Geo-Distributed IoT Data Analytics With Deadline Constraints Across Network Edge

Yiting Chen, Lailong Luo, *Member, IEEE*, Bangbang Ren, and Deke Guo, *Senior Member, IEEE*

***Abstract*—Owing to the advancement of the Internet of Things (IoT) and 5G mobile technologies, various IoT devices produce massive data, which is usually transferred to nearby sites, such as edge nodes or datacenters. Many large-scale IoT applications need to analyze the data distributed across multiple sites to obtain final results. A dominant challenge of this type of data analytics is the heterogeneities of resource capacities across geo-distributed sites. In this article, we find that the resource capacity as well as the resource price differ among sites, and the price heterogeneity has a significant impact on geo-distributed IoT data analytics. Thus, each geo-distributed IoT data analytics job prefers to minimize the job execution cost while guaranteeing its deadline requirement under the resource constraints of involved sites. Specifically, we propose to jointly consider the resource heterogeneities of both capacity and price, and minimize the cost of each job before its deadline. We characterize this optimization problem as a quadratically constrained quadratic programming problem. To tackle such an NP-hard problem, we propose the minimize the job completion cost before a given deadline (MCGL) method, which calculates a task placement solution by the gradient adjustment strategy according to the remarkable negative correlation relationship between job completion time and job completion cost of geo-distributed IoT data analytics job. The task placement strategy can optimize resource cost with respect to the deadline requirement of any geo-distributed data analytics job. The trace-driven evaluations indicate that MCGL significantly reduces the total cost compared with existing methods; moreover, they satisfy the deadline constraints simultaneously.**

***Index Terms*—Deadline constraint, geo-distributed Internet of Things (IoT) data analytics, IoT, price heterogeneity.**

I. Introduction

WITH the enormous advancement of the Internet of Things (IoT) and 5G mobile technologies, many applications, such as mobile augmented reality and virtual reality, Internet of vehicles, and automatic driving, have developed rapidly [1]. As shown in Fig. 1, various IoT devices continuously generate data worldwide. These huge amounts of data are usually transferred to nearby edge clusters, which can provide services with low latency and save the scarce WAN bandwidth. Due to the resource constraints of edge clusters, historical IoT data of edge clusters will be periodically transferred to cloud datacenter clusters. Many IoT applications need to analyze data distributed across multiple sites to extract useful information. For example, analyzing the environmental monitoring data collected in various places to provide data support for the solution of pollution control; and analyzing the electricity consumption behavior data collected in various regions to provide help for the pricing and distribution decisions of the state grid. However, this would lead to significant consumption of geo-distributed network resources to obtain those large-scale data from all involved sites; moreover, it demonstrates a high latency toward the completion of such jobs. Thus, many efforts have been made to improve the efficiency of these types of jobs by concurrently executing a data analytics job across multiple sites [2]–[5].

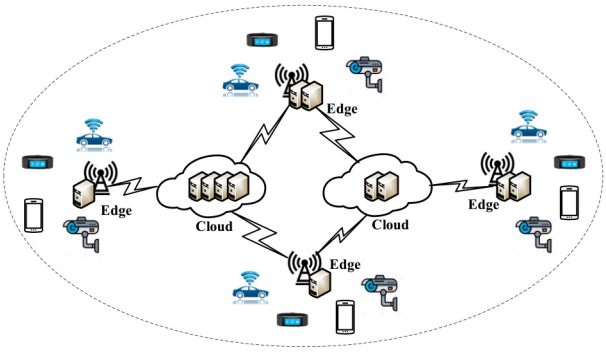

Fig. 1. Service providers deploy massive data centers and edge nodes to support various types of IoT applications.

These data analytics jobs are usually based on the MapReduce framework. A significant challenge of the geo-distributed MapReduce computing paradigm is the heterogeneity of hardware resource capacities among geo-distributed sites, including the computing, storage, uplink bandwidth, and downlink bandwidth. As reported in literature [3], the computation capacity of the largest online service provider can be up to two orders of magnitude larger than that of the ordinaries. Additionally, the gap between the bandwidth among Amazon EC2 sites is up to 12× [6]. Clearly, sites with sufficient resources can complete tasks scheduled to them faster than the sites with insufficient resources. Hardware capacities heterogeneity is an important factor for the job

Manuscript received 9 March 2022; revised 17 April 2022; accepted 16 June 2022. Date of publication 24 June 2022; date of current version 7 November 2022. This work was supported in part by the National Natural Science Foundation of China under Grant U19B2024, and in part by the Major Scientific Research Project of Zhejiang Lab under Grant 2021PE0AC01. *(Corresponding authors: Lailong Luo; Deke Guo.)*

Yiting Chen, Bangbang Ren, and Deke Guo are with the Science and Technology Laboratory on Information Systems Engineering, National University of Defense Technology, Changsha 410073, Hunan, China (e-mail: chenyiting18@nudt.edu.cn; renbangbang11@nudt.edu.cn; dekeguo@nudt.edu.cn).

Lailong Luo is with the College of Computer, National University of Defense Technology, Changsha 410073, Hunan, China (e-mail: luolailong09@nudt.edu.cn).

Digital Object Identifier 10.1109/JIOT.2022.3186173

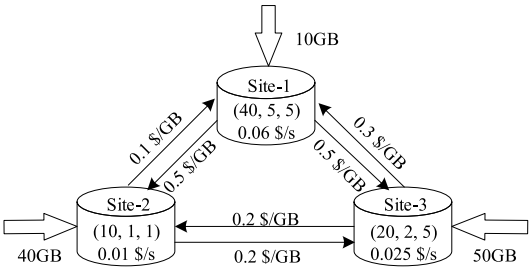

Fig. 2. Three geo-distributed sites offer heterogeneous hardware resources with unequal prices. The triple labeled on each site represents the available computing slots, the uplink bandwidth, and the downlink bandwidth, respectively. The single value labeled on sites or links counts the price of computing slots or bandwidth, respectively.

completion time (JCT) because in the traditional MapReduce framework, the execution time of each stage is decided by the bottleneck site. Many efforts have been made to optimize the JCT by carefully managing and scheduling the involved tasks, input data, and resources [2], [3], [6]–[8]. However, such methods just pursue the minimum average response latency of time-sensitive jobs while incurring a nontrivial cost.

Apart from the heterogeneity of hardware resource capacities, the heterogeneity of resource prices also significantly affects the task scheduling of a geo-distributed data analytic job, when the job completion cost (JCC) is considered. In fact, the price of intersite bandwidth (i.e., network resource) and the VM (i.e., computation resource) price at different sites are highly different, as reported in literatures [9]–[12]. For example, the highest bandwidth price is an order of magnitude higher than the cheapest one, and the prices of a unit computation capacity also vary significantly on the Amazon Website [13]. Furthermore, the charging standards for hardware resources of different cloud or edge service providers are also diverse. Thus, if the job executes tasks on expensive computing slots and transfers data through expensive links, the completion cost of this job will also be high. Recently, owing to the realization that the WAN bandwidth is very expensive, several studies have rather focused on reducing the intersite transmission cost in geo-distributed data analytics jobs [14]–[16]. However, those studies mostly assume that the prices of network and computing resources at geo-distributed sites are similar. Therefore, attempts should be geared toward minimizing the volume of transfer data solely [14]–[18].

In this article, we present a new abstraction to characterize the heterogeneities among geo-distributed sites, as shown in Fig. 2. The triple labeled on each site represents the available computing slots, the uplink bandwidth, and the downlink bandwidth, respectively. The single value labeled on sites or links represents the unit price of computing slots or bandwidth, respectively. For example, the triple (40, 5, 5) labeled on site 1 means that site 1 has 40 computing slots, and its uplink bandwidth and downlink bandwidth are both 5 GB/s. The 0.06 $/s labeled on site 1 represents the price of each computing slot per second on site 1. The 0.5 $/GB labeled on the arrow from site 1 to site 2 represents the price of transferring data from site 1 to site 2. This figure clearly demonstrates the heterogeneities of both the resource capacities and resource prices among different sites.

As mentioned above, existing methods mainly focus on solely optimizing either the JCT or JCC, and thus, they fail to balance the tradeoff between the global completion time and total cost. For this reason, we emphasize that both JCT and JCC need to be considered when processing a geo-distributed IoT data analytics job. Actually, not all geo-distributed data analytics jobs are required to be executed as soon as possible and obtain real-time results. There are many jobs that just need to be completed before a given deadline [19], [20], such as geo-distributed batch jobs [18] and geo-distributed scientific computations [21]. With this in mind, we propose to minimize the JCC of geo-distributed analytic jobs, which have certain deadline requirements. We characterize the task placement problem of a set of geo-distributed data analytics jobs, considering the heterogeneities of both the resource capacities and prices.

We prove that this problem is NP-hard and accordingly propose a heuristic algorithm called minimize the JCC before a given deadline (MCGL) to solve it efficiently. This method fully exploits the relation of the completion time and the completion cost of the geo-distributed IoT data analytics job under the MapReduce framework; it follows that there is a remarkable negative correlation between the JCT and the JCC. Thus, it first calculates two task placement strategies for each job via the MinCost and MinTime algorithms, which optimize the JCT and the JCC, respectively. Thereafter, based on the results, it adopts a heuristic method to figure out another task placement strategy, which achieves the minimal cost of the geo-distributed IoT data analytics job and satisfies its deadline constraint. Besides, the existing methods mainly consider the jobs with just two stages, making them not practical. To this end, we provide a more general method MCGL+. In MCGL+, it utilizes the task placement solutions of improved MinCost and MinTime algorithms, and calculates an optimal task placement solution to minimize the completion cost of multistage jobs under a given deadline.

The major contributions of this article can be summarized as follows.

1) We present a new abstraction to characterize the geo-distributed data IoT analytics jobs under the MapReduce framework by jointly considering both resource capacities heterogeneity and resource price heterogeneity. Furthermore, a general formulation is presented to minimize the total cost of a geo-distributed IoT data analytics job under a deadline constraint.
2) To tackle such an NP-hard problem, we propose an approximate method, MCGL, to derive out a suboptimal solution. It minimizes the completion cost of the geo-distributed IoT data analytics job while respecting to deadline requirements.
3) We further improve our MCGL method to tackle multistage jobs instead of prior two-stage jobs. A more general method MCGL+ is designed to minimize the JCC under a given deadline constraint.

TABLE I
SUMMARY OF GEO-DISTRIBUTED DATA ANALYTICS FRAMEWORKS AND ALGORITHMS

Objectives	Frameworks/ algorithms	Heterogeneities of capacity		Heterogeneities of price		Deadline constraint	MapReduce -based
		Network	Compute	Network	Compute		
Time	Iridium [2]	√					√
	Liu [23]						√
	Tetrium [3]	√	√				√
	Clarient [8], Gaia [7]	√					
	Lube [24]	√	√				
	Monarch [25]	√					
	HPS+ [22]	√	√				
	iStore [26]	√	√				
Cost	Yugong [19]	√					
	MiniBDP [12]	√	√	√	√		√
	Li [16], Pixida [15]	√					√
	Flutter [11]	√		√			√
	Geode [17],WANalytics [27]						
	Kimchi [28]	√		√			√
	MCGL	√	√	√	√	√	√

The remainder of this article is organized as follows. Section II depicts the related work. Section III presents the background and motivation examples. Section IV describes the system model and formulates the task placement problem of a geo-distributed IoT data analytic job. Section V presents our MCGL and MCGL+ methods. We conduct extensive evaluations using realistic traces in Section VI, and conclude this article with discussion on future work in Section VII.

II. RELATED WORK

Many efforts have been made to optimize the execution of a given geo-distributed data processing job, and Table I is the summary of the characteristics of these work. Prior methods mainly fall into the following two categories, i.e., shortening the JCT or saving the JCC.

