

By summing up the data portion from the storage infrastructures of all clouds, the data loaded into virtual cluster of job v at cloud j, denoted by IM_j^v, can be calculated by:

$$IM_j^v = \sum_{i \in N} TM_{ij}^v = \sum_{i \in N} \sum_{b_i^u \in B_i} |b_i^u| x_{ij}^{uv}, \forall j \in N, \forall v \in V. \tag{3}$$

As the amount of intermediated results generated by map tasks are approximately proportional to their input data size, we use a constant δ^v to denote their ratio that can be measured by profile running before job execution. The amount of output data generated by map tasks of each job v at cloud j, denoted as OM_j^v, is therefore estimated by:

$$OM_j^v = \delta^v IM_j^v, \forall j \in N, \forall v \in V. \tag{4}$$

We define a variable $y_k^v \in [0, 1]$ as the fraction of reduce tasks to be executed at cloud k, and the sum of y_k^v over all clouds should be equal to 1, i.e.,

$$\sum_{k \in N} y_k^v = 1, \forall v \in V. \tag{5}$$

The amount of intermediate data of job v delivered over link (j, k), denoted by TR_{jk}^v as shown in Fig. 5, can be expressed as:

$$TR_{jk}^v = OM_j^v y_k^v = \sum_{i \in N} \sum_{b_i^u \in B_i} \delta^v |b_i^u| x_{ij}^{uv} y_k^v, \forall j, k \in N, \tag{6}$$

since output data of map tasks are uniformly distributed among reduce tasks [18]. Finally, let IR_k^v be the total volumes of data fed to reduce tasks of job v at cloud k, i.e.,

$$\begin{aligned} IR_k^v &= \sum_{j\in N} TR_{jk}^v, \\ &= \sum_{i,j\in N}\sum_{b_i^u\in B_i} \delta^v |b_i^u| x_{ij}^{uv} y_k^v, \forall k\in N, \forall v\in V. \quad (7)\end{aligned}$$

The total computation cost can be calculated by summing up cost of both map and reduce tasks as:

$$\begin{aligned} C_{cc} &= \sum_{v\in V}\Big(\sum_{j\in N} p_j^v IM_j^v + \sum_{k\in N} q_k^v IR_k^v\Big) \\ &= \sum_{v\in V}\sum_{i,j,k\in N}\sum_{b_i^u\in B_i} |b_i^u|(p_j^v x_{ij}^{uv} + \delta^v q_k^v x_{ij}^{uv} y_k^v) \quad (8)\end{aligned}$$

6.2.2 *Traffic cost*

Our big data-centric architecture incurs two kinds of traffic. One is the traffic within cloud because of loading data from storage (e.g., Amazon S3) to the virtual cluster before processing, and writing results back to the storage after processing. The other traffic is the data transmission among clouds, which involves input data of map tasks, and intermediate results fed to reduce tasks. Compared with the high-speed network within cloud, cross-cloud network usually suffers from limited and dynamic bandwidth, and becomes the new bottleneck of geo-distributed applications [3], [4]. Based on this observation, we focus on minimizing the traffic across clouds.

To fully exploit the inter-job data characteristics, we consider the input data loading problem from a block-level perspective. When a data block is targeted by multiple jobs, it will be loaded into the corresponding virtual clusters that may reside in different clouds, naturally forming a multicast session with destinations that cannot be determined in advance.

Let $f_{ij}^{uv}(k,l)$ denote the flow of block b_i^u requested by job v from source cloud i to destination cloud j over link $(k,l)\in A$. We have the following constraints for flow conservation:

$$\sum_{l:(k,l)\in A} f_{ij}^{uv}(k,l) - \sum_{l:(l,k)\in A} f_{ij}^{uv}(l,k) = \\ |b_i^u| x_{ij}^{uv}\beta_{ij}(k,l), \forall i,j,k\in N, b_i^u\in B_i, v\in V, \quad (9)$$

where $\beta_{ij}(m,n)$ is an indicator that can be expressed as:

$$\beta_{ij}(k,l) = \begin{cases} 1, & \text{if } k=i, \\ -1, & \text{if } k=j, \\ 0, & \text{otherwise.}\end{cases}$$

