SDTP: Accelerating Wide-Area Data Analytics with Simultaneous Data Transfer and Processing

Yiting Chen, Lailong Luo, *Member, IEEE*, Deke Guo, *Senior Member, IEEE*, Ori Rottenstreich, and Jie Wu, *Fellow, IEEE*,

Abstract—For the efficient analysis of geo-distributed datasets, cloud providers implement data-parallel jobs across geo-distributed sites (e.g., datacenters and edge clusters), which are generally interconnected by wide-area network links. However, current state-of-the-art geo-distributed data analytic methods fail to make full use of the available network and computing resources. The main reason is that such geo-distributed methods must wait for bottleneck sites to complete the corresponding transmission and computation in each phase. Furthermore, such geo-distributed methods may be impractical to the network bandwidth dynamicity and diverse job parallelism. To this end, we propose a Simultaneous Data Transfer and Processing (SDTP) mechanism to accelerate wide-area data analytics, with the joint consideration of network bandwidth dynamics and job parallelism. In the SDTP, a site can execute the computation, provided that it obtains the required input data. As a result, the input data loading, map, shuffle, and reduce phases at each site need not wait for the completion of the previous phases of other sites. We further improve the SDTP method by offering more accurate time estimation and generalizing the mechanism to dynamic situations. The trace-driven results demonstrate that SDTP can improve the wide-area analytic job response time by 19% to 72% compared to other methods.

Index Terms—Wide-area data analytics, task scheduling, job response time, dynamic network, job parallelism.

1 Introduction

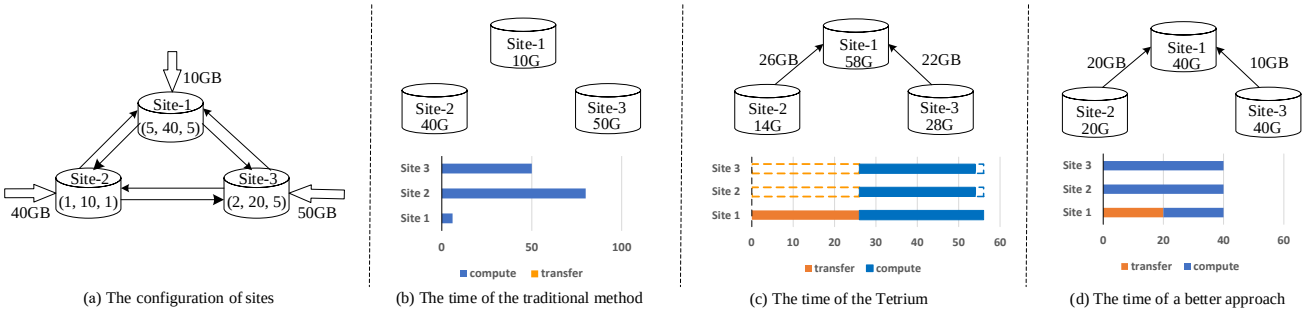
Cloud providers such as Google, Amazon, and Alibaba have deployed data centers globally to provide instant services. These services generate a large volume of data across the world [1], including transaction data, user logs and performance logs, etc. Mining geo-distributed data (also known as wide-area data analytics) is crucial for commercial recommendations, anonymous detection, performance upgrades, and system maintenance, among others. A distributed computing framework such as MapReduce is generally implemented to mine such massive datasets.

A dominant challenge in this computing paradigm is the heterogeneity of hardware resources among geo-distributed sites, including the computing, uplink bandwidth, and downlink bandwidth. For example, the gap between the bandwidth among sites of Amazon EC2 is up to 12 $\times$ [2], and the computation capacity of the largest online service provider may be up to two orders of magnitude larger than that of ordinary ones [3], [4]. With the development of edge computing, many applications are placed at the edge. However, the edge resources are naturally heterogeneous and insufficient [5], [6]. Moreover, the data amounts among geo-distributed sites are also highly heterogeneous [3], [7], [8]. As reported in reference [9], the amounts of Skype logs in over 100 different Azure sites indicate that the largest sites had 22 $\times$ the values as the smallest site. These heterogeneities significantly affect the execution of wide-area data analytics.

The job response time is a key metric in the analysis of geo-distributed data, which usually contains multiple tasks on one stage and is dominated by the completion time of the last task [3], [8], [10]. However, the heterogeneity of the hardware resources and diversity of the data volumes among geo-distributed sites have a serious impact on the job completion time. Thus, it is challenging to optimize this metric because multiple factors must be considered, including the WAN link bandwidth among sites [7], [8], [11], [12], the cost of the WAN links [13], [14], [15], [16], the computing resources in each site [3], [4], [17], and the data distribution [8], [9], [18]. To this end, Iridium [8] considers the heterogeneity of the WAN link bandwidth among sites and optimizes the placement of the reduce tasks to minimize the overall response time. Flutter [13] jointly considers the heterogeneities of both the bandwidth and cost of the WAN links among the sites. In contrast, Tetrium [3] offers a novel placement strategy for the map and reduce tasks with respect to the heterogeneities of both the WAN links and computing capacity among the sites. These methods can improve the response time to a certain extent.