Optimization of the JCT: In the first category, Liu *et al.* [22] proactively aggregated the output data of map tasks and avoided repetitive data transfers in the shuffle stage to reduce the JCT. However, it does not consider the effect of the heterogeneity of resources across geo-distributed sites. Iridium [2] considers the heterogeneity of the wide-area network, and achieves low latency by optimizing the placement of reduce tasks and involved input data. However, it ignores the effect of the computing resource. Tetrium [3] jointly considers the heterogeneity of computing in tandem with network resources in designing the placement strategy of the involved map and reduce tasks. To save the amount of data transmission and reduce the makespan, HPS+ [21] proposes a new resource allocation algorithm. Literature [25] provides a federation file system atop of geo-distributed edge servers, and proposes a generic job and resource-aware data storage and placement algorithm (JRAP). JRAP estimates the job execution time to map the request to the best edge server to minimize the job execution time; however, it only determines the number of the sites used to execute the job request; without considering the task scheduling of the job. Generally, the above methods mainly reduce the JCT, without considering the overall cost of wide-area data analytics.

There also exist some approaches to reduce the execution time of other types of geo-distributed data analytics jobs. Lube [23] monitors geo-distributed data analytics queries in real time, and detects and mitigates potential bottlenecks (e.g., bandwidth scarcity) at runtime to reduce the query response time. Clarient [7] proposes a novel WAN-aware query optimizer, which places tasks to sites and carefully schedules tasks to ensure fast query response. Gaia [6] and Monarch [24] accelerate the execution of geo-distributed machine learning jobs and geo-distributed graph analytics, respectively; however, they are all not applicable to MapReduce frameworks.

Optimization of the JCC: In the second category, because the intersite bandwidth is scarce and expensive, researchers propose to save the transmission cost by reducing the data movement across geo-distributed sites. WANalytics [26] and Geode [16] aim to reduce bandwidth usage across geo-distributed datacenters for query requests and data replication, respectively. For geo-distributed graph analytics, Pixida [14] formulates a new graph partitioning problem and proposes a method to minimize the data movement across bandwidth constrained links. To minimize cross-dc bandwidth usage, Yugong [18] proposes a novel data placement and job placement strategy in Alibaba's geo-distributed datacenters. However, the above methods do not consider the diversity of bandwidth cost across geo-distributed datacenters.

Considering the diversity of the intersite bandwidth price, Flutter [10] proposes a new task scheduling method to minimize the job response time of one stage considering the bandwidth budget constraints. Li *et al.* [15] devoted to minimizing the completion time and the average transmission cost of any coflow. Kimchi [27] proposes a cost-aware task placement decisions for scheduling tasks to avoid the cost bottleneck. However, they do not consider the price diversity of computing resources across different sites. Especially, MiniBDP [11] determines that bandwidth prices vary over different VPN links, and the VM price also changes over time and, thereafter, it characterizes a complex cost optimization problem to solve it. However, this method only focuses on minimizing the time-averaged operation cost of a long-term job and does not impose any deadline constraint on that job. In

TABLE II
PRICE OF COMPUTE, NETWORK, AND STORAGE OF SITES AT DIFFERENT REGIONS ON NOVEMBER 8, 2021

	Virginia(N)	Tokyo	Seoul	Frankfurt	Sao Paulo	Cape Town
Compute ($/hour)	0.1664	0.2176	0.208	0.192	0.2688	0.217
Network ($/GB)	0.02	0.09	0.08	0.02	0.138	0.147
Storage ($/GB/month)	0.025	0.03	0.029	0.03	0.048	0.028

addition, Bi *et al.* [28] tried to minimize the service provider's energy cost and maximize revenue in a single virtualized cloud data center, but it considers the resource provisioning in a single cloud data center and is applicable to geo-distributed data analytics jobs.

The aforementioned methods mainly focus on optimizing either the completion time or the transmission cost. In our method, we provide a better tradeoff between the JCT and total resource cost for computing a geo-distributed data analytics job. We aim to minimize the total resource cost resulting from executing such a job before a deadline, when considering the heterogeneity of the capacities and prices of hardware resources.

III. BACKGROUND AND MOTIVATION

To realize the geo-distributed data analytics, many data processing frameworks, such as MapReduce and Spark, are deployed across multiple sites to execute a set of tasks in a distributed manner. Such geo-distributed sites are connected with a wide-area network, and are usually heterogeneous in terms of the capability and the price of offered resources. In this section, we start with the analysis of the heterogeneities among geo-distributed sites, and briefly introduce the concept of MapReduce, and then describe two IoT applications. Finally, we provide a motivation example to present the influence of different task placement methods on JCC and JCT.

A. Heterogeneities of Geo-Distributed Sites

An important characteristic of geo-distributed IoT data analytics is that sites are highly heterogeneous in terms of the capacities and prices of hardware resources and the volume of IoT data.

As reported in [3], the computing capacity of the largest online service provider can be up to two orders of magnitude larger than that of the ordinary sites. Specifically, the computing capacity of datacenter is much higher than that of edge nodes. Moreover, those sites provide computing resources at diverse prices. For example, Amazon EC2 releases the same type of VM instance (t3.xlarge 4 vCPU 16-GB Memory) at different costs across six regions [13], as shown in Table II. It is clear that the prices of the same type of VM would considerably vary across geo-distributed sites; for instance, the highest price is 1.62 times that of the cheapest one in Amazon EC2 (i.e., Sao Paulo versus Virginia).

The WAN bandwidth across geo-distributed sites is very scarce compared with that inside each site [29]. Additionally, the network bandwidths among geo-distributed sites are heterogeneous, as reported in literatures [2], [6], and [30]. According to the measurement result of the Amazon EC2 service in 11 different regions, the gap between the bandwidths among sites is up to 12× [6]. Moreover, those intersite links also differ in the pricing models. As shown in Table II, the price of the uplink bandwidth at different sites on Amazon EC2 is also highly heterogeneous, and the price gap can be up to 7.4 times in the worst case. Moreover, even if the price of computing resources at one site is the most expensive across all sites, the price of network resources at this site may not be the highest (i.e., Sao Paulo versus Cape Town).

Such geo-distributed sites also differ in storage price. For example, the price of per GB data storage in Amazon S3 varies across different regions, as shown in Table II. The storage price usually ranges from 0.025 to 0.03 when renting 1-GB storage for 30 days. The highest price is two times higher than the cheapest one (i.e., Sao Paulo versus N. Virginia); however, the JCT of most geo-distributed data analytics jobs is usually less than 1 h, which is significantly lower than 30 days. Thus, we omit the impact of the storage cost on the total cost of executing a geo-distributed data analytics job.

Besides, the amounts of IoT data generated on different sites are also heterogenous. Currently, a large number of sensors are widely deployed at different locations, and the generated IoT data are stored nearby sites. Therefore, the IoT data generated are naturally geo-distributed [31]. Second, the number of sensors deployed in each area and the frequency of device being used vary greatly, and thus, the amounts of IoT data generated on different sites are highly heterogeneous [32].

B. MapReduce Framework

MapReduce is a parallel computing framework for large-scale data processing first introduced by Google in 2004. MapReduce includes two stages: 1) the map stage and 2) the reduce stage. The map stage applies a user-defined map function to process the given input data, and produces intermediate data. In this stage, the input data are divided into independent chunks, and processed in parallel. The intermediate data are generated in the form of key-value pairs, and they are shuffled to reduce tasks. The reduce stage applies a user-defined reduce function to keys and their associated values to generate the final results. Many parallel processing frameworks are realized based on MapReduce.

C. IoT Applications

Analyzing the IoT data of multiple sites can get very valuable information. Two IoT applications are listed as follows.

Application 1 (Pollution Control): Assuming that various environmental monitoring sensors are installed in various regions to obtain local environmental quality data. These data will be transmitted and stored in nearby edge nodes or edge clouds. Analyzing the environmental monitoring data collected in different regions can provide more effective data support for environmental protection workers in pollution control. For instance, analyzing the main factors affecting air quality

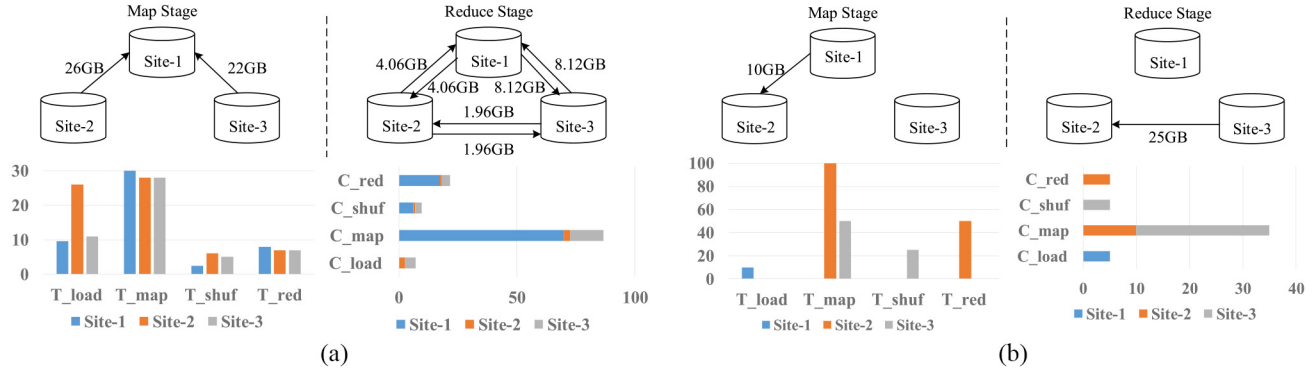

Fig. 3. Completion time and the total cost of executing a geo-distributed data analytics job under the Tetrium algorithm and a more economical approach. The top of each subfigure represents the task placement scheme of each stage. The bottom of each subfigure indicates the execution time and cost on the input data loading, map computation, shuffle transfer, and reduce computation four phases of each task placement scheme. In this figure, we use some symbols to represent the time and cost in these phases. For instance, C_load denotes the cost of the input data loading phase; T_load represents the consumed time during the input data loading phase. (a) Tetrium. (b) More economical approach.

from the historical environmental monitoring data of multiple regions.

Application 2 (Intelligent Transportation): In the city, a large number of nodes are deployed on streets to monitor the situation of crowds. Each node is equipped with various sensors, such as video cameras, audio sensors, and proximity sensors. All monitored data will be transmitted to nearby edge nodes. Analyzing the traffic data of different regions can provide data support for urban road planning, route planning, congestion prediction, and dredging.

D. Motivation: The Tradeoff Between the Completion Time and the Total Cost

We illustrate the motivation of this article with a representative example of calculating the completion time and total cost in Tetrium [3], which discovers the heterogeneity of geo-distributed sites in computing as well as network resources and attempts to reduce the JCT.

As mentioned in Section I, Fig. 2 shows an example of geo-distributed sites, among which site 1 has the most abundant network resource and computing resources, and it is associated with higher bandwidth and computing slot prices. Additionally, such sites may generate different amounts of metadata by IoT applications , i.e., site 1 has 10-GB metadata, site 2 has 40-GB metadata, and site 3 has 50-GB metadata. To ease the presentation, we assume that the analytics job has one map stage and one reduce stage. The map stage contains the input data loading phase and the map computation phase, while the reduce stage comprises the shuffle transfer phase and the reduce computation phase. The completion time of a phase is decided by the last site to finish the data transmission or computation tasks. We assume that the amount of the intermediate data generated by the map stage is half of the amount of the input data. We further assume that each map/reduce task would consume one computing slot. Each map task can process 100 M data in 2 s, and each reduce task can process 100 M data in 1 s.