Since the data blocks loaded to a cloud will be shared by all jobs residing in the same cloud, the actual traffic of delivering data volume $|b_i^u|$ is:

$$f_{ij}^u(k,l) = \max_{v\in V}\{f_{ij}^{uv}(k,l)\}, \\ \forall i,j\in N, \forall b_i^u\in B_i, \forall (k,l)\in A. \quad (10)$$

As we know, packets injected into the network would go through multiple paths to destinations and some paths leading to different destinations would be overlapped. An important theoretical discovery of network coding in [6] is that coded packets delivered on overlapped links for each destination do not contend with each other. Let $h_i^u(k,l)$ denote the actual flow on b_i^u over link (k,l), which can be calculated by:

$$h_i^u(k,l) = \max_{j\in N}\{f_{ij}^u(k,l)\} = \max_{j\in N, v\in V}\{f_{ij}^{uv}(k,l)\}, \\ \forall i\in N, \forall b_i^u\in B_i, \forall (k,l)\in A. \quad (11)$$

We then consider the dissemination of intermediate results among clouds. Recall that the set of jobs considered in our model are of different types, even though their map tasks have the same input data. Therefore, the generated intermediate results will be different in general. The resulting routing becomes a traditional flow problem with no NC involved. Let $g_{ij}^v(k,l)$ denote the flow of job v's intermediate results from source cloud j to destination cloud k over link $(k,l)\in A$. The constraints for flow of intermediate results can be summarized as:

$$\sum_{l:(k,l)\in A} g_{ij}^v(k,l) - \sum_{l:(l,k)\in A} g_{ij}^v(l,k) = TR_{ij}^v\beta_{ij}(k,l), \\ \forall i,j,k\in N, \forall v\in V. \quad (12)$$

Summing up the cost of delivering both input data and intermediate results, the total transmission cost can be calculated by:

$$C_{tc} = \sum_{i,j\in N}\sum_{(k,l)\in A} a_{kl}\Big[\sum_{b_u\in B_i} h_i^u(k,l) + \sum_{v\in V} g_{ij}^v(k,l)\Big]. \quad (13)$$

6.2.3 *Formulation*

With the objective of minimizing total cost, the problem can be formulated by:

$$\min C_{cc} + C_{tc} \qquad \text{subject to:} \\ f_{ij}^{uv}(k,l) \le h_i^u(k,l), \forall i,j\in N, b_i^u\in B_i, (k,l)\in A, \quad (14) \\ (1), (5), (9), \text{and } (12).$$

Note that constraint (14) is equivalent to (11) in a linear form. We observe that above formulation is still nonlinear because of the products of variables in (8) and (12). We define a new variable z_{ijk} as:

$$z_{ijk}^{uv} = x_{ij}^{uv} y_k^v, \forall i,j,k\in N, \quad (15)$$

such that (8) and (12) become:

$$C'_{cc} = \sum_{v\in V}\sum_{i,j,k\in N}\sum_{b_i^u\in B_i} |b_i^u|(p_j^v x_{ij}^{uv} + \delta^v q_k^v z_{ijk}^{uv}). \quad (16)$$

$$\sum_{l:(m,l)\in A} g_{jk}^v(m,l) - \sum_{l:(l,m)\in A} g_{jk}^v(l,m) = \\ \sum_{i\in N}\sum_{b_i^u\in B_i} \delta^v |b_i^u| z_{ijk}^{uv}\beta_{jk}(m,l), \forall i,j,k\in N, \forall v\in V. \quad (17)$$

respectively, in linear forms. Furthermore, (15) can be linearized as:

$$0 \le z_{ijk}^{uv} \le x_{ij}^{uv}, \forall i,j,k \in N, b_i^u \in B_i, v \in V \quad (18)$$

$$y_k^v + x_{ij}^{uv} - 1 \le z_{ijk}^{uv} \le y_k^v, \forall i,j,k \in N, b_i^u \in B_i, v \in V. \quad (19)$$

The equivalence holds for the following reasons. When $x_{ij}^{uv} = 1$, both constraints (15) and (19) become $z_{ijk}^{uv} = y_k^v$, and (18) is redundant. When $x_{ij}^{uv} = 0$, both (15) and (18) become $z_{ijk}^{uv} = 0$, and (19) is redundant. Finally, we obtain a new formulation written as follows:

CMCM: $\min C'_{cc} + C_{tc}$ subject to:

$$(1), (5), (9), (14), \text{and } (17) - (19).$$

Although the above formulation is a mixed integer linear programming (MILP), there exist highly efficient approximation algorithms, e.g., branch-and-bound, and fast off-the-shelf solvers, e.g., CPLEX. Since our focus in the paper is to develop an optimization framework of the CMCM problem for big data applications, we omit the details of solving MILP due to space limit.