Observation: *The state-of-the-art task scheduling strategies execute the MapReduce tasks in a strict sequential manner, which may idle the distributed sites and lead to unnecessary waiting time.* A toy example with three sites at the map stage is presented in Fig. 1, and the basic settings are shown in Fig. 1(a). We assume that each task processes 100 MB of data, and the time required to process each task is 2 sec.

- *Y. Chen, L. Luo and D. Guo are with the Science and Technology Laboratory on Information Systems Engineering, National University of Defense Technology, Changsha Hunan, 410073, China. E-mail: {chenyiting18, luolailong09, dekeguo}@nudt.edu.cn.*
- *Deke Guo is also with the College of Intelligence and Computing, Tianjin University, Tianjin, 300350, China.*
- *O. Rottenstreich is with the Department of Computer Science, Technion Israel Institute of Technology, Haifa 3200003, Israel, and also with the Department of Electrical Engineering, Technion Israel Institute of Technology, Haifa 3200003, Israel. E-mail: or@cs.technion.ac.il*
- *J. Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122. E-mail: jiewu@temple.edu.*
- *Corresponding author: Lailong Luo and Deke Guo.*



(a) The configuration of sites (b) The time of the traditional method (c) The time of the Tetrium (d) The time of a better approach

Fig. 1. Toy examples of heterogeneous geo-distributed sites and unequal job response time of different scheduling approaches. In Fig. 1(a), sites 1, 2 and 3 contain 10, 40, 50 GB of local data, respectively. The triple on each site represents the uplink bandwidth, the number of computing slots, and the downlink bandwidth, respectively.

Initially, sites 1, 2, and 3 contain 10, 40, and 50 GB of local data, respectively. They have an unequal number of computing slots, and the link bandwidth between a pair of sites is also heterogenous. We use a triple at each site to represent the upload bandwidth, the number of computing slots and the download bandwidth, respectively. Using such a setting, the traditional method usually executes the map tasks on local sites and thus avoids transferring local data to others. As shown in Fig. 1(b), the bottleneck is site 2 which needs 80 sec to complete its tasks. To reduce the response time, Tetrium [3] tries to balance the transferring and computation workloads among the sites, and migrates part of the data from those sites which cannot handle its workload effectively. It generates the map task placement strategy illustrated in Fig. 1(c). In this placement solution, sites 2 and 3 need to transmit 26 and 22 GB of data to site 1, respectively. During the transfer period, sites 2 and 3 remain idle. The sites can only begin the data computation when all transmissions have been completed. In this example, Tetrium requires 56 sec in total.

As indicated in Fig. 1(d), it is possible to improve the existing work on wide-area data analytics by balancing and parallelizing the data transfer and data processing. In this solution, sites 2 and 3 transmit less data to site 1 because the transfer time may overwhelm the processing time, even if site 1 has more computing slots. Moreover, sites 2 and 3 begin to process the data at time 0 as they require no data from the other sites. Therefore, the data transmission and data processing parallelized from 0 to 20 sec. As a result, the total running time of the map stage is 40 sec.

According to the above observation and motivation, this study presents a novel task scheduling mechanism known as Simultaneous Data Transfer and Processing (SDTP) to accelerate wide-area data analytics by decreasing the response time of wide-area data analytic jobs. With the joint consideration of the WAN link heterogeneity and parallel execution in each site, SDTP first models the scheduling problem as a non-linear programming problem, which is generally complicated and is hard to resolve. To resolve this non-linear programming problem, SDTP relaxes the non-linear programming model to a linear programming model. Thereafter, SDTP reduces the overall response time by migrating part of the data from the straggler site which leads to the longest response time to the idlest site which has the least response time.

In practice, the WAN bandwidth is dynamic over time. Reference [11] reports that the available bandwidth is below 25% of the maximum bandwidth between Amazon EC2 sites in some cases. However, previous work which devoted to the geo-distributed batch analytics assumed that the WAN bandwidth was static over the job execution period [3], [8], [17], [19]. Moreover, a complex relationship exists between the computation time and degree of parallelism. In general, the computation time of parallel tasks will decrease with an increase in the degree of parallelism. The assignment of additional resources to the job has a marginal impact on the performance. However, many references have assumed that the computation time will decrease constantly with the increase of the degree of parallelism [3], [17]. Therefore, we propose two improved approaches SDTP+ and SDTP++, which provide a more accurate computation time estimation of each stage within a site and can be generalized to dynamic situations.

The main contributions of this paper can be summarized as follows.

- We discover that state-of-the-art task scheduling strategies cannot make full use of the available network and computing resources, which may lead to unnecessary waiting time. To minimize the job response time, we present a new non-linear programming model to characterize the geo-distributed data analytic job with joint consideration of the resource heterogeneity and the degree of parallelism, as well as the dynamic WAN links among the sites.
- We make reasonable relaxations and assumptions on this model and propose a novel scheduling mechanism known as **SDTP** to accelerate wide-area data analytics. SDTP allows the sites to begin the data processing once they obtain the required input data.
- Considering the dynamicity of WAN and the influence of the degree of parallelism, we further present two improved scheduling mechanisms known as SDTP+ and SDTP++. The two approaches can provide more accurate time estimation and can be generalized to dynamic situations.
- We conduct trace-driven experiments to evaluate the performance of SDTP, SDTP+ and SDTP++. The results demonstrate that our methods outperforms existing methods and achieves a 19% to 72% reduction in the overall job response time.