Tetrium [3] is a representative task placement solution for geo-distributed data analytics. It minimizes the JCT by proposing an effective task placement strategy, as shown in Fig. 3(b). Before executing map tasks, to reduce the completion time of the map stage, Tetrium shifts some workload away from the bottleneck site (i.e., site 2) to other sites with a short transfer time increment. Specifically, site 2 and site 3 will transfer 26-and 22-GB data to site 1, respectively. The time of loading input data loading phase is decided by the data transmission process in which site 2 transfers 26-GB data to site 1. The available bandwidth of this process is dominated by the minimum value between the uplink bandwidth (1 GB/s) of site 2 and the downlink bandwidth (5 GB/s) of site 1. Thus, the transmission takes exactly $26/1 = 26$ s. The computation bottleneck is site 1 because it takes $2 \times \lceil[(26 + 10 + 22) \times 10]/40\rceil = 30$ s in the map stage. After the map stage, sites 1–3 generate 29-, 7-, and 14-GB intermediate data that will be assigned with 58%, 14%, and 28% of the reduce tasks, respectively. Thus, the consumption time of the entire shuffle transfer is decided by the download bandwidth of site 2, and it must download $29 \times 14\% = 4.06$-GB and $14 \times 14\% = 1.96$-GB data from sites 2 and 3, respectively. Thus, the download time of site 2 is $(4.06 + 1.96)/1 = 6.02$ s. The consumption time of the entire reduce computation phase is decided by site 1, which is $1 \times \lceil[(29 \times 10)/40]\rceil = 8$ s. Thus, the total JCT is $26 + 30 + 6.02 + 8 = 70.02$ s.

However, the resulting total cost of executing that job is very high. The bandwidth cost results from two stages, i.e., the input data loading and the shuffle transfer, which is equal to $(26\times0.1+22\times0.3)+(4.06\times0.1+4.06\times0.5+8.12\times0.3+8.12\times0.5+1.96\times0.2+1.96\times0.2) = 18.916$\$. The compute cost is also composed of the costs for the map computation phase and the reduce computation phase, which is calculated as $2\times(0.06\times580+0.01\times140+0.025\times280)+1\times21.6 = 108$\$. In summary, the total cost of running the job is $18.916+108 = 126.916$\$.

In reality, many geo-distributed data analytics have an upper bound for the JCT. If the real JCT does not exceed that upper bound, this will not introduce a significant negative impact on the Quality of Service (QoS) of these applications. To solve this essential problem, Fig. 3(a) presents a more economical

TABLE III
NOTATIONS AND DEFINITIONS

Notation	Definition
D	set of sites
A_r	amount of the IoT data generated at site r
x_r^d	amount of the IoT data transferred from site r to site d in the map stage
U_d, D_d	uplink/downlink bandwidth of site d
α_d	fraction of reduce tasks executed at site d
q	ratio of intermediate data to input data
I_d^{shuf}	volume of intermediate data at site d
S_d	number of computing slots at site d
P_r^d	price of bandwidth from site r to site d
C_d	price of a computing slot per unit time at site d
t_{map}, t_{red}	duration of a map/reduce task
r_{map}, r_{red}	amount of data for each map/reduce task
N_d^{map}, N_d^{red}	number of tasks at map/reduce stage in site d
T	deadline for the job

solution. In the input data loading phase, site 1 transfers all its data to site 2 with $10/1 = 10s$ and $10 \times 0.5 = 5\$$. The map tasks are executed in site 2 and site 3 at the cost of $max\{100, 50\} = 100s$ and $2 \times (5 + 12.5) = 35\$$, respectively. To use the cheapest computing slots, we transfer all data from site 3 to site 2. Thus, the shuffle transfer incurs $25/1 = 25$ s and $25 \times 0.2 = 5\$$. The reduce tasks in site 2 cause $1 \times \lceil (50 \times 10)/10 \rceil = 50$ s and $1 \times 500 \times 0.01 = 5\$$. Overall, the completion time of this job is $10 + 100 + 25 + 50 = 185$ s, and the total cost is $5 + 35 + 5 + 5 = 50\$$. When the deadline of this job is 200 s, the scheme in Fig. 3(a) is a better method because it can satisfy the deadline of this job with less cost. Therefore, in this article, we propose an approach that can minimize the JCC while adhering to its deadline.

IV. MODELING AND PROBLEM FORMULATION

In this section, we start with the problem definition, and then discuss the calculation of the total cost and completion time. Thereafter, we present the optimal cost problem for geo-distributed data analytics jobs.

A. Problem Description

To effectively analyze IoT application data across multiple edge nodes or data centers, we realize geo-distributed IoT data analytics based on the MapReduce framework. We investigate the problem of how to place tasks in the map and reduce stages to minimize the JCC of a geo-distributed IoT data analytics job under its deadline constraint. In each stage, we need to decide the tasks that should be placed on the site and determine the site that stores the required input data. Table III summarizes the major notations used in this article.

We assume that the input data of the job are already generated and saved across multiple geo-distributed sites, denoted as $D = \{1, 2, \ldots, j\}$. A_r represents the amount of the IoT data generated at site r. Each site can communicate with each other. Owing to the heterogeneities of the WAN bandwidth and computing capacity among the geo-distributed sites, the transmission time for obtaining the required input data on different sites can be uneven. Therefore, the available computational resource of site $d \in D$ is described as S_d, and the uplink and downlink bandwidth of site d is denoted as U_d and D_d, respectively. Furthermore, the resource prices among the geo-distributed sites are also heterogeneous; hence, the transmission and computing costs change consistently when different resources are employed. Let C_d represent the price of a computing slot per unit time at site d, and P_r^d denote the price of bandwidth per GB from site r to site d. In particular, if $r == d$, P_r^d is equal to 0.

An analytics job usually includes multiple MapReduce stages. For simplicity, we assume that each job has exactly one map stage and one reduce stage. We formulate the task placement of the geo-distributed IoT data analytics job for each stage independently. The map stage includes the input data loading and map computation phases. The reduce stage is divided into the shuffle transfer and reduce computation phases. Let $C_{\text{load}}, C_{\text{map}}, C_{\text{shuf}}, C_{\text{red}}$ denote the cost of the input data loading, the map computation, the shuffle transfer and the reduce computation, respectively; $T_{\text{load}}, T_{\text{map}}, T_{\text{shuf}}$, and T_{red} denote the consumed time during the input data loading, the map computation, the shuffle transfer, and the reduce computation, respectively. In Section V-D, we discuss in detail the problem of how to optimize the total cost of an analytics job with multiple map-reduce stages subject to a given deadline.

B. Modeling the Cost of Executing Geo-Distributed IoT Data Analytics Job

As aforementioned, performing a geo-distributed IoT data analytics job would incur the network cost as well as the computation cost. The data transfer cost consists of two parts, i.e., the transfer cost in the map stage and transfer cost in the reduce stage. In the map stage, the scheduler decides the amount of data exchanged between any pair of sites. Supposing that x_r^d GB data should be transmitted from site r to site d, then the total cost of input data loading in the map stage can be expressed as follows:

$$C_{\text{load}} = \sum_{r \in D} \sum_{d \in D - \{r\}} \left(x_r^d \times P_r^d \right). \tag{1}$$

To calculate the transfer cost in the reduce stage, we have to consider the volume of intermediate data generated at each site after completing the map stage. Note that after the input data loading, the volume of data in site d turns out to be $\sum_{r \in D} x_r^d$. After performing the map tasks in site d, some intermediate data would be generated and act as the input data of all reduce tasks. Let I_d^{shuf} denote the volume of intermediate data in site d and q represent the ratio of the amount of the intermediate data to the input data of the job. Then, we have

$$I_d^{\text{shuf}} = q \times \sum_{r \in D} x_r^d \quad \forall d \in D. \tag{2}$$

In the reduce stage, each site is assigned to perform a fraction of reduce tasks. Let α_d represent the fraction of reduce tasks executed in site d, where $\sum_{d \in D} \alpha_d = 1$. To finish the reduce tasks, all sites need to exchange part of their intermediate data with other sites hosting correspondence reduce tasks. This many-to-many transfer is referred as the shuffle transfer in many literature, which is very common in big data analytics jobs. In this article, we assume that the

keys of reduce tasks is evenly distributed. Thus, we can calculate the amount of data transferred to a site from other sites based on the number of tasks that is allocated to this site. Specifically, the amount of intermediate data that site r needs to obtain from site d is equal to $I_d^{\text{shuf}} \times \alpha_r$, where I_d^{shuf} is the amount of intermediate data at site d. Let C_{shuf} represent the cost of transferring intermediate data to reduce tasks, and can be calculated as

$$C_{\text{shuf}} = \sum_{d \in D} \sum_{r \in D-\{d\}} \left(I_d^{\text{shuf}} \times \alpha_r \times P_d^r \right). \tag{3}$$

The computation cost is another important part of the total cost of a geo-distributed job. The computation cost results from executing the map tasks as well as the reduce tasks. The corresponding costs are represented by C_{map} and C_{red}, respectively. The number of map and reduce tasks in each site is determined by the total size of data to be addressed and the amount of data each map and reduce task can process. As aforementioned, the amount of input data for map tasks in site d is $\sum_{r \in D} x_r^d$. After the shuffle phase, the size of input data for reduce tasks in site d is $\alpha_d \times \sum_{r \in D} I_r^{\text{shuf}}$. We assume that r_{map} and r_{red} denote the amount of data that each map/reduce task can process. N_d^{map} and N_d^{red} represent the number of map tasks and reduce tasks in site d, respectively, and can be described as follows:

$$N_d^{\text{map}} = \left\lceil \frac{\sum_{r \in D} x_r^d}{r_{\text{map}}} \right\rceil \tag{4}$$

$$N_d^{\text{red}} = \left\lceil \frac{\alpha_d \times \sum_{r \in D} I_r^{\text{shuf}}}{r_{\text{red}}} \right\rceil. \tag{5}$$

We further assume that each task consumes one computing slot. Thus, the number of computing slot in the map and reduce stage is N_d^{map} and N_d^{red}, respectively. Let t_{map} and t_{red} represent the execution time of a map/reduce task. Then, we have

$$C_{\text{map}} = t_{\text{map}} \times \sum_{d \in D} \left(N_d^{\text{map}} \times C_d \right) \tag{6}$$

$$C_{\text{red}} = t_{\text{red}} \times \sum_{d \in D} \left(N_d^{\text{red}} \times C_d \right). \tag{7}$$

C. Modeling the Completion Time of Executing Geo-Distributed IoT Data Analytics Job

The consumption time of such a job is a critical factor since the results of geo-distributed analytics jobs are usually used to make crucial and real-time decisions. The total processing time of a job consists of the network transfer time and computation time.