7 PARALLEL ALGORITHM

Although commercial solvers, e.g., CPLEX, are available for our formulation, they hardly support large-scale problems with lots of jobs targeting on big data. In this section, we develop a parallel algorithm for the CMCM problem by decomposing the original large-scale problem into several distributively solvable subproblems that are coordinated by a high-level master problem.

We first consider to decompose the original problem into several subproblems, each of which handles the computation and traffic routing of a single job. By carefully analyzing the structure of our formulation, we find that the main difficulty in such decomposition comes from the coupling variable $h_i^u(k,l)$. Fortunately, when the variable $h_i^u(k,l)$ is fixed to a value $\hat{h}_i^u(k,l)$, the rest of optimization problem can decouple into several subproblems. Based on this observation, we have the following subproblem for each job $v \in V$:

SUB1_JOB : $\min \mathcal{C}$ subject to:

$$f_{ij}^{uv}(k,l) \le \hat{h}_i^u(k,l), \forall i,j \in N, b_i^u \in B_i, (k,l) \in A, \quad (20)$$

$$(1), (5), (9), \text{and } (17) - (19).$$

where x_{ij}^{uv}, y_k^v, z_{ijk}^{uv}, $f_{ij}^{uv}(k,l)$, and $g_{ij}^v(k,l)$ are variables, and the objective function can be written as:

$$\mathcal{C} = \sum_{i,j,k \in N} \Big[\sum_{b_i^u \in B_i} |b_i^u| (p_j^v x_{ij}^{uv} + \delta^v q_k^v z_{ijk}^{uv}) + \sum_{(m,l) \in A} a_{ml} \Big(g_{ij}^v(m,l) + \sum_{b_i^u \in B_i} \hat{h}_i^u(m,l) \Big) \Big]. \quad (21)$$

At the higher level, we have the master problem in charge of updating the coupling variable $h_i^u(k,l)$ by solving:

$$\min \mathcal{C}\big(h_i^u(k,l)\big) \quad (22)$$

which can be solved with a gradient method. The objective function $\mathcal{C}\big(h_i^u(k,l)\big)$ is in a similar form of (21) except treating $h_i^u(k,l)$ as the variable and others as known constants by solving subproblems.

When we deal with big data of a large number of data blocks, each subproblem is still in large-scale. We then consider to further decompose each subproblem for a single job $v \in V$ by introducing an auxiliary variable y_k^{uv} such that each subproblem can be equivalently formulated as:

$$\min \mathcal{C} \quad \text{subject to:}$$

$$y_{ik}^{uv} + x_{ij}^{uv} - 1 \le z_{ijk}^{uv} \le y_{ik}^{vu}, \forall i,j,k \in N, b_i^u \in B_i; \quad (23)$$

$$y_{ik}^{uv} = y_k^v, \forall i,k \in N, b_i^u \in B_i; \quad (24)$$

$$(1), (5), (9), (17), (18), \text{and } (20).$$

The corresponding Lagrangian function is as follows:

$$\begin{aligned}
\mathcal{L} &= \mathcal{C} + \lambda_{ik}^{uv}(y_k^v - y_{ik}^{uv}) \\
&= \sum_{i,j,k \in N} \Big[\sum_{b_i^u \in B_i} |b_i^u| (p_j^v x_{ij}^{uv} + \delta^v q_k^v z_{ijk}^{uv}) \\
&\quad + \sum_{(m,l) \in A} a_{ml} \Big(g_{ij}^v(m,l) + \sum_{b_i^u \in B_i} \hat{h}_i^u(m,l) \Big) \Big] \\
&\quad + \sum_{i,k \in N} \sum_{b_i^u \in B_i} \lambda_{ik}^{uv}(y_k^v - y_{ik}^{uv}) \\
&= \sum_{i,j,k \in N} \sum_{b_i^u \in B_i} \Big[|b_i^u| (p_j^v x_{ij}^{uv} + \delta^v q_k^v z_{ijk}^{uv}) \\
&\quad + \sum_{(m,l) \in A} a_{ml} \Big(g_{ijk}^{vu}(m,l) + \sum_{b_i^u \in B_i} \hat{h}_i^u(m,l) \Big) \\
&\quad - \lambda_{ik}^{uv} y_{ik}^{uv} \Big] + \sum_{i,k \in N} \sum_{b_i^u \in B_i} \lambda_{ik}^{uv} y_k^v. \quad (25)
\end{aligned}$$