The remainder of this paper is organized as follows: Section 2 presents the background and related work. Section 3 outlines the system model and formulates the task

placement problem of a geo-distributed data analytic job. Our SDTP, SDTP+, and SDTP++ methods are described in Section 4 and 5. We conduct extensive evaluations using realistic traces in Section 6, and we conclude the paper in Section 7.

2 Background and Related Work

In geo-distributed data analytics, multiple sites are connected by WAN, which restricts massive data transmission. Data processing frameworks, such as Hadoop and Spark, rely on the MapReduce model to implement their tasks on multiple geo-distributed sites in parallel. These geo-distributed sites may be highly heterogeneous in terms of the hardware capacity and data distribution. Furthermore, as the WAN bandwidth is also dynamic, the running time of the parallel frameworks exhibits certain properties. Therefore, in this section, we introduce the heterogeneities in geo-distributed sites, the dynamicity of the WAN bandwidth, and the parallel computing properties, followed by a discussion of related work on the geo-distributed data analytics.

2.1 Characteristics of Computation and Network Resources among Distributed Sites

Heterogeneity of resources and data distribution: An important characteristic and challenge in geo-distributed data analytics is that the resources among different sites are highly heterogeneous in terms of the hardware capacity [3]. Different sites are built at varying times and in varying regions, with diverse goals and budgets, thereby resulting in high heterogeneity [3], [20]. To demonstrate the heterogeneity of hardware resources, we compare the two most important hardware capacities, namely the computation and bandwidth. In particular, the computation capacity of one of the largest online service providers may be up to two orders of magnitude larger than that of ordinary ones [3], [4]. The impending trend of edge computing increases the heterogeneity. In private clusters, the compute capacities vary significantly from just a handful of cores to hundreds of cores [21]. The link bandwidth among different sites is also extremely diverse [8], [18], [22]. According to a measurement of Amazon EC2 in 11 different regions, the bandwidth among the sites is 15× smaller than the bandwidth within a site and 60× smaller in the worst case [2].

Moreover, the amounts of data generated on different sites are heterogenous, which has a serious impact on the job response time [23]. An analysis of Skype logs obtained from over 100 different Azure sites demonstrated that the median, 90th percentile, and maximum values were 8, 15, and 22 × larger than those of a site with the minimum log data [9]. Therefore, the data distribution across the sites may not be constant or may even be skewed in certain cases.

Dynamicity of the WAN: The dynamicity of the WAN bandwidth across different sites also poses significant challenges to geo-distributed data analytic jobs. Researchers discovered that large variances exist across different sites, and in certain cases, the available bandwidth is below 25% of the maximum bandwidth [11]. Consequently, it is difficult to develop task placement strategies for minimizing the job response time prior to the bandwidth change.

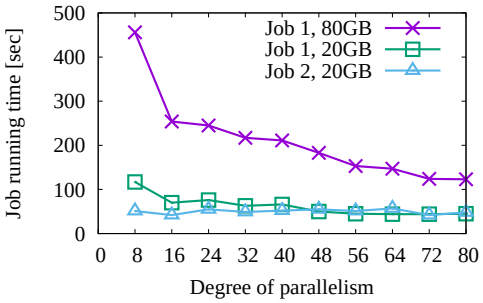


Fig. 2. Job running times with different degrees of parallelism (the number of slots) and amounts of data.

Degree of parallelism: In parallel computing, the amounts of data processed by the job is varied. Even if the jobs contain the same amounts of data, different degrees of parallelism will also lead to different job response time [24]. To determine the relationship among the computation time, the size of input data, and degree of parallelism, we have constructed a Spark cluster based on 11 virtual machines (1 manager node and 10 worker nodes). Each virtual machine contains 8 cores and 8 GB of main memory.

We have measured the running time of two jobs that ran on Spark under BigDataBench [25], and the results are presented in Fig. 2. The running time of Job 2 exhibits little variation with the increase in the parallelism. Job 1 exhibits a significant acceleration of up to 56 parallel slots when it processes 80 GB of input. When Job 1 processes a small input of 20 GB, it requires no more than 16 parallel slots. For all jobs, assigning additional parallel tasks beyond a "sweet spot" in the curve adds only diminishing gains. Thus, we need to design a method which can calculate more accurate computation time according to the type of job, the size of the input data and the degree of parallelism.

2.2 Related Work

Geo-distributed data analytics has received substantial attention over the past several years. Numerous efforts have been made to optimize the response times of such jobs.

Geo-distributed data analytics on MapReduce-based systems: Iridium [8] considers the heterogeneity of the WAN, and aims to minimize the response time of jobs across geo-distributed sites by optimizing the placement of the reduce task and involved input data. Considering the heterogeneities of the WAN links and the WAN bandwidth cost, Flutter [13] is a new task scheduling algorithm for reducing both the response time and the network cost of big data processing jobs. Tetrium [3] jointly considers the heterogeneities of the computing and networking resources when designing the placement strategy of the map and reduce tasks. Yugong [1] proposes a novel data and job placement strategy to minimize the cross-DC bandwidth use and to reduce the query latency. Liu et al. proactively aggregate the output data of map tasks and avoid repetitive data transfers in the shuffle stages to reduce the job response time [19]. The above methods mainly decrease the job response time, while massive WAN links and computing resources remain idle during the job execution.