The network transfer time refers the total time of aggregating input data across sites (T_{load}) and the network shuffle time (T_{shuf}). T_{load} and T_{shuf} are dominated by the size of data to be transferred and the upload and download bandwidth of the involved sites. In the map stage, the amount of upload data from site d is $\sum_{r \in D-\{d\}} x_d^r$, and the amount of data that each site d need to download is $\sum_{r \in D-\{d\}} x_r^d$. Hence, the transfer time to upload and download data in site d is $[(\sum_{r \in D-\{d\}} x_d^r)/U_d]$ and $[(\sum_{r \in D-\{d\}} x_r^d)/D_d]$, respectively. For the reduce stage, site d needs to process α_d fraction of all reduce tasks. Therefore, the amount of data to be transferred out of site d is $(1 - \alpha_d) \times I_d^{\text{shuf}}$, and the volume of data to be transferred to site d is $\sum_{r \in D-\{d\}} (I_r^{\text{shuf}} \times \alpha_d)$. Thus, in the reduce stage, the transfer time to upload and download data is $([(1-\alpha_d) \times I_d^{\text{shuf}}]/U_d)$ and $[(\sum_{r \in D-\{d\}} I_r^{\text{shuf}} \times \alpha_d)/D_d]$, respectively. Consider that the completion time of flow is determined by the maximum one of the upload and download time in that process. Hence, we have

$$T_{\text{load}} = \max_{\forall d \in D} \left\{ \frac{\sum_{r \in D-\{d\}} x_d^r}{U_d}, \frac{\sum_{r \in D-\{d\}} x_r^d}{D_d} \right\} \tag{8}$$

$$T_{\text{shuf}} = \max_{\forall d \in D} \left\{ \frac{(1-\alpha_d) \times I_d^{\text{shuf}}}{U_d}, \frac{\sum_{r \in D-\{d\}} I_r^{\text{shuf}} \times \alpha_d}{D_d} \right\}. \tag{9}$$

The tasks in each site often need to be executed via multiple waves due to the limited setting of computing slots [3]. If site d has s_d slots, it requires $\lceil N_d^{\text{map}}/s_d \rceil$ and $\lceil N_d^{\text{red}}/s_d \rceil$ waves to finish its all map and reduce tasks, respectively. Let t_{map} and t_{red} denote the time consumed by a each map and a reduce task. Thus, the map and reduce computation time in site d are $t_{\text{map}} \times \lceil N_d^{\text{map}}/s_d \rceil$ and $t_{\text{red}} \times \lceil N_d^{\text{red}}/s_d \rceil$, respectively. Let T_{map} and T_{red} be the computation time of the map stage and the reduce stage, and each of which is dominated by the maximum computation time across all sites. Therefore, we have

$$T_{\text{map}} = \max_{\forall d \in D} \left\{ t_{\text{map}} \times \left\lceil \frac{N_d^{\text{map}}}{s_d} \right\rceil \right\} \tag{10}$$

$$T_{\text{red}} = \max_{\forall d \in D} \left\{ t_{\text{red}} \times \left\lceil \frac{N_d^{\text{red}}}{s_d} \right\rceil \right\}. \tag{11}$$

Hitherto, we have specified the calculation process of the completion time and the total cost when executing a geo-distributed analytics job.

D. Modeling the Cost-Aware Task Placement Problem

From the above descriptions, given any geo-distributed IoT data analytics job, we can formulate the task placement problem **P1** as follows:

$$\min \quad C_{\text{load}} + C_{\text{map}} + C_{\text{shuf}} + C_{\text{red}} \tag{12}$$

$$\text{s.t.} \quad T_{\text{load}} + T_{\text{map}} + T_{\text{shuf}} + T_{\text{red}} < T \tag{13}$$

$$\sum_{d \in D} x_r^d = A_r, x_r^d \geq 0 \quad \forall r \in D \tag{14}$$

$$\sum_{d \in D} \alpha_d = 1, \alpha_d \geq 0. \tag{15}$$

The optimization goal is to minimize the overall cost of a geo-distributed big data analytics job under a deadline constraint. Equation (13) indicates that the job must be completed before its deadline T. Equation (14) ensures that the total amount of transferred data to other sites $\sum_{d \in D-\{r\}} x_r^d$ from site r plus the amount of remaining data x_r^r in site r must equal the amount of original data in site r. Equation (15) requires that the sum of the ratios α_i at reduce task on all sites must be equal to 1.

In problem **P1**, the map task placement will directly impact the decision of reduce task placement. The object (12) and

(13) contain multiple monomials $k \times (x_d^r \times \alpha_d)$. Thus, the degree of the object and constraint (13) is 2, and problem **P1** is a quadratically constrained quadratic programming (QCQP) problem, which is NP-hard in general [33]. Equivalently, it is difficult to obtain the optimal solutions in polynomial time. Therefore, we further propose a heuristic algorithm to solve this problem under a given deadline constraint.

V. Cost-Aware Task Placement

In geo-distributed IoT data analytics, the task placement strategy dominates the total completion time and the cost overhead of such a job. This placement strategy mainly determines how many map tasks and reduce tasks should be placed at each site, and from which site each task reads data. Due to the NP-hard nature of the problem **P1**, we aim to propose a heuristics approach, MCGL, to derive out a reasonable task placement strategy. It can achieve a minimal total execution cost of such a job with respect to a given completion deadline.

We find that minimizing the JCT only may aggravate the total cost of the jobs; in contrast, minimizing the total cost slows down the completion of the jobs. Therefore, our MCGL method attempts to optimize the total cost with respect to a given deadline for each job. We first introduce the MinTime and MinCost algorithms to optimize the total cost and JCT, independently. Thereafter, based on the results, we propose a heuristic method (MCGL) to calculate the final task placement strategy, which minimizes the total cost with respect of the given deadline constraint. In this section, T^{cost} and T^{time} denote the JCT under the MinCost and MinTime algorithm, respectively. The JCC under the MinCost and MinTime algorithm is denoted by C^{cost} and C^{time}, independently.

A. Optimization of Total Cost

The problem **P1** is a quadratic programming (QP) problem, when the time constraint is not considered. The QP problem is usually NP-hard when the coefficient matrix of quadratic term of variables is indefinite [34]. However, the coefficient matrix of the quadratic term of variables of the geo-distributed IoT data analytic job is usually indefinite. To minimize the JCC, we first derive the task placement solution for each stage independently.

In the case of map task placement, it can be formulated as a linear program problem, as characterized by (16) to (17). The optimization goal (16) is to minimize the total cost in the map stage, which is the sum of the communication cost C_{load} and the computation cost C_{map}. C_{load} and C_{map} can be calculated by (1) and (6), respectively. To this end, we have to determine the volume of the data x_r^d migrated from site r to site d. Equation (17) contains n equality constraints. That is, for any site r, the sum of the remaining data at site r and the data transferred to other sites is equal to the generated data at site r for all sites

$$\text{Min} \quad C_{\text{load}} + C_{\text{map}} \tag{16}$$

$$\text{s.t.} \quad A_r = \sum_{d \in D} x_r^d, x_r^d \geq 0 \quad \forall r \in D. \tag{17}$$

Algorithm 1: MinCost Algorithm

Input: $A_r, q, t_{map}, t_{red}, r_{map}, r_{red}$. \\ job attributes
Output: An optimal solution for minimizing the JCC
1 $x_{r,cost}^d$ = minMapCost(A_r, t_{map}, r_{map});
2 I_{shuf}^d = getInterData($x_{r,cost}^d, q$);
3 $\alpha_{d,cost}$ = minReduceCost($I_{shuf}^d, t_{red}, r_{red}$); \\ According to the result in the map stage, we get the fraction of all reduce tasks $\alpha_{d,cost}$ allocated to each site;
4 T^{cost} = getTime($x_{r,cost}^d, \alpha_{d,cost}$);
5 C^{cost} = getCost($x_{r,cost}^d, \alpha_{d,cost}$);
6 **return** $x_{r,cost}^d, \alpha_{d,cost}, T^{cost}, C^{cost}$;

In the case of the reduce stage, each reduce task needs to read the generated intermediate data from all map tasks. Thus, we ought to determine the number of reduce tasks that should be executed on each site. Equation (18) shows the optimization goal, which minimizes the sum of the network shuffle cost (C_{shuf}) and the reduce computation cost (C_{red}). C_{shuf} and C_{red} are calculated by (3) and (7), respectively. The decision variables α_d are the proportion of reduce tasks allocated to site d. The constraint (19) indicates that all the sites must complete all the reduce tasks of that job in a distributed manner

$$\text{Min} \quad C_{\text{shuf}} + C_{\text{red}} \tag{18}$$

$$\text{s.t.} \quad \sum_{d \in D} \alpha_d = 1, \alpha_d \geq 0. \tag{19}$$

Based on above analysis, we design Algorithm 1 to optimize the total cost. The functions minMapCost() and minReduceCost() in this algorithm correspond to (16) and (17), and (18) and (19), respectively. The inputs of Algorithm 1 include the parameters of job attributes, and the definitions of those parameters are shown in Table III. When the job is submitted to the system, the amount of input data (A_r) at each site can be determined. In the production cluster, most analytics jobs recur [18]. Thus, other parameters (q, t_{map}, t_{red}, r_{map}, and r_{red}) can be set according to historical workloads.

Specifically, it first employs the function minMapCost() to minimize the sum of the communication cost C_{load} and the computation cost C_{map} in the map stage, and calculates the value of $x_{r,\text{cost}}^d$, i.e., the amount of the data to be transferred from site r to site d (line 1). In the function minMapCost(), it uses existing solvers to solve the linear programming problem expressed from (16) to (17). Thereafter, it obtains the amount of intermediate data I_{shuf}^d by $x_{r,\text{cost}}^d$ based on (2) (line 2). Next, it calculates the fraction $\alpha_{d,cost}$ of reduce tasks executed at each site using the function minReduceCost(), which also calls existing solvers to solve the problem formulated in (18) to (19). This function is aimed at minimizing the sum of the network shuffle cost (C_{shuf}) and the reduce computation cost (C_{red}) in the reduce stage (line 3). Then, we use the getTime() and getCost() functions to calculate the JCT and the JCC ($T^{\text{cost}}, C^{\text{cost}}$), respectively (lines 4 and 5). Finally, it returns the task placement scheme ($x_{r,cost}^d, \alpha_{d,cost}$), JCT ($T^{cost}$), and total cost ($C^{cost}$) under the MinCost algorithm.

In this process, we use the getTime() function to calculate the JCT under the decision variables ($x_{r,\text{cost}}^d$, $\alpha_{d,\text{cost}}$).

Algorithm 2: MinTime Algorithm

Input: $A_r, q, t_{map}, t_{red}, r_{map}, r_{red}$. \\ job attributes
Output: An optimal solution for minimizing the JCT
1 $x^d_{r,time}$ = minMapTime(A_r, t_{map}, r_{map});
2 I^d_{shuf} = getInterData($x^d_{r,time}, q$);
3 $\alpha_{d,time}$ = minReduceTime($I^d_{shuf}, t_{red}, r_{red}$); \\ According to the result in the map stage, we derive the fraction of reduce tasks $\alpha_{d,time}$ executed at each site;
4 T^{time} = getTime($x^d_{r,time}, \alpha_{d,time}$);
5 C^{time} = getCost($x^d_{r,time}, \alpha_{d,time}$);
6 **return** $x^d_{r,time}$, $\alpha_{d,time}$, T^{time}, C^{time};

Specifically, we first calculate the time of input data loading, the map computation, the shuffle transfer, and the reduce computation phases, respectively, according to (8)–(11). For example, the time of the input data loading phase can be calculated by $x^d_{r,\text{cost}}$ and (8). Thereafter, the overall time of the job T^{cost} can be obtained by summing T_{load}, T_{map}, T_{shuf}, and T_{red} together.

Similarly, we use the getCost() function to calculate the overall JCC. The cost of input data loading, the map computation, the shuffle transfer, and the reduce computation phases can be calculated by (1), (3), (6), and (7). For instance, the cost of the input data loading phase can be calculated by $x^d_{r,\text{cost}}$ and (1). Thus, the overall cost of the job C^{cost} is equal to $C_{\text{load}} + C_{\text{map}} + C_{\text{shuf}} + C_{\text{red}}$.

B. Optimization of Completion Time

If we do not consider the cost overhead and just want to finish a geo-distributed IoT data analytics job as soon as possible, the optimization problem for minimizing the completion time can be formulated as a mixed-integer linear programming (MILP) problem. As the increasing of sites, this problem would be very time consuming to derive a better feasible solution. Many efforts have been made to accelerate the execution of geo-distributed data analytics jobs [2], [3], [6], [26].