Note that we denote $g_{jk}^v = \sum_{i \in N}^{b_i \in B_i} g_{ijk}^{uv}$ in above equation. Given λ_{ik}^{uv}, the dual decomposition results in two sets of subproblems: global scheduling and reducer placement. The subproblem of global scheduling for each data block b_i^u is as follows:

SUB2_GS_DB : $\max \sum_{j,k \in N} \Big[|b_i^u| (p_j^v x_{ij}^{uv} + \delta^v q_k^v z_{ijk}^{uv})$

$$+ \sum_{(m,l) \in A} a_{ml} \Big(g_{ijk}^{vu}(m,l) + \sum_{b_i^u \in B_i} \hat{h}_i^u(m,l) \Big) - \lambda_{ik}^{uv} y_{ik}^{uv} \Big]$$

$$f_{ij}^{uv}(k,l) \le \hat{h}_i^u(k,l), \forall j \in N, (k,l) \in A, \quad (26)$$

$$(1), (9), (17), (18), (20), \text{and } (23).$$

which can be distributed and solved on multiple machines in a parallel manner. The subproblem of

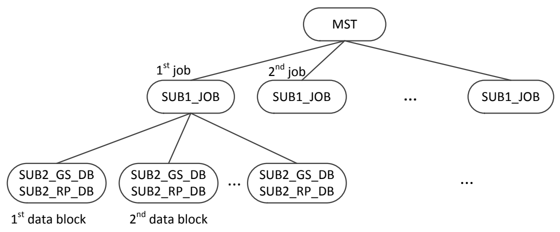

Fig. 6. A hierarchical decomposition process.

reducer placement can be simply written as:

$$\textbf{SUB2\_RP\_DB} : \max \sum_{k \in N} \lambda_{ik}^{uv} y_k^v, \text{subject to:}$$
$$\sum_{k \in N} y_k^v = 1, \forall v \in V. \quad (27)$$

The values of λ_{ik}^{uv} are updated in the following master problem:

$$\max \mathcal{L} = \hat{\mathcal{C}} + \lambda_{ik}^{uv}(\hat{y}_k^v - \hat{y}_{ik}^{uv}), \text{subject to:}$$
$$\lambda_{ik}^{uv} \geq 0, \forall i, k \in N, b_i^u \in B_i, v \in V, \quad (28)$$

where $\hat{\mathcal{C}}$, $\hat{y}_k^v$ and $\hat{y}_{ik}^{uv}$ are optimal solutions returned by subproblems. Since the objective function of the master problem is differentiable, it can be solved by the following gradient method:

$$\lambda_{ik}^{uv}(t+1) = \left[\lambda_{ik}^{uv}(t) + \xi\big(\hat{y}_k^v(\lambda_{ik}^{uv}(t)) - \hat{y}_{ik}^{uv}(\lambda_{ik}^{uv}(t))\big)\right]^+ \quad (29)$$

where t is the iteration index, ξ is a positive stepsize, and '+' denotes the projection onto the nonnegative orthant.

In summary, we finally have a hierarchical decomposition as illustrated in Fig. 6, and pseudo codes of corresponding parallel algorithm is shown in Algorithm 1. In this algorithm, we iteratively approximate the optimal solution by first decomposing the original problem into several subproblems **SUB1_JOB** that handle cost minimization of each job, respectively. The total number of iterations in the first-level decomposition is T_1 and t_1 is used as an iteration counter. Then, each subproblem **SUB1_JOB** can be further decomposed into subsubproblems **SUB2_GS_DB** and **SUB2_RP_DB** that handle the computation and routing of data blocks. After solving these subproblems, we update the values of λ_{ik}^{uv} according to (29), and send the results to all subproblems for next-round optimization. Similarly, we use T_2 and t_2 to denote the total number of iterations and an counter, respectively, for the second-level decomposition. After collecting results from all subprolems **SUB1_JOB** in the first-level decompostion, we update the values of $h_i^u(m,l)$ and distribute them for next-round optimization.