To minimize the average job makespan, Zheng et al. study a joint scheduling optimization mechanism by overlapping the map and shuffle phases of two jobs to form

a strong pair [26]. However, it focuses on the single data-center applications. Furthermore, these methods all ignore the dynamic nature of the available WAN bandwidth and the influence of the degree of parallelism. Decima [24] uses reinforcement learning and neural networks to learn workload-specific scheduling algorithms and sets an efficient parallelism degree for each job to minimize the average job response time. However, it only attempts to optimize the average job response time in a single cluster.

Geo-distributed data analytics on other distributed systems: With a focus on the geo-distributed SQL query, CLARIENT [27] includes a novel WAN-aware query optimizer, which can achieve multi-query network-aware plan selection and task placement to ensure low query latency. WANalytics [28], Pixida [29] and Geode [30] attempt to reduce the bandwidth use across geo-distributed data centers and decrease the latency for SQL query requests. Lube [31] monitors geo-distributed data analytic queries in real-time, and detects and mitigates potential bottlenecks (e.g., bandwidth scarcity) at runtime to reduce the query response time.

Furthermore, Gaia [2] provides a machine learning synchronization model for cross-site learning tasks. It dynamically eliminates insignificant communication between sites to accelerate the execution of machine learning jobs. Monarch [32] optimizes the iterative processing style of graph-parallel systems to execute geo-distributed graph analytics effectively. Liu et al. present a hierarchical synchronous parallel mode, which results in lower WAN bandwidth use, faster convergence, and a lower WAN cost for wide-area graph analytics [12]. G-Cut [33] optimizes the performance of graph processing jobs by minimizing the inter-DC data transfer time. To save the amount of data transferred and to reduce the makespan, HPS+ [34] offers a new resource allocation algorithm. However, these methods have mainly focused on wide-area machine learning, SQL analytics, astronomical applications, and certain special fields. Therefore, they are not applicable for general big data processing frameworks such as MapReduce.

Optimizing systems for dynamic settings: Considering the challenge of scarce and variable WAN bandwidth, Turbo [7] adjusts the query execution plans for geo-distributed SQL queries in response to runtime resource variations across data centers. AWStream [11] automatically learns an accurate profile to model the relationship between the accuracy and bandwidth consumption of an application. Thereafter, it carefully adjusts the application data rate to match the available bandwidth, while maximizing the achievable accuracy. Besides, Magrino et al. introduce predictive treaties to predict the evolution of the system state in distributed transaction processing [35]. This method can reduce the coordination of geo-distributed applications and improve their performance. Unfortunately, these methods only focus on some specific areas and are not applicable to geo-distributed data analytics on MapReduce-based systems.

In this study, we propose a novel scheduling mechanism named SDTP to accelerate wide-area data analytics. The method attempts to make full use of the available network and computing resources to avoid unnecessary waiting time, and it can realize an effective balance between the data transfer and data processing. Moreover, with a focus on the dynamic network and diverse job parallelism, we further improve the SDTP method by offering more accurate time estimation and generalizing it to dynamic situations.

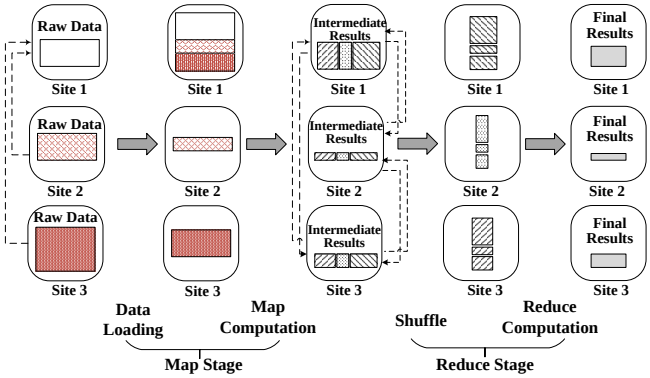


Fig. 3. An illustration of the execution process of a wide-area data analytic job.

3 Modeling and Problem Formulation

In this section, we describe the execution of the wide-area data analytic jobs and formulate the optimal response time problem for such jobs, including the details of calculating the overall time consumption. Table 1 summarizes the major notations used in this paper.

3.1 Execution of Wide-area Data Analytic Job

In this section, we describe how we place tasks in the map and reduce stages to minimize the entire response time of a wide-area data analytic job in the system. The task placement in each stage involves deciding which tasks should be placed on the site and determining the source of the task input data.

In this paper, we focus on the jobs that have exactly one map stage and one reduce stage. We formulate the task placement of the wide-area data analytic job for each stage independently. The map stage includes the input data loading and map computation phases, and the reduce stage is divided into the shuffle transfer and reduce computation phases.

Fig. 3 presents an example of the execution process of a wide-area data analytic job. The 3 sites have different amounts of unprocessed local data. At the data loading phase, the sites with heavy workload transfer some raw data to those sites which have sufficient computing and bandwidth resources. After that, each site executes the map computation on its raw data and generates intermediate results. At the reduce stage, each reduce task needs to read the corresponding intermediate data generated by all map tasks. In the shuffle phase, according to the fraction of reduce tasks performed at each site, each site transfers the intermediate data to corresponding sites. Finally, each site performs the reduce computation to get the final results.