In this article, we first rely on Tetrium [3], the state-of-the-art solution to optimize the completion time of the map stage and the reduce stage, separately, as shown in Algorithm 2. In the map stage, the task placement problem can be formulated as a linear programming. The goal is to minimize the input data loading time plus the computation time of map tasks (T_{load}+T_{map}). The constraints (21) and (22) demonstrate that the input data loading time is determined by the time when the final data are transferred. Equation (23) further indicates that the computation time of map tasks is bounded by the completion time of the last map task. The output of this process is the amount of data that should be transmitted from an arbitrary site r to another site d

$$\text{Min} \quad T_{\text{load}} + T_{\text{map}} \tag{20}$$

$$\text{s.t.} \quad T_{\text{load}} \geq \frac{\sum_{r \in D-\{d\}} x^r_d}{U_d} \quad \forall d \in D \tag{21}$$

$$T_{\text{load}} \geq \frac{\sum_{r \in D-\{d\}} x^d_r}{D_d} \quad \forall d \in D \tag{22}$$

$$T_{\text{map}} \geq t_{\text{map}} \times \left\lceil \frac{N^{\text{map}}_d}{s_d} \right\rceil \quad \forall d \in D \tag{23}$$

$$A_r = \sum_{d \in D} x^d_r, x^d_r \geq 0 \quad \forall r \in D. \tag{24}$$

After the map stage, each site r needs to retrieve $\alpha_r \times I_d$ data from site d. The upload and download times at site d are $([(1-\alpha_d) \times I^{\text{shuf}}_d]/U_d)$ and $[(\sum_{r \in D-\{d\}} I^{\text{shuf}}_r \times \alpha_d)/D_d]$, respectively. Let N^{red}_d denote the number of allocated computing slots in site d for reduce tasks. Thus, the computing time of each reduce task at each site d is $t_{\text{red}} \times \lceil \frac{N^{\text{red}}_d}{s_d} \rceil$. To minimize the completion time of the entire reduce stage, the reduce task placement problem can be formulated as follows:

$$\text{Min} \quad T_{\text{shuf}} + T_{\text{red}} \tag{25}$$

$$\text{s.t.} \quad T_{\text{shuf}} \geq \frac{(1-\alpha_d) \times I^{\text{shuf}}_d}{U_d} \quad \forall d \in D \tag{26}$$

$$T_{\text{shuf}} \geq \frac{\sum_{r \in D-\{d\}} I^{\text{shuf}}_r \times \alpha_d}{D_d} \quad \forall d \in D \tag{27}$$

$$T_{\text{red}} \geq t_{\text{red}} \times \left\lceil \frac{N^{\text{red}}_d}{s_d} \right\rceil \quad \forall d \in D \tag{28}$$

$$\sum_{d \in D} \alpha_d = 1, \alpha_d \geq 0. \tag{29}$$

Algorithm 2 first invokes the minMapTime() function to calculate the amount of data $x^d_{r,\text{time}}$ in the input data loading phase (line 1). This function can solve the problem expressed in (20)–(24) to minimize the completion time in the map stage. Next, it calculates the value of I^d_{shuf} by $x^d_{r,\text{time}}$ (line 2). Then, it obtains the fraction of reduce tasks executed at each site using the minReduceTime() function (line 3). This function is used to solve the problem formulated in (25) to (29) to minimize the network shuffle time plus the reduce computation time of all reduce tasks ($T_{\text{shuf}} + T_{\text{red}}$). Thereafter, we use the getTime() and getCost() functions to calculate the JCT and total cost (T^{time}, C^{time}) independently (lines 4 and 5). Finally, the algorithm returns the final solution (line 6).

C. Tradeoff Between Completion Time and Total Cost

The above two algorithms only attempt to optimize either the completion time or the total cost; that is, they fail to provide a reasonable tradeoff between these two metrics. To obtain a better tradeoff between the completion time and the total cost, we provide a novel algorithm MCGL. For any geo-distribute data analytics job, MCGL attempts to optimize the total cost with respect to the constraint of JCT.

To optimize the total cost, the task placement solution of the MinCost algorithm will choose cheaper WAN links to transfer data and cheaper computing slots to execute computation. In contrast, for optimizing the completion time, the task placement solution of the MinTime algorithm attempts to transfer data on the links with high bandwidth, and compute tasks on the sites with sufficient computing resources. To discover the relationship between two solutions, we conducted an experiment.

In this experiment, we measure the JCT and JCC of the job shown in Fig. 2, where we adjust the decision variables

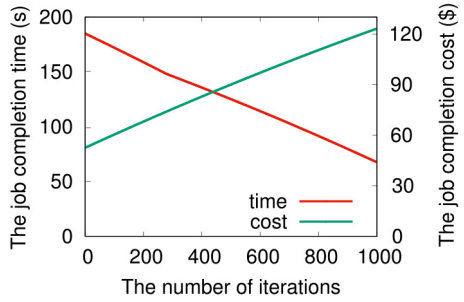

Fig. 4. JCT and the JCC at different adjust ratio of variables.

Algorithm 3: MCGL Algorithm

Input: $A_r, q, t_{map}, t_{red}, r_{map}, r_{red}$, \\ job attributes T, β. \\ the job deadline and the adjusting step

Output: An optimal solution for minimizing the JCC with a deadline

1 $x^d_{r,time}, \alpha_{d,time}, T^{time}, C^{time}$ = MinTime();
2 $x^d_{r,cost}, \alpha_{d,cost}, T^{cost}, C^{cost}$ = MinCost();
3 **if** $T^{time} > T$ **then**
4 **return** *null* \\ cannot get feasible solution
5 **if** $T^{cost} \leq T$ **then**
6 **return** $x^d_{r,cost}, \alpha_{d,cost}$;
7 $x^d_{r,new} \leftarrow x^d_{r,cost}$; $\alpha_{d,new} \leftarrow \alpha_{d,cost}$; $T_{new} \leftarrow T^{cost}$;
8 **while** $T_{new} > T$ **do**
9 $x^d_{r,new}, \alpha_{d,new}$ = adjustment(β);
10 T_{new} = getTime($x^d_{r,new}, \alpha_{d,new}$);
11 $x^d_r \leftarrow x^d_{r,new}$; $\alpha_d \leftarrow \alpha_{d,new}$;
12 **return** x^d_r, α_d;

between the two task placement solutions. Specifically, we assume that the value for the decision variables using the MinCost algorithm is $x^d_{r,\text{cost}}$ and $\alpha_{d,\text{cost}}$, and the value for the decision variables using MinTime algorithm is $x^d_{r,\text{time}}$ and $\alpha_{d,\text{time}}$. We assume that the adjusting step of variables is $\beta \in [0, 1]$. In this experiment, β for each adjustment is set to 0.001. For each adjustment, the variable value is adjusted as $x^d_{r,\text{cost}} - (x^d_{r,\text{cost}} - x^d_{r,\text{time}}) \times \beta \times k$ when $x^d_{r,\text{cost}} \geq x^d_{r,\text{time}}$, or as $x^d_{r,\text{cost}} + (x^d_{r,\text{time}} - x^d_{r,\text{cost}}) \times \beta \times k$ when $x^d_{r,\text{cost}} < x^d_{r,\text{time}}$. k is the number of iterations and ranges from 0 to 1000. Fig. 4 shows the JCT and the JCC at different adjust ratio ($\beta \times k$).

We observe that if we adjust the decision variables between the two task placement solutions, then the JCT and JCC exist the monotonicity in this interval. The JCC increases, while the JCT decreases, with increasing number of iterations; therefore, there is a remarkable negative correlation between the JCT and the JCC. Thus, we propose the MCGL algorithm (Algorithm 3), and the flowchart of MCGL is shown in Fig. 5. The specific steps are as follows.

The first step in our MCGL method is to generate two sets of solutions via leveraging the MinTime algorithm and the MinCost algorithm, respectively. Each solution contains a set of decision variables (lines 1 and 2). The decision variables of problem **P1** include the volume of the data, transferred from each site to all other sites in the map stage, and the fraction of reduce tasks assigned to perform at each site. To further judge

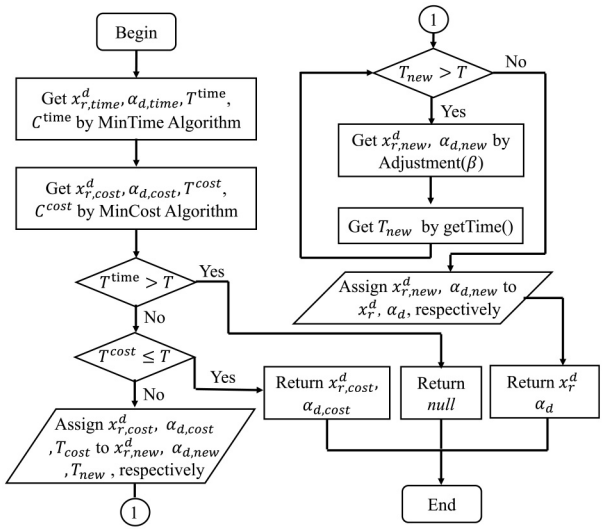

Fig. 5. Flowchart of MCGL.

the existence of a feasible solution for problem **P1**, MCGL checks if the time calculated by the MinTime algorithm (i.e., T^{time}) exceeds the given deadline T (lines 3 and 4). T is usually set by users who want to complete this job. If true, it means that we cannot obtain a feasible solution for problem **P1**. Thereafter, we check whether the time calculated by the MinCost algorithm (T^{cost}) is less than given time T (lines 5 and 6). If true, MCGL returns $x^d_{r,\text{cost}}$ and $\alpha_{d,\text{cost}}$ directly. This indicates that the JCT of the feasible solution, generated by our MinCost algorithm, satisfies the deadline T.

If MCGL does not find a feasible solution via the above steps, it has to heuristically adjust the parameter settings to find a feasible solution, which incurs the least cost within the completion deadline. Specifically, we assign the variables $x^d_{r,\text{cost}}$, $\alpha_{d,\text{cost}}$, T^{cost} to $x^d_{r,\text{new}}$, $\alpha_{d,\text{new}}$, T_{new} (line 7). Thereafter, in the *while* loop, we iteratively adjust the variables $x^d_{r,\text{cost}}$ and $\alpha_{d,\text{cost}}$ to generate feasible solutions. Specifically, when the current JCT is larger than the deadline T, we use function adjustment(β) to update decision variables $x^d_{r,\text{new}}$ and $\alpha_{d,\text{new}}$ in a greedy manner. The parameter $\beta \in [0, 1]$ is the adjustment step. In each adjustment, we update $x^d_{r,\text{new}}$ and $\alpha_{d,\text{new}}$ to $x^d_{r,\text{new}} - (x^d_{r,\text{cost}} - x^d_{r,\text{time}}) \times \beta$ and $\alpha_{d,\text{new}} - (\alpha_{d,\text{cost}} - \alpha_{d,\text{time}}) \times \beta$, respectively. $x^d_{r,\text{new}}$ and $\alpha_{d,\text{new}}$ will be updated for multiple rounds until $T_{\text{new}} < T$. Since β ranges from 0 to 1, $x^d_{r,\text{new}}$ is always between $x^d_{r,\text{cost}}$ and $x^d_{r,\text{time}}$, and $\alpha_{d,\text{new}}$ is always between $\alpha_{d,\text{cost}}$ and $\alpha_{d,\text{time}}$. Thus, $x^d_{r,\text{new}}$ and $\alpha_{d,\text{new}}$ can always satisfy its constraint. Finally, MCGL will return the final feasible solution.