Algorithm 1 Parallel Algorithm

```
1:  set $t1 = 1$, and $h_i^u(m,l)$ to arbitrary nonnegative values that are large enough.
2:  for $t1 < T1$ do
3:      set $t2 = 1$, and $\lambda_{ik}^{uv}$ to arbitrary nonnegative values;
4:      for $t2 < T2$ do
5:          distributively solve the subproblem SUB2_GS_DB and SUB2_RP_DB on multiple machines in a parallel manner;
6:          update the values of $\lambda_{ik}^{uv}$ with the gradient method, and send the results to all subproblems SUB2_GS_DB and SUB2_RP_DB;
7:          set $t2 = t2 + 1$;
8:      end for
9:      update the values of $h_i^u(m,l)$ and send the results to all subproblems SUB1_JOB;
10:     set $t1 = t1 + 1$;
11: end for
```

8 Performance Evaluation

In this section, we first evaluate the performance of our proposed algorithm for single job. Then, extensive simulations are conducted to evaluate the performance for multiple jobs with various simulation settings. The optimal solution under our proposed scheme is denoted by OPT in this section. For comparison, we also show the results of the following two approaches.

Simple Data Aggregation (SDA): For each job, its virtual cluster is constrained within a single cloud, to which the distributed input data should be aggregated. For each job, we calculate the total cost at every cloud respectively, and then select the best one with minimum cost.

Cross-cloud Virtual Cluster (CVC): Only intra-job data characteristic is exploited in this algorithm, in which any common data block will be loaded by unicast to each of its associated jobs across clouds. It can be implemented by replacing the first term in constraint (20) with $\sum_{i,j \in N} \sum_{b_i^u \in B_i} \sum_{(k,l) \in A} a_{kl} f_{ij}^u(k,l)$.

All results in the following are averaged over 20 random problem instances.

8.1 Single job

We consider the execution traces of 5 MapReduce jobs collected from Facebook cluster [27]. These jobs include data aggregation, expansion, and transformation, and their input data, intermediate data, and execution time is shown in Table. 2. The unit cost for processing one GB of data is set within [0.45,0.55] dollars/hour according to typical charges in Amazon EC2 [4], [25]. We consider a typical cloud provider with 8 data centers (e.g., Amazon EC2 consists of 8 data centers), and input data of each job are randomly distributed on them. The communication cost between two clouds is \$0.01 per GB [25].

The total cost of each job, in the form of log scale, is shown in Fig. 7. Compared with traditional sin-

TABLE 2
Experimental results

Job #	Job description	Input	Shuffle	Output	Duration
1	Aggregate, fast	230GB	8.8GB	491MB	15 min
2	Aggregate and expand	1.9TB	502MB	2.6GB	30 min
3	Expand and aggregate	419GB	2.5TB	45GB	1 hr 25 min
4	Data transform	255GB	788GB	1.6GB	35 min
5	Data summary	7.6TB	51GB	104KB	55 min

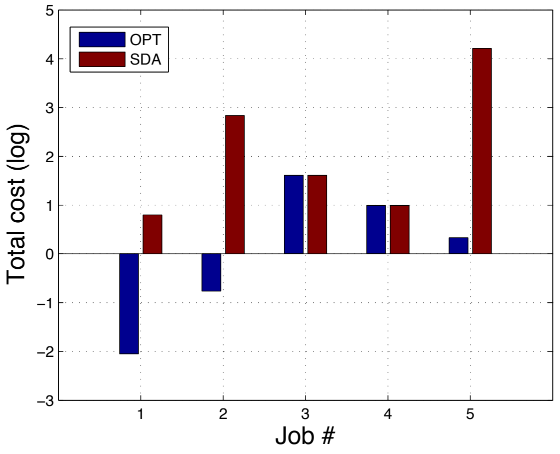

Fig. 7. The results of trace-driven experiments.