Owing to the heterogeneities of the WAN bandwidth and computing capacity among the geo-distributed sites, the transmission times for obtaining the required input data on different sites are uneven. However, in previous approaches, the sites can execute the map computation only when all of

TABLE 1
Notations and definitions

Notation	Definition
D	set of sites
A_i	amount of the data generated at site i
T^i_{load}	input data loading time of site i
T^i_{shuf}	communication time of the shuffle phase of site i
S_i	number of computation slots at site i
q	ratio of intermediate data to input data
I^j_{shuf}	volume of intermediate data at site j
x^j_i	amount of the data transferred from site j to site i at the map stage
α_i	fraction of reduce tasks to place at each site i
$B^i_{down}(t), B^i_{up}(t)$	uplink/downlink bandwidth of site i at time t
$t^{down}_{i,s}, t^{down}_{i,e}$	start/end time when site i downloads all input data from other sites
$t^{up}_{j,s}, t^{up}_{j,e}$	start/end time when site j uploads input data to other sites
T^i_{map}, T^i_{red}	map/reduce computation time of site i
T_{load}, T_{shuf}	input data loading time and shuffle time
T_{map}, T_{red}	map/reduce computation time
$T^i_{load,down}, T^i_{load,up}$	download/upload time of site i at the input data loading phase
$T^i_{shuf,start}$	start time of site i at the shuffle phase
$T_{shuf,load}$	sum time of the first three phase
$T^i_{shuf,down}, T^i_{shuf,up}$	upload/download time of site i at the shuffle phase

the transmissions are completed. This will idle the distributed sites and lead to unnecessary waiting times. By contrast, in our method, a site can execute its task computation, once it obtains the required input data, to avoid unnecessary waiting time.

3.2 Response Time at Map Stage

At the map stage, the task placement problem involves determining the amount of data x^j_i that should be transferred from site j to site i, and $i, j \in D$, where D is the set of sites and $\mathbf{x}$ represents the set of the data volume that is transferred across all sites. We assume that the map stage contains the input data loading phase and map computation phase, as shown in Fig. 3. Each site should obtain its input data from other sites during the input data loading phase. We suppose that a site can begin to execute its tasks if the data assigned to it has been collected. Let T^i_{load} represent the input data loading time of site i. The map computation time of site i is denoted by T^i_{map}. At this stage, the goal is to minimize the maximum response time among the sites. That is:

$$\min_{\mathbf{x}} \quad \max_{i} \quad T^i_{load} + T^i_{map}. \tag{1}$$

As the bandwidth of the WAN links among the sites may be dynamic, let $B^i_{down}(t)$ represent the download bandwidth of site i at time t, and let $B^i_{up}(t)$ represent the corresponding upload bandwidth of site i at time t. According to $\mathbf{x}$, we can obtain the fraction of map tasks at each site. Therefore, the total volume of data that site i needs to download is $\sum_{j\in D, j\neq i} x^j_i$. We use $t^{down}_{i,s}$ and $t^{down}_{i,e}$ to denote the start and end times when site i downloads all input data from other sites. Let $t^{up}_{j,s}$ and $t^{up}_{j,e}$ denote the start and end times when site j uploads all data to other sites that need to fetch data from site j. Then, we have:

$$\int_{t^{down}_{i,s}}^{t^{down}_{i,e}} B^i_{down}(t)\,dt = \sum_{j\in D, j\neq i} x^j_i \tag{2}$$

$$\int_{t^{up}_{j,s}}^{t^{up}_{j,e}} B^j_{up}(t)\,dt = \sum_{i\in D, j\neq i} x^j_i. \tag{3}$$

From the above two equations, we can determine that the download time of site i at the map stage is $t^{down}_{i,e} - t^{down}_{i,s}$, and the upload time of other sites that contain the input data of site i is $t^{up}_{j,e} - t^{up}_{j,s}$. Thus, the input data loading time of site i is the maximum value between the download time of site i and the upload time of other sites that need to transmit data to site i. Therefore, we have:

$$T^i_{load} = \max_{j\in D}(t^{down}_{i,e} - t^{down}_{i,s}, t^{up}_{j,e} - t^{up}_{j,s}), j \neq i, x^j_i \neq 0. \tag{4}$$

The computation time of tasks at a site is determined by the total volume of data to process, the degree of parallelism of the site, and the job operation processes. Thus, in the map computation phase, we use a function f to estimate the computation time of site i according to the job type, input data size $\sum_{j\in D} x^j_i$, and number of computation slots S_i on site i. That is:

$$T^i_{map} = f(\sum_{j\in D} x^j_i, S_i). \tag{5}$$

Moreover, there is a constraint on the data volume. That is, the sum volume of all data mitigated from site i to others and the data remaining at site i must be equal to the original data size A_i.