In MinCost and MinTime algorithms, the most time-consuming process is solving the linear programming problems [minMapCost(), minReduceCost(), minMapTime(), and minReduceTime()]. The linear programming problem can be solved by normal solvers (simplex method, interior point method, etc.) in the polynomial time, and thus, the optimal time complexity of the two algorithms is $O(n^{3.5} \times L)$ [35], where n represents the number of variables, and L denotes the scale of the problem. In our MCGL algorithm, the most

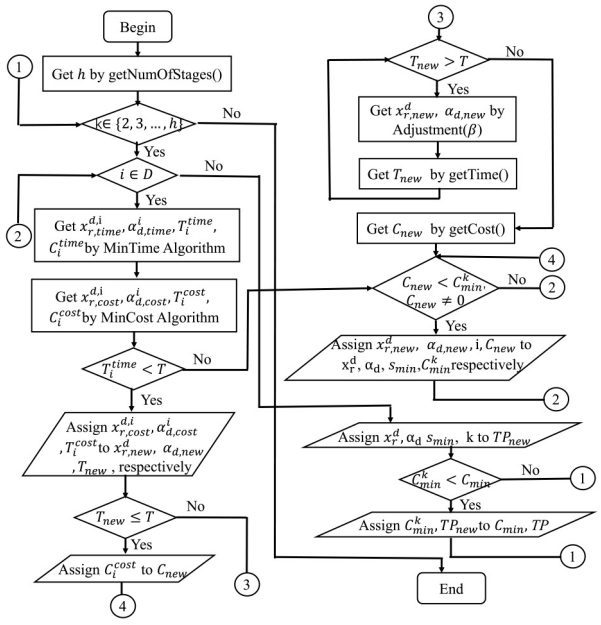

Fig. 6. Flowchart of MCGL+.

time-consuming process is calling MinCost and MinTime algorithms for execution. Hence, the time complexity of MCGL algorithm is $O(n^{3.5} \times L)$.

D. Improved Method for General Multistage Job

MCGL optimizes the total cost of a two-stage job under a given deadline. However, a big data analytics job usually contains multiple dependent stages, which can be abstracted as a directed acyclic graph (DAG). In addition, to obtain the final result data for such a job, the system has to aggregate all results data from all sites to a main site because the resulting data of MCGL are distributed across multiple sites. This process incurs more data transfer, increasing the total cost and the JCT.

To minimize the total cost of a multistage job under the given deadline, a traditional solution performs the job in a centralized manner, where a site receives the entire input data of all sites and finishes all tasks independently. This solution is easy-to-deploy but inefficient, as proved in literature [2] and [3]. Another intuitive solution is to split the multistage job into multiple two-stage jobs, and each of these jobs is solved by the MCGL method, which computes the task placement strategy and spreads tasks across all sites. Obviously, this solution suffers from unacceptable computing time. Moreover, the accumulated errors of all split two-stage jobs can make the resulted cost deviating significantly from the optimal value.

In this section, we propose the MCGL+ method to execute multistage jobs effectively and efficiently. The flowchart of MCGL is shown in Fig. 6. This method is designed based on the observation that the amount of data typically decreases after completing each stage. Specifically, MCGL+ first computes the placement strategy of map and reduce tasks across all sites. In a multistage job, tasks of the previous stage that have been completed generate the input data for the remaining stages, and the data amount is usually far less than the amount of the original data. Therefore, the MCGL+ method will select a single site to receive all generated data and execute all remaining stages. The MCGL+ method eliminates data transmissions among the remaining stages, saving transmission time and cost. Especially, some jobs may generate massive intermediate data after executing the first reduce stage, which can deteriorate the performance of MCGL+. Thus, we need to decide the number of stages that execute tasks in distributed sites. Algorithm 4 describes the details of this method.

Algorithm 4: MCGL+ Algorithm

Input: A_r, q, t_{map}, t_{red}, r_{map}, r_{red}, \\ job attributes T, β. \\ the job deadline and the adjusting step
Output: An optimal solution for minimizing the JCC with a deadline

1 h = *getNumOfStages*();
2 $C_{min} \leftarrow$ *Double.MAX_VALUE*, $TP \leftarrow$ *null*;
3 **foreach** $k = 2, 3, ..., h$ **do**
4 $C^k_{min} \leftarrow$ *Double.MAX_VALUE*; $s_{min} \leftarrow 0$, $i \leftarrow 1$;
5 **foreach** i *in* D **do**
6 $x^{d,i}_{r,time}$, $\alpha^{i}_{d,time}$, T^{time}_i, C^{time}_i = MinTime(i, k);
7 $x^{d,i}_{r,cost}$, $\alpha^{i}_{d,cost}$, T^{cost}_i, C^{cost}_i = MinCost(i, k);
8 $C_{new} \leftarrow 0$;
9 **if** $T^{time}_i < T$ **then**
10 $x^{d}_{r,new} \leftarrow x^{d,i}_{r,cost}$; $\alpha_{d,new} \leftarrow \alpha^{i}_{d,cost}$; $T_{new} \leftarrow T^{cost}_i$;
11 **if** $T_{new} \le T$ **then**
12 $C_{new} \leftarrow C^{cost}_i$;
13 **else**
14 **while** $T_{new} > T$ **do**
15 $x^{d}_{r,new}$, $\alpha_{d,new}$ = adjustment(β);
16 T_{new} = getTime($x^{d}_{r,new}$, $\alpha_{d,new}$, i);
17 C_{new} = getCost($x^{d}_{r,new}$, $\alpha_{d,new}$, i);
18 **if** $C_{new} < C^k_{min}$ *and* $C_{new} \neq 0$ **then**
19 $x^{d}_{r} \leftarrow x^{d}_{r,new}$; $\alpha_{d} \leftarrow \alpha_{d,new}$; $s_{min} \leftarrow i$;
20 $C^k_{min} \leftarrow C_{new}$;
21 $TP_{new} \leftarrow (x^{d}_{r}, \alpha_{d}, s_{min}, k)$;
22 **if** $C^k_{min} < C_{min}$ **then**
23 $C_{min} \leftarrow C^k_{min}$, $TP \leftarrow TP_{new}$;
24 **else**
25 break;
26 **return** TP

In Algorithm 4, steps 5–21 calculate the task placement with minimal JCC when the tasks after stage k are executed at a single site. Notations $C^k_{\min}$ and $s_{\min}$ represent the minimum JCC and the selected site to execute all tasks after stage k, respectively. For each site, we will calculate the cost of executing all remaining tasks, and the site with the minimum cost will be selected as $s_{\min}$. In each iteration, MCGL+ checks a new site and generates two task placement strategies by the MinTime and MinCost algorithms at site i (lines 6 and 7). Different from Algorithms 2 and 1, MinTime(i, k) and MinCost(i, k) execute the tasks before or equal to stage k in distributed sites, but the remaining tasks after stage k are executed on site i, and add the cost of executing all remaining tasks to the sum JCT and JCC.

We use $(T_i^{\text{time}}, C_i^{\text{time}})$ to denote the generated JCT and JCC for the MinTime(i, k), and $(T_i^{\text{cost}}, C_i^{\text{cost}})$ for the MinCost(i, k).

Recall that the two algorithms optimize the JCC or JCT of the first map and first reduce stages without deadline constraint. Therefore, T_i^{time} and C_i^{cost} represent the lower bound of the JCT and JCC, respectively. Therefore, MCGL+ can obtain a feasible solution only when $T_i^{\text{time}} < T$, where T is the deadline. Furthermore, if $T_i^{\text{cost}} < T$, MCGL+ can achieve the minimum cost C_i^{cost} (lines 11 and 12). Otherwise, we heuristically adjust the parameters to find the solution with the minimum cost C_{new} under the deadline constraint (lines 14–16). If C_{new} is the least among the costs of all checked sites, MCGL+ marks it as C_{min}, and the current site as s_{min} (lines 18–20). Thus, we can obtain the most suitable site after all sites are checked.

Furthermore, a simple strategy is adopted to determine the number of stages that are executed in distributed sites. We calculate the corresponding minimal costs of executing different number of stages in distributed sites, and select the one with the least cost. Specifically, we use h to represent the number of the whole stages of current job, and we calculate the task placement strategy and corresponding cost for at most h times. Among that in the kth iteration, we calculate the minimal cost of executing the first k stages in distributed sites. Notation C_{min} represents the minimal cost until the kth iteration. TP records the task placement strategy. Since the data amount generated by each stage is usually far less than the amount of the original data, when the minimal cost of executing the first k stages in distributed sites is larger than that of executing the first $k-1$ stages in distributed sites, we can determine the number of stages that are executed in distributed sites is $k-1$ (lines 22–25). In addition, in the MCGL+ algorithm, the iteration steps (lines 5–20) can be executed in parallel to reduce the running time.

From the time complexity analysis of MCGL in Section V-C, the complexities of the MinTime and MinCost algorithms are all $O(n^{3.5} \times L)$. In the MCGL+ algorithm, the MinCost and MinCost algorithms are called for $|D| \times h$ times, where $|D|$ is the number of sites, and h is the number of job stages. Hence, the complexity of the MCGL+ algorithm is also $O(|D| \times h \times n^{3.5} \times L)$.

VI. Performance Evaluation

In this section, we conduct comprehensive evaluations to measure the performance of our MCGL and MCGL+ methods using real datasets from Google [36], [37] and Alibaba [38].

A. Settings of Evaluation

Cluster Settings: We construct two networks, which contain 10 and 30 geo-distributed sites, respectively. In each network, the resource capacities and prices of different sites are heterogeneous. The configuration of the network is shown in Table IV. The resource capabilities of each site are set based on literature [2] and [3], and the resource prices are set according to Amazon EC2 [13]. More precisely, the bandwidth of each intersite link (U_d, D_d) ranges from 100 Mb/s to 2 Gb/s, and the link price P_r^d ranges from 0.02 to 0.5 \$/GB. The number of computing slots S_d in each site ranges in [100, 1000]. The price of each slot C_d is set between 4.5×10^{-5} and 7.5×10^{-5} \$ per second. Moreover, by default, we set the adjusting step β as 0.1.

TABLE IV
Configurations of Sites and System

Configuration	Value
Bandwidth (GB)	[0.1, 2]
Bandwidth price (\$/GB)	[0.02, 0.5]
Computing slots	[100, 1000]
Compute price (\$/s)	$[4.5, 7.5] \times 10^{-5}$
Adjustment ratio	0.1

Workload: We use synthetic workloads with job size distributions obtained from Google's production cluster and Alibaba cluster workload trace. The Google trace [36], [37] collects the information of machines, jobs, and tasks in a datacenter with 12.5k-machines over a month. For each job recorded in the trace, its arrival time, task number, input data size and distribution, and the required resources are recorded. However, the Google trace does not include the DAG structure information of jobs. Since MCGL focuses on the jobs consisting of a Map stage and a Reduce stage, it is evaluated on the Google trace. The Alibaba trace [38] was published by Alibaba Group in 2018. It contains records about 4k machines in a period of eight days. This trace includes many types of batch workloads, and most of them are DAG jobs. Therefore, we use Alibaba trace to evaluate the performance of MCGL+.

In the experiments, the parameters settings of each job are all assigned according to the data sets. Specifically, the number of tasks in each stage can be obtained directly from the data sets. The task execution time of task is assigned to the average execution time of all tasks in this stage. We divide the machines into different sites. According to the distribution of tasks for each job on those machines, we can obtain the volume of input data for each job on all sites. We set the deadline of a job according to its minimum completion time by MinTime (T^{time}) and maximum completion time using MinCost (T^{cost}) with the entire volume of cluster resources. Specifically, the deadline of a job is equal to $T^{\text{time}} + (T^{\text{cost}} - T^{\text{time}}) \times U$, where U is the ratio of deadline.