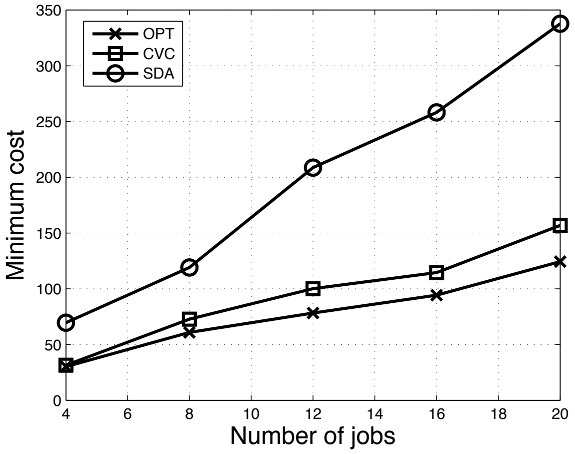

Fig. 8. The minimum cost versus different numbers of jobs.

gle virtual cluster, our proposed cross-cloud virtual cluster can reduce 94.1%, 97.3%, and 97.9% total cost for jobs 1, 2, and 5, respectively. That is because our proposed cross-cloud virtual cluster avoids inter-cloud data transmission of huge input data. For jobs 3 and 4 whose input data is less than intermediate data, our proposal has the same performance with single virtual cluster because it is better to first aggregate small input data and then process them at the cloud with the lowest processing cost.

8.2 Multiple jobs

In this subsection, we first present a set of default simulation settings and then study the performance of all approaches by changing one of them each time. In the default settings, we consider 20 MapReduce jobs in a geo-distributed cloud environment of 12 nodes. We consider the *commonIP* data distribution trace [30], in which data are placed as close as possible to the IP address the most commonly accesses it. The computation and transmission cost for unit data, i.e., p_i^v, q_i^v and a_{ij}, is specified as the same with the settings in last subsection. We ignore the cost of intra-cloud data loading by setting $a_{ii} = 0, \forall i \in N$ because it is much less than that of inter-cloud communication. Half of jobs belongs to type I, whose values of δ^v are uniformly distributed within $[0.001, 1]$, while other jobs' are within $[1, 5]$. Each job chooses a data block as input with a probability of 0.5.

We first study the minimum cost under different number of jobs in 12-node networks. As shown in Fig. 8, the cost of all approaches grows as the number of jobs increases from 4 to 20. This is because more data need to be processed under larger number of jobs, leading to an increased computation and transmission cost. We also observe that the performance gap between SDA and the other two algorithms becomes larger as the number of jobs grows.

We then investigate the influence of job types on total cost. As shown in Fig. 9, the minimum cost of all algorithms decreases as the number of type-I jobs increases from 2 to 18 because such jobs generate small intermediate data that result in less transmission and computation cost of reduce tasks. The performance of three approaches is close when the number of type-I jobs is 2 since they all prefer to aggregate input data to avoid delivering large amount of intermediate results. As the number grows to 18, CVC and OPT outperform SDA significantly by saving 70% and 76% cost.

The cost under different means of transmission cost per unit data is shown in Fig. 10. The performance gaps between OPT and others become larger as the mean increases because more transmission cost can be reduced due to NC-based traffic routing.

We change the probability of choosing data blocks as input from 0.1 to 0.9 and show the resulting performance ratio of OPT/CVC and OPT/SDA in Fig. 11. The larger the probability is set to, the more

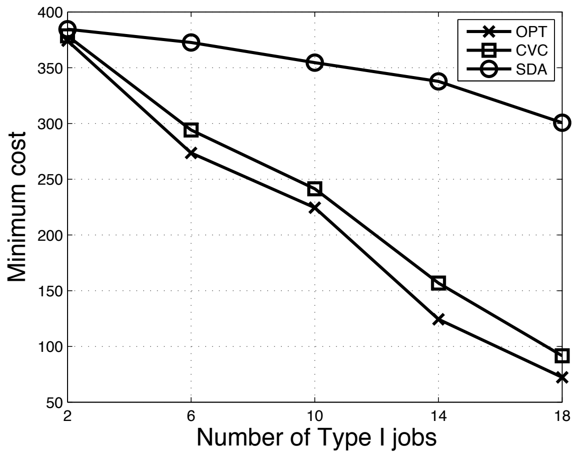

Fig. 9. The minimum cost versus different numbers of Type I jobs.