$$\sum_{j\in D} x^i_j = A_i, x^i_j \geq 0, \forall i \in D \tag{6}$$

Using the above descriptions, we formulate the task placement problem **P1** at the map stage as follows:

$$\begin{aligned} &\min_{\mathbf{x}} \quad \max_{i} \quad T^i_{load} + T^i_{map} \\ &s.t. \qquad Constraints \quad (2), (3), (4), (5), (6). \end{aligned} \tag{7}$$

3.3 Response Time at Reduce Stage

At the reduce stage, we should decide the fraction α_i of reduce tasks to place on each site i, where $\boldsymbol{\alpha}$ denotes the set of the fraction of reduce tasks on all sites. We suppose that the reduce stage includes the shuffle phase and reduce computation phase, as shown in Fig. 3. In this case, T^i_{shuf} represents the communication time of the shuffle phase at site i. This is the transfer time during which site i obtains its input data from other sites. The reduce computation time on site i is denoted by T^i_{red}. At this stage, the goal is to minimize the maximum response time among the sites; that is:

$$\min_{\boldsymbol{\alpha}} \quad \max_{i} \quad T^i_{shuf} + T^i_{red}. \tag{8}$$

At the shuffle phase, the total amount of data that site i needs to download is $\sum_{j\in D, j\neq i}(I^j_{shuf} \times \alpha_i)$. Furthermore, I^j_{shuf} is the amount of intermediate data on site j. Besides, $t^{down}_{i,s}$, and $t^{down}_{i,e}$ denotes the start and end times when site i downloads its input data from other sites. Let $t^{j,up}_s$ and $t^{j,up}_e$ represent the start and end times when site j uploads all data to the corresponding sites. Hence, we have:

$$\int_{t_{i,s}^{down}}^{t_{i,e}^{down}} B_{down}^{i}(t)\,dt = \sum_{j\in D, j\neq i} (I_{shuf}^{j} \times \alpha_i) \tag{9}$$

$$\int_{t_{j,s}^{up}}^{t_{j,e}^{up}} B_{up}^{j}(t)\,dt = \sum_{i\in D, j\neq i} (I_{shuf}^{j} \times \alpha_i). \tag{10}$$

Similar to the calculation of the time on map stage, the download time of site i at the reduce stage is $t_{i,e}^{down} - t_{i,s}^{down}$, and the upload time of other sites that contain the input data of site i is $t_{j,e}^{up} - t_{j,s}^{up}$. The data shuffle time of site i is equal to the maximum transmission time between the download time of site i ($t_{i,e}^{down} - t_{i,s}^{down}, \alpha_i \neq 0$) and the upload time of other sites ($t_{j,e}^{up} - t_{j,s}^{up}, j \neq i$) that contain the input data of site i . Then, we have:

$$T_{shuf}^{i} = \max_{j\in D}(t_{i,e}^{down} - t_{i,s}^{down}, t_{j,e}^{up} - t_{j,s}^{up}), j \neq i, \alpha_i \neq 0. \tag{11}$$

Following the map stage, the intermediate data from the map tasks on site i are equal to $q \times \sum_{j\in D} x_i^j$, where q denotes the ratio of the intermediate data to the input data of the map stage. Moreover, $\sum_{j\in D} x_i^j$ indicates the input data of the map stage on site i, and it is the sum of the amount of data $\sum_{j\in D, i\neq j} x_i^j$ transferred from site j to site i ($i \neq j$) and the remaining data x_i^i on site i.

$$I_{shuf}^{i} = q \times \sum_{j\in D} x_i^j. \tag{12}$$

The ratio α_i of the reduce task on each site needs to satisfy the following constraint:

$$\sum_{i\in D} \alpha_i = 1, \alpha_i \geq 0. \tag{13}$$

At the reduce computation phase, the input data of the reduce computation on site i is $I_{shuf} \times \alpha_i$, where I_{shuf} is the total amount of intermediate data calculated by the map tasks across all sites. We use a function h to estimate the reduce computation time of site i, according to the job type, input data of the reduce phase, and number of computation slots S_i on site i. Thus, we have:

$$T_{red}^{i} = h(I_{shuf} \times \alpha_i, S_i). \tag{14}$$

Using the above descriptions, we formulate the task placement problem **P2** at the reduce stage as follows:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \quad \max_{i} \quad T_{shuf}^{i} + T_{red}^{i} \\ &s.t. \quad Constraints \quad (9), (10), (11), (12), (13), (14). \end{aligned} \tag{15}$$

By means of the above formulations, we have specified the calculation of the map and reduce stage response times for a wide-area analytic job. However, according to our model, the function of f is usually complicated and non-linear. Therefore, the problems **P1** and **P2** are both non-linear programming problems in general. Besides, the bandwidth of the WAN links among the sites may be dynamic. Thus, it is hard to obtain the optimal solutions in polynomial time.

4 SDTP: Task Scheduling for Wide-Area Data Analytics

In this section, we investigate a more special case and convert the wide-area data analytic problem into a simpler problem. First, we assume that the WAN is static and each site uses a fixed WAN bandwidth. Second, we assume that the computation time of each task is fixed and the computation time can be calculated by a simple formula.

4.1 SDTP at the Map Stage

According to [36], only 7% of jobs in a production MapReduce cluster are reduce-heavy. That is, a reduction in the running time at the map stage is particularly important to minimize the response time of the entire job. As the WAN is static, each site has a fixed upload and download WAN bandwidth. In this section, B_{down}^{i} and B_{up}^{i} represent the download and upload bandwidths of site i, respectively.