Simulation Framework: We developed MCGL and MCGL+ methods using Java language. The linear programming in our methods is solved by CPLEX. All simulation experiments are performed on a laptop with Intel 1.99-GHz processor and 24-GB memory. In our experiment, the computing slot is an abstract computing resource and does not represent a particular type of server. The duration of a task for different jobs is set according to the workload trace.

Baselines: We compare the MCGL and MCGL+ methods with the following methods in our evaluations.

1) *Centralized:* A traditional method, which aggregates all input data of job to a main site that runs the job and will cost the least.
2) *Tetrium [3]:* A state-of-the-art approach in recent years, which aims to optimize the placement of reduce tasks and the input data, and improves the JCT.

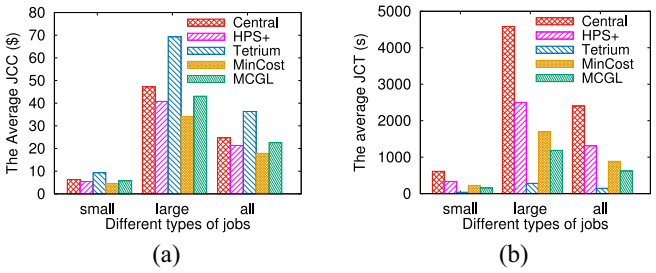

Fig. 7. (a) Average JCC and the (b) average JCT under different baselines.

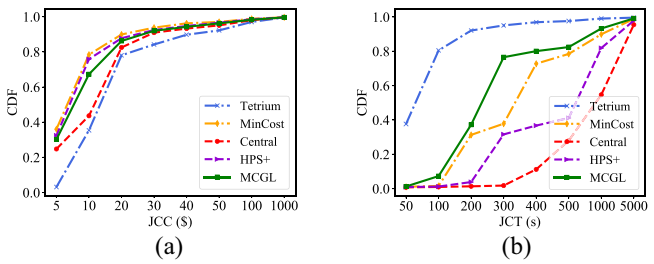

Fig. 8. CDF of JCC and JCT under different baselines. (a) CDF of JCC. (b) CDF of JCT.

3) *MinCost:* A pure method that just optimizes the total cost of executing a geo-distributed data analytics job. It attempts to transmit data along with low-cost links and compute data in low-cost slots as much as possible, without the consideration of the JCT.
4) *HPS+ [21]:* A new resource allocation algorithm that orchestrates the replica selection and task placement to minimize the volume of transferred data.

Evaluation Metrics: The following two metrics are used to measure the performance.

1) *Average JCT:* The total time of jobs divided by the number of jobs.
2) *Average JCC:* The total cost of jobs divided by the number of jobs.

B. Comparing MCGL With Others

In this section, we conduct evaluations based on the Google trace. According to the amount of input data, we divide the jobs in Google trace into two types, including the small-scale jobs and the large-scale jobs. Specifically, when the amount of input data of a job is less than 60 GB, the job is considered as a small-scale job. Otherwise, it is a large-scale job. Furthermore, we set the deadline of each job as $T^{\text{time}} + (T^{\text{cost}} - T^{\text{time}}) \times 0.7$.

We first evaluate the average JCT and the average JCC under different methods. Fig. 7(a) and (b) depicts the average cost and time of a set of jobs under different task scheduling methods. Clearly, for all methods, processing large-scale jobs incurs much higher JCC and JCT than processing small-scale jobs. MCGL incurs less JCC for any type of jobs than the centralized and Tetrium methods. Compared with Tetrium, MCGL reduces the average JCC by 40%. The HPS+ and MinCost methods aim at minimizing the transmission cost and the total cost, respectively. We can see that they incur less JCC than MCGL at the cost of violating the deadline constraints. Among these methods, it is clear that MCGL achieves the minimal average JCC with respect to the deadline constraints.

To illustrate the distribution of the JCC and JCT, we compare the CDF of JCC and JCT under different approaches, as shown in Fig. 8(a) and (b), respectively. In Fig. 8(a), MinCost achieves minimal JCC, and the JCC by MCGL is close to the JCC by MinCost. From Fig. 8(b), we can find that the JCT under MCGL is less than the JCT achieved by other methods, except for Tetrium. However, Tetrium causes the highest JCC. This experiment proves that MCGL achieves a better tradeoff than the other methods.

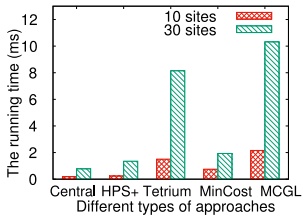

Fig. 9. Average running time of different algorithms.

TABLE V
PERCENTAGE OF DEADLINE VIOLATIONS

Methods	Central	HPS+	Tetrium	MinCost	MCGL
Percentage (%)	100	98.89	0	100	0

Furthermore, we calculate the percentage of deadline violations under different task placement methods, as shown in Table V. We can find that the jobs using MCGL and Tetrium algorithms can always complete tasks before the deadline. The jobs under centralized and Mincost algorithms always violate the deadline. The percentage of deadline violations with HPS+ is very high, which almost reaches 100%. Combined with the results shown in Fig. 7(a) and (b), it is clear that MCGL achieves minimal JCC before the deadline of each job compared with other baselines.

Thereafter, we evaluate the average running time of different algorithms under a varied scaling of the sites, and the results are shown in Fig. 9. We can see that the running time of different algorithms increases with the increasing number of sites. The running time of MCGL is slightly higher than those of other methods; however, the gap among these methods is small, and the running time of all these methods is less than 10 ms, which means they are effective for real application scheduling requests. In practice, solving the task placement model for each job can run in parallel and further reduces their running time.

Finally, we consider another impact factor, i.e., the number of sites, and set it as 10 and 30 to quantify its impact. Similarly, the configuration is shown in Table IV. The deadline of each job is equal to $T^{\text{time}} + (T^{\text{cost}} - T^{\text{time}}) \times 0.7$.

Fig. 10(a) and (b) shows the total JCC and average JCT when the number of sites is 10 and 30, respectively. Note that increasing the number of sites leads to a slight increase in costs increasing for Tetrium, HPS+, and MCGL. The reason is that Tetrium, HPS+, and MCGL attempt to reduce the JCT or the volume of transferred data; however, the more the sites the

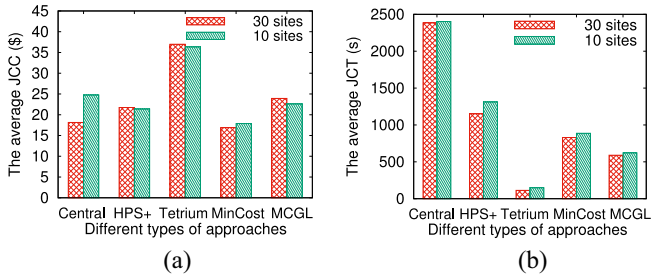

Fig. 10. (a) Average JCC and the (b) average JCT under different number of sites.

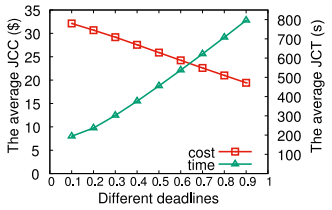

Fig. 11. Influence under different deadlines.

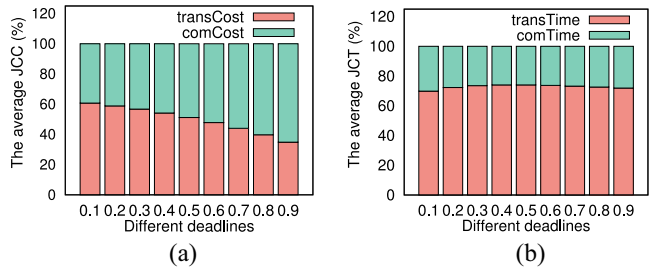

Fig. 12. Proportion of cost and time in transmission and computation under different deadlines. (a) Average JCC. (b) Average JCT.

more data there are to be exchanged, which increases the JCC. Therefore, a geo-distributed data analytics job across more sites could accelerate the completion process in MCGL, at the risk of further cost. The Centralized and MinCost algorithms attempt to minimize the JCC, and can select cheaper links and computing resources to complete job with more sites. The HPS+ aims at minimizing the transmission cost.

C. Impact of Varied Parameters

In this part, we quantify the impact of diverse parameters on MCGL, including the job deadline, and adjusting step.

Fig. 11 presents the average JCT and JCC with different deadlines. In the previous experiments, we set the deadline ratio as 0.7. In this experiment, we choose a range for the ratio of deadline T for each job from 0.1 to 0.9, and the time interval between two adjacent deadlines is $(T^{\text{cost}} - T^{\text{time}}) \times 0.1$. We find that the JCT decreases significantly with increasing job deadline. This is because longer deadline may allow MCGL to transmit more data with cheaper links, and execute more tasks on cheaper slots. In Fig. 11, we can see that MCGL can offer a proper tradeoff between the completion time and total cost according to users' requirements.

To ease the presentation, Fig. 12(a) indicates the proportions of transmission cost and computation cost in the total JCC, and Fig. 12(b) shows the proportions of transmission time and computation time in the total JCT. We find that the proportion of transmission cost continues to decrease with increasing deadline times. This phenomenon indicates that transferring more data with cheaper links can effectively reduce the total JCC. Additionally, as shown in Fig. 12(b), the proportions of transmission and computation time in the total JCT are stable, which shows that transmission and computation time of a job increase at the same rate with the increase of deadline times.

We further vary the adjusting step β from 0.001 to 0.1, and measure the performance of MCGL. Table VI presents the average JCT and average JCC, number of iterations, and running time of MCGL against the adjusting step. It shows that MCGL iterates fewer rounds and the running time increases when the adjusting step increases. Therefore, if the job is time sensitive, it is better for users to adopt a larger adjusting step. Additionally, the average completion cost decreases significantly with the decrease of the adjusting step. Therefore, if users want to save cost, it would be better to adopt a small adjusting step.

Therefore, the JCC under centralized, HPS+, and MinCost algorithms is lower with more sites at the risk of violating the deadline constraint. Furthermore, compared to other methods, MCGL achieves the lowest JCC except for the MinCost method, regardless of the number of sites is involved. As depicted in Fig. 10(b), the JCT decreases when more sites participate in the geo-distributed data analytics. This is because the more the sites, the more parallel the transmission and computing data, leading to an acceleration of the analysis process.

TABLE VI
IMPACT OF ADJUSTING STEP

Adjusting step	0.001	0.01	0.1
Average Cost($)	24.6324	24.7203	25.8842
Average Time(s)	517.3879	512.9817	455.6842
Iteration Number	423.2546	42.8101	4.9980
Running Time (ms)	2.2419	1.7011	1.4128

D. Performance of MCGL+

We compare MCGL+ with the centralized algorithm, Tetrium algorithm, and MinCost algorithm. We simulate ten sites based on Alibaba's trace data that includes the DAG information of those batched processing jobs. Similarly, we divide the jobs in Alibaba trace into two types according to the number of tasks in each job, including the small-scale jobs and the large-scale jobs. The configuration of the network in this experiment is similar to previous settings and is shown in Table IV.

In this experiment, the deadline of each job was also set according to the JCTs calculated by MinCost and MinTime. We first use MinTime(i) to calculate all the JCTs on all sites, and select the minimum JCT among those JCTs, represented by T^{time}. Thereafter, we use MinCost(i) to calculate all the JCTs on all sites, and choose the maximum JCT across those JCTs, denoted by T^{cost}. In this experiment, we set the deadline of the job as $T^{\text{time}} + (T^{\text{cost}} - T^{\text{time}}) \times 0.5$.