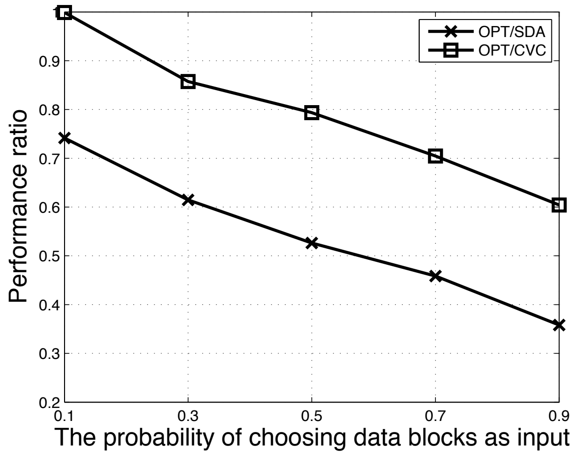

Fig. 11. The performance ratio versus different probabilities of choosing data blocks as input.

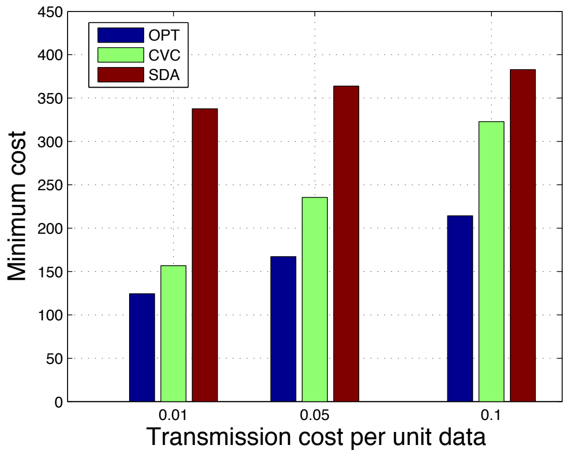

Fig. 10. The minimum cost versus different means of transmission cost per unit data.

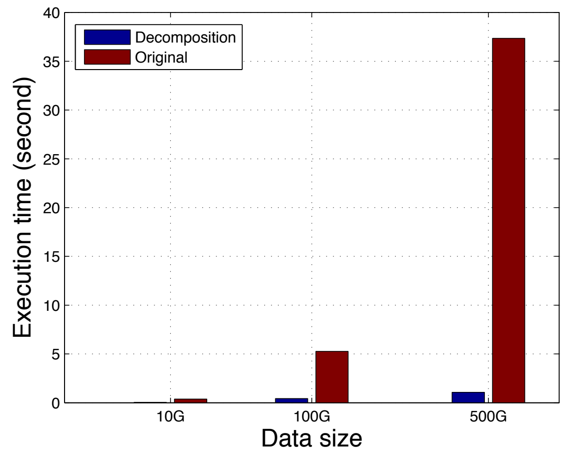

Fig. 12. The execution time of our original formulation and decomposition method

data are shared among jobs. When the probability is 0.1, the performance ratio of OPT/CVC is close to 1, which indicate that there is nearly no opportunity of exploiting inter-job data characteristics. On the other hand, OPT still outperforms SDA with 25% cost reduction due to cross-cloud virtual cluster.

Finally, we study the time efficiency of our proposed decomposition method by comparing its execution time with our original formulation. As shown in Fig. 12, our decomposition method always finishes within 1 seconds, while the execution time of our original formulation increases significantly as the growth of data size.

9 Conclusion

In this paper, we consider a geo-distributed cloud architecture to support MapReduce jobs of global applications. To reduce the cost of running a set of MapReduce jobs, we propose three key techniques, namely, cross-cloud virtual cluster, data-centric job placement, and NC-based traffic routing. We prove that the cost minimization problem is NP-hard by reducing from well-known bin-packing problem. To solve the problem, we propose an optimization framework that takes our proposed three key techniques into consideration. Finally, real-world experiments and extensive simulations are conducted to show the advantages of our proposals.

References

[1] *Hadoop http://hadoop.apache.org.*
[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proc. of OSDI*, San Francisco, CA, 2004, pp. 1–10.
[3] S. Jain, A. Kumar, S. Mandal, and etc., "B4: experience with globally-deployed software defined wan," in *Proc. of ACM SIGCOMM*, Hong Kong, China, August 2013, pp. 1–12.
[4] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. Lau, "Moving big data to the cloud," in *Proc. of IEEE INFOCOM*, April 2013, pp. 405–409.