Let T_{load}^{i} represent the input data loading time of site i, which is dominated by the maximum transfer time between the download time ($T_{load,down}^{i}$) of site i and upload time ($T_{load,up}^{j}$) of sites that need to transfer data to site i. Thus, the input data loading time of site i can be formulated as Eq. 18. $T_{load,down}^{i}$ is equal to $\sum_{j\in D, i\neq j} x_i^j$ divided by B_{down}^{i}, and $\sum_{j\in D, i\neq j} x_i^j$ is the sum of the amounts of data that need to be transferred to site i from other sites. $T_{load,up}^{j}$ is the maximum value among the upload times of site j that need to transfer data to site i. Thus, we have:

$$T_{load,down}^{i} = \sum_{j\in D, i\neq j} x_i^j / B_{down}^{i}, \forall i \in D \tag{16}$$

$$T_{load,up}^{j} = \sum_{i\in D, i\neq j} x_i^j / B_{up}^{j} \tag{17}$$

$$T_{load}^{i} = \max_{j\in D} \left(T_{load,down}^{i}, T_{load,up}^{j} \right), i \neq j, x_i^j \neq 0. \tag{18}$$

To simplify the computation time calculation, we assume that the computation time of each task is fixed and t_{map} is the computation time of a map task. When the number of tasks on one site exceeds its available compute slots, the tasks will be usually executed with subsequent waves locally and cannot use the idle slots in other sites [3]. For example, site 1 has 50 slots and site 2 has 100 slots. If sites 1 and 2 both compute 100 map tasks, site 1 completes those tasks within two waves and site 2 completes those tasks with one wave. Thus, the computing time of site 1 is twice that of site 2. To this end, the computation time on each site is equal to the number of tasks divided by the degree of parallelism and subsequently multiplied by the execution time of a single task. The map computation time of site i can be formulated as Eq. 19.

$$T_{map}^{i} = \left\lceil \frac{\sum_{j\in D} x_i^j}{S_i} \right\rceil \times t_{map} \tag{19}$$

The goal of the map stage problem is similar to that of Eq. 1. Finally, when the WAN of each site is static and the computation time of each task is also fixed, the task placement problem of the map stage is formulated as problem **P3**:

$$\begin{aligned} &\min_{\mathbf{x}} \quad \max_{i} \quad T_{load}^{i} + T_{map}^{i} \\ &s.t. \quad Constraints \quad (6), (16), (17), (18), (19). \end{aligned} \tag{20}$$

Due to the complexity of the problem, we turn to an approximation algorithm (Algorithm 1), reducing the total response time significantly. The Algorithm 1 iterates to the optimal solution by adjusting map task placement on each site. To accelerate the Algorithm 1 approaching the optimal solution, we formulate the problem **P4** to obtain an initial input for Algorithm 1.

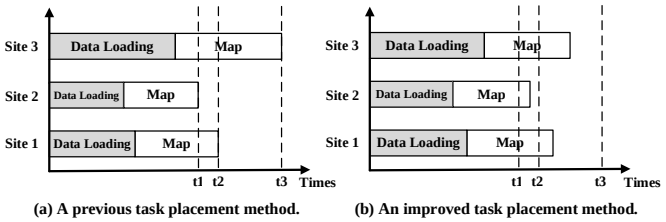


Fig. 4. Different task placement methods.

We assume that the computation of the map tasks must wait for the completion of the data transmission. The input data loading time and map computation time of the map stage are both dominated by the bottleneck site. The objective of this problem can be transformed into Eq. 21. Specifically, the input data loading time in the map stage of this job is equal to the largest input data loading time across all sites (Eq. 22). The map computation time in the map stage of this job is dominated by the maximum computation time across all sites (Eq. 23). Therefore, if the data transfer and the computation cannot be performed simultaneously, the map task placement problem **P4** can be formulated as follows:

$$min \quad T_{load} + T_{map} \tag{21}$$

$$T_{load} = max(T^i_{load}) \tag{22}$$

$$T_{map} = max(T^i_{map}). \tag{23}$$

The problem **P4** is a linear programming problem and it can be solved in polynomial time by existing methods, such as the interior point algorithm or other linear programming algorithms that have been realized by many solvers. With the initial input from **P4** and the following three theorems, we design the Algorithm 1 to achieve map task placement with a reduced job response time. Intuitively, when a site has higher response time than others, we can migrate some tasks from this site to others to reduce the response time of the whole stage. Moreover, when all sites have the same response time at this stage, it means the current task placement is optimal and cannot be further optimized. Based on these intuitions, we formulate 3 theorems which can be proved as follows.

Theorem 1. *Assuming that the input data of this stage can be divided, the response time of a stage is minimized when all sites have the same response time in this stage.*

Proof. Suppose that a task placement scheme exists in which the response times of all sites are not the same, and this task placement scheme has a minimized time in the map stage. An example is presented in Fig. 4(a), where site 3 is the bottleneck site that has the longest response time, and site 2 has the minimum response time. After t1, site 3 is still working, while site 2 is idle.