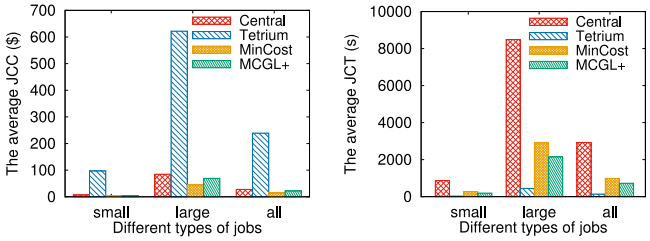

Fig. 13. (a) Average JCC and the (b) average JCT of multistages jobs under different methods.

As depicted in Fig. 13(a) and (b), the centralized method incurs much higher JCC and JCT than our MCGL+ method. The main reason is that the centralized method transfers too much data across different sites; hence, it requires more time and leads to more transmission cost. At the same time, executing the entire job on one site will also lead to longer computation time. Furthermore, we can find that the average JCC and the average JCT of MCGL+ range between the results of Tetrium and MinCost. However, the average completion cost using Tetrium is well above the average completion cost of MCGL+. In other words, the MCGL+ method is more cost-effective compared with the centralized and Tetrium methods. This is because when the job contains multiple stages, there usually exist multiple shuffle phases in this job. Completing the shuffle phase according to Tetrium will cause massive data transmission costs across geo-distributed sites. Compared with MinCost, because the ratio of deadline is 0.5, and thus, the JCT using MinCost cannot satisfy its deadline constraint. To this end, MCGL+ is most effective compared with the baselines and can offer a better tradeoff between the completion time and total cost.

VII. Conclusion and Future Work

In this study, we utilized geo-distributed data analytics technology to extract the value of IoT application data distributed across multiple geo-distributed sites. In geo-distributed IoT data analytics, the resource capacities and prices across sites are heterogeneous. We presented MCGL, which minimizes the completion cost of geo-distributed IoT data analytics job subject to a given deadline. MCGL adjusts the parameter settings heuristically from two locally optimal solutions to obtain a great approximate solution subject to a given deadline constraint. Furthermore, we proposed a more general method MCGL+ to tackle the geo-distributed IoT data analytics job with multiple stages. Extensive experiments show that our method considerably reduces the JCC and can always satisfy the deadline constraint. Especially, in our experiments, the percentage of jobs that are completed before their deadlines by MCGL is 100%. Besides, for jobs that are completed before their deadlines by different methods, MCCL achieves a 40% reduction in average JCC compared with other methods.

Future work is mainly threefold. First, the dynamicity of the WAN bandwidth across different sites also poses significant challenges to geo-distributed IoT data analytic jobs. Researchers discovered that large variances exist across different sites, and in certain cases, the available bandwidth is below 25% the maximum bandwidth [8]. Consequently, it is challenging to develop task placement strategies to optimize the JCC with a specific deadline in that dynamic scenario. This challenge could be solved by constructing a suitable model using the deep learning method to predict the intersite bandwidths.

Second, in this article, we mainly considered how to minimize the JCC of a geo-distributed IoT data analytics job. However, many users usually need to process multiple geo-distributed analytics jobs for which the optimization model will become very complex when these jobs have to compete for the shared bandwidth and computation resources. Thus, the deadline-aware job scheduling, resource allocation, and task placement method for multiple geo-distributed data analytics jobs must be studied in detail.

Finally, we proposed a heuristic based on the intuitive design principles and phenomenon that there is a remarkable negative correlation between the JCT and JCC, which has no theoretical guarantee in performance. Because many studies have focused on the QCQP problem, we can attempt to utilize the strengths of existing methods that can solve the QCQP problem, and calculate a better solution with acceptable running time.

References

[1] M. Bradbury, A. Jhumka, and T. Watson, "Trust trackers for computation offloading in edge-based IoT networks," in *Proc. IEEE INFOCOM*, Vancouver, BC, Canada, May 2021, pp. 1–10.

[2] Q. Pu *et al.*, "Low latency geo-distributed data analytics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 421–434, 2015.

[3] C.-C. Hung, G. Ananthanarayanan, L. Golubchik, M. Yu, and M. Zhang, "Wide-area analytics with multiple resources," in *Proc. CM EuroSys*, Apr. 2018, p. 12.

[4] S. M. Marzuni, A. Savadi, A. N. Toosi, and M. Naghibzadeh, "Cross-mapreduce: Data transfer reduction in geo-distributed mapreduce," *Future Gener. Comput. Syst.*, vol. 115, pp. 188–200, Feb. 2021.

[5] Y. Chen, L. Luo, D. Guo, O. Rottenstreich, and J. Wu, "SDTP: Accelerating wide-area data analytics with simultaneous data transfer and processing," *IEEE Trans. Cloud Comput.*, early access, Oct. 15, 2021, doi: 10.1109/TCC.2021.3119991.

[6] K. Hsieh *et al.*, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *Proc. USENIX NSDI*, Boston, MA, USA, Apr. 2017, pp. 1–20.

[7] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "ClariNet: WAN-aware optimization for analytics queries," in *Proc. USENIX OSDI*, Nov. 2016, pp. 435–450.

[8] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "AWStream: Adaptive wide-area streaming analytics," in *Proc. ACM SIGCOMM*, Aug. 2018, pp. 1–17.

[9] W. Chen, I. Paik, and Z. Li, "Cost-aware streaming workflow allocation on geo-distributed data centers," *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 256–271, Feb. 2017.

[10] Z. Hu, B. Li, and J. Luo, "Time-and cost-efficient task scheduling across geo-distributed data centers," *IEEE Trans. Parallel Distrib. Syst.* vol. 29, no. 3, pp. 705–718, Mar. 2018.

[11] W. Xiao, W. Bao, X. Zhu, and L. Ling, "Cost-aware big data processing across geo-distributed datacenters," *IEEE Trans. Parallel Distrib. Syst.* vol. 28, no. 11, pp. 3114–3127, Nov. 2017.

[12] A. C. Zhou, B. Shen, Y. Xiao, S. Ibrahim, and B. He, "Cost-aware partitioning for efficient large graph processing in geo-distributed datacenters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1707–1723, Jul. 2020.

[13] "Amazon EC2 Pricing." [Online]. Available: https://www.amazonaws.cn/ec2/pricing/ (Accessed: Nov. 8, 2021).

[14] K. Kloudas, M. Mamede, N. Preguiça, and R. Rodrigues, "Pixida: Optimizing data parallel jobs in wide-area data analytics," in *Proc. VLDB*, Aug./Sep. 2015, pp. 72–83.
[15] W. Li, R. Xu, H. Qi, K. Li, and X. Zhou, "Optimizing the cost-performance tradeoff for geo-distributed data analytics with uncertain demand," in *Proc. IEEE/ACM IWQoS*, Barcelona, Spain, Jun. 2017, pp. 1–6.
[16] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *Proc. USENIX NSDI*, May 2015, pp. 323–336.
[17] H. Hu, Y. Wen, T. Chua, and X. Li, "Cost-optimized microblog distribution over geo-distributed data centers: Insights from cross-media analysis," *ACM Trans. Intell. Syst. Technol.*, vol. 8, no. 3, pp. 1–18, 2017.
[18] Y. Huang *et al.*, "Yugong: Geo-distributed data and job placement at scale," *Proc. VLDB Endowment*, vol. 12, no. 12, pp. 2155–2169, Aug. 2019.
[19] C.-H. Chen, J.-W. Lin, and S.-Y. Kuo, "MapReduce scheduling for deadline-constrained jobs in heterogeneous cloud computing systems," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 127–140, Jan.–Mar. 2018.
[20] N. Lim, S. Majumdar, and P. Ashwood-Smith, "MRCP-RM: A technique for resource allocation and scheduling of mapreduce jobs with deadlines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1375–1389, May 2017.
[21] L. Zhao, Y. Yang, A. Munir, A. X. Liu, Y. Li, and W. Qu, "Optimizing geo-distributed data analytics with coordinated task scheduling and routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 279–293, Feb. 2020.
[22] S. Liu, W. Hao, and B. Li, "Optimizing shuffle in wide-area data analytics," in *Proc. IEEE ICDCS*, Jun. 2017, pp. 560–571.
[23] H. Wang and B. Li, "Mitigating bottlenecks in wide area data analytics via machine learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, pp. 155–166, Jan.–Mar. 2020.
[24] A. P. Iyer, A. Panda, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica, "Monarch: Gaining command on geo-distributed graph analytics," in *Proc. USENIX HotCloud*, Jul. 2018, pp. 1–7.
[25] A. Khan, M. Attique, and Y. Kim, "iStore: Towards the optimization of federation file systems," *IEEE Access*, vol. 7, pp. 65652–65666, 2019.
[26] A. Vulimiri *et al.*, "WANalytics: Geo-distributed analytics for a data intensive world," in *Proc. ACM SIGMOD*, May/Jun. 2015, pp. 1087–1092.
[27] K. Oh, A. Chandra, and J. B. Weissman, "A network cost-aware geo-distributed data analytics system," in *Proc. IEEE CCGRID*, May 2020, pp. 649–658.
[28] J. Bi *et al.*, "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 2, pp. 1172–1184, Apr. 2017.
[29] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing timeliness and cost in geo-distributed streaming analytics," *IEEE Trans. Cloud Comput.* vol. 8, no. 1, pp. 232–245, Mar. 2020.
[30] W. Li, X. Yuan, K. Li, H. Qi, and X. Zhou, "Leveraging endpoint flexibility when scheduling coflows across geo-distributed datacenters," in *Proc. IEEE INFOCOM*, Apr. 2018, pp. 873–881.
[31] B. Cheng, A. Papageorgiou, F. Cirillo, and E. Kovacs, "GeeLytics: Geo-distributed edge analytics for large scale IoT systems based on dynamic topology," in *Proc. IEEE WF-IoT*, Milan, Italy, Dec. 2015, pp. 565–570.
[32] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS*, Apr. 2017, pp. 1–10.
[33] V. Jeyakumar, A. M. Rubinov, and Z. Y. Wu, "Non-convex quadratic minimization problems with quadratic constraints: Global optimality conditions," *Math. Program.*, vol. 110, no. 3, pp. 521–541, 2007.
[34] P. M. Pardalos and S. A. Vavasis, "Quadratic programming with one negative eigenvalue is NP-hard," *J. Global Optim.*, vol. 1, no. 1, pp. 15–22, 1991.
[35] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. ACM STOC*, Apr./May 1984, pp. 302–311.
[36] "Google Cluster Data." [Online]. Available: https://commondatastorage.googleapis.com/clusterdata-2011-2/ (Accessed: May 1, 2011).
[37] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. ACM SoCC*, Oct. 2012, p. 7.
[38] "Alibaba Cluster Trace Program." [Online]. Available: https://github.com/alibaba/clusterdata (Accessed: Sep. 26, 2021).

Yiting Chen received the M.S. degree from the School of Information and Communication, Guilin University of Electronic Technology of China, Guilin, China, in 2017. She is currently pursuing the Ph.D. degree with the College of Systems Engineering, National University of Defense Technology, Changsha, China.

Her current research interests include geo-distributed data analytics, distributed computing, and machine learning.

Lailong Luo (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the School of Systems Engineering, National University of Defence Technology, Changsha, China, in 2013, 2015, and 2019, respectively.

He is currently a Lecturer with the School of Systems Engineering, National University of Defense Technology. His research interests include probabilistic data structures and data analysis.

Bangbang Ren received the B.S. and M.S. degrees in management science and engineering from the National University of Defense Technology, Changsha, China, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree with the College of Systems Engineering.

His research interests include software-defined network, network function virtualization, and approximation algorithm.

Deke Guo (Senior Member, IEEE) received the B.S. degree in industry engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008.

He is currently a Professor with the College of System Engineering, National University of Defense Technology. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks.

Prof. Guo is a member of ACM.