[5] *http://hadoopblog.blogspot.jp/2010/05/facebook-has-worlds-largest-hadoop.html*.
[6] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.
[7] M. Cardosa, A. Singh, H. Pucha, and A. Chandra, "Exploiting spatio-temporal tradeoffs for energy-aware mapreduce in the cloud," *IEEE Transactions on Computers*, vol. 61, no. 12, pp. 1737–1751, 2012.
[8] S. Sehrish, G. Mackey, P. Shang, J. Wang, and J. Bent, "Supporting hpc analytics applications with access patterns using data restructuring and data-centric scheduling techniques in mapreduce," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 158–169, 2013.
[9] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguadé, "Resource-aware adaptive scheduling for mapreduce clusters," in *Proc. of ACM/IFIP/USENIX international conference on Middleware*, 2011, pp. 187–207.
[10] B. Sharma, R. Prabhakar, S. Lim, M. Kandemir, and C. Das, "Mrorchestrator: A fine-grained resource orchestration framework for mapreduce clusters," in *Proc. of IEEE CLOUD*, 2012, pp. 1–8.
[11] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55, Apr 2009.
[12] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: a platform for fine-grained resource sharing in the data center," in *Proc. of USENIX NSDI*, Boston, MA, 2011, pp. 22–22.
[13] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map task scheduling in mapreduce with data lacality: throughput and heavy-traffic optimality," in *Proc. of IEEE INFOCOM*, 2013, pp. 1657–1665.
[14] J. Tan, S. Meng, X. Meng, and L. Zhang, "Improving reducetask data locality for sequential mapreduce jobs," in *Proc. of IEEE INFOCOM*, 2013, pp. 1675–1683.
[15] *Amazon MapReduce http://aws.amazon.com/elasticmapreduce/*.
[16] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "Mapreduce in the clouds for science," in *Proc. of IEEE CloudCom*, 2010, pp. 565–572.
[17] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, Boston, MA, 2010, pp. 1–7.
[18] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality-aware resource allocation for mapreduce in a cloud," in *Proc. of SC*, Seattle, Washington, 2011, pp. 1–11.
[19] H. Kang, Y. Chen, J. L. Wong, R. Sion, and J. Wu, "Enhancement of xen's scheduler for mapreduce workloads," in *Proc. of the 20th international symposium on High performance distributed computing*, 2011, pp. 251–262.
[20] B. Sharma, T. Wood, and C. R. Das, "Hybridmr: a hierarchical mapreduce scheduler for hybrid data centers," in *Proc. of IEEE ICDCS*, 2013, pp. 102–111.
[21] M. Cardosa, C. Wang, A. Nangia, A. Chandra, and J. Weissman, "Exploring mapreduce efficiency with highly-distributed data," in *Proceedings of the Second International Workshop on MapReduce and Its Applications*, San Jose, California, USA, 2011, pp. 27–34.
[22] A. Iordache, C. Morin, N. Parlavantzas, E. Feller, and P. Riteau, "Resilin: Elastic mapreduce over multiple clouds," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2013, pp. 261–268.
[23] A. Mandal, Y. Xin, I. Baldine, P. Ruth, C. Heerman, J. Chase, V. Orlikowski, and A. Yumerefendi, "Provisioning and evaluating multi-domain networked clouds for hadoop-based applications," in *IEEE Cloud Computing Technology and Science (CloudCom)*, Nov 2011, pp. 690–697.
[24] S. Kailasam, P. Dhawalia, S. Balaji, G. Iyer, and J. Dharanipragada, "Extending mapreduce across clouds with bstream," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 362–376, July 2014.
[25] *Amazon EC2: http://aws.amazon.com/ec2*.
[26] *https://github.com/SWIMProjectUCB/SWIM/wiki*.
[27] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, Aug. 2012.
[28] *Diseases and Conditions of Pneumonia: http://www.mayoclinic.org/diseases-conditions/pneumonia/basics/tests-diagnosis/con-20020032*.
[29] *Diseases and Conditions of Appendicitis: http://www.mayoclinic.org/diseases-conditions/appendicitis/basics/tests-diagnosis/con-20023582*.
[30] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proc. of the 7th USENIX Conference on Networked Systems Design and Implementation*, 2010, pp. 1–16.