As all sites can transfer data to one another, if site 3 transfers little data to site 2, and the response time of site 2 and 1 does not exceed the maximum time across other sites, the tasks of site 3 will decrease. The computation time of site 3 will decrease with the decrease in the number of tasks. The transmission time of site 3 is dominated by the maximum transfer time between the download time of site 3 and the upload time of the sites that need to transfer data to site 3. As the download data of site 3 unchanged, the download time of site 3 is unchanged. Similarly, the upload time of the sites which need to transfer data to site 3 is also unchanged. Thus, the transmission time of site 3 will decrease or remain unchanged. Therefore, the response time of site 3 decreased, and the response times of sites 1 and 2 are both less than the response time of site 3. Finally, the response time of this stage will decrease.

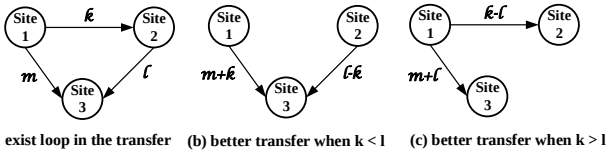


Fig. 5. Improved task placement method.

This example can be easily extended to any number of sites. Similarly, we assume that the response times of the n sites are tunable, and this task placement scheme has a minimized time at this stage. The site with the longest time can either transfer part of its data to other sites or reduce the amount of data received from other sites, to reduce the response time. In this process, it is required that the completion time of other sites does not exceed the completion time of bottleneck site. After that, the response time of this stage can be significantly reduced. Consequently, this result contradicts the assumption, and *Theorem* 1 is proven. □

Theorem 2. *At a given stage, the maximum response time of all sites t^{old}_{max} determines the entire response time of this stage. The decrease of t^{old}_{max} may shorten the gap of all sites' response time, which guarantee the potential of reduce the entire response time.*

Proof. In parallel data analytics, a stage is finished when all sites complete their allocated tasks. Thus, the entire response time is determined by the bottleneck site, and is equal to t^{old}_{max}. When some tasks are transferred from the bottleneck site to other sites, the t^{old}_{max} will be reduced. Consider the simple case in which the tasks of the bottleneck site are only transferred to site i whose response time is minimum across all sites. After the task transfer, if the response time of site i is less than t^{old}_{max}, the response time of site max is also decreased, then the gap of these two sites is narrowed, as well as the gap among all sites. Let t^{new}_{max} denote the maximum response time of the new task placement strategy which follows the above conditions. Based on Theorem 1, t^{new}_{max} is less than t^{old}_{max}, which means the decrease of entire response time. □

Theorem 3. *In the map stage, if site 1 needs to transfer k data to site 2, site 3 requires l data to be transferred from site 2, and site 1 needs to transfer m data to site 3, we can determine other transfer strategies whereby the job response time is less than or equal to the response time of the original transfer strategy, as illustrated in Fig. 5.*

Proof. Assume that the upload and download bandwidths of site i are B^i_{up} and B^i_{down}, respectively. Hence, in Fig. 5(a), the transfer time of site 1 is 0, because it does not need to obtain data from other sites. The transfer time of site 2 is $max((k+m)/B^1_{up}, k/B^2_{down})$ and the transfer time of site 3 is $max(l/B^2_{up}, (m+k)/B^1_{up}, (l+m)/B^3_{down})$. However, if we adopt other strategies, as illustrated in Fig. 5(b) and Fig. 5(c), the job response time will be less than or equal to the response time of Fig. 5(a). If $k < l$, the transfer

Algorithm 1: SDTP at the map stage.

1 Obtain **x** by solving **P4** based on A_i, B^i_{down}, B^i_{up}, and S_i;
2 Calculate the map response time $T^i_{e,m}$ of each site, bottleneck site m, and difference ratio r;
3 Call $decreaseInputData(T^i_{e,m}, \mathbf{x})$ to obtain new **x**;
4 Call $equalResponseTime(T^i_{e,m}, \mathbf{x})$ to obtain new **x**;
5 **return x**;

6 **Function** *decreaseInputData*($T^i_{e,m}$, ***x***)
7 $T = T^m_{e,m}$;
8 **while** $r \geq \beta$ and $T^m_{e,m} \leq T$ **do**
9 $T = T^m_{e,m}$;
10 Obtain the sites I where $x^l_m > 0, l \in I$;
11 **for** l in I **do**
12 Reduce x^l_m by $\gamma\%$;
13 Calculate the new map response time $T^i_{e,m}$ of each site, bottleneck site m, and difference ratio r;
14 **return x**, $T^i_{e,m}$;
15 **Function** *equalResponseTime*($T^i_{e,m}$, ***x***)
16 $T = T^m_{e,m}$;
17 **while** $r \geq \beta$ and $T^m_{e,m} \leq T$ **do**
18 $T = T^m_{e,m}$;
19 Sort $T^i_{e,m}$ and divide $T^i_{e,m}$ into two groups, G_l, G_s;
20 Let $T^i_{e,m} = T^j_{e,m}, l \in G_l, j \in G_s$;
21 Check whether a loop exists and update **x**;
22 Calculate the new map response time $T^i_{e,m}$ of each site, bottleneck site m, and difference ratio r;
23 **return x**, $T^i_{e,m}$;

time of sites 1 and 2 is 0, and the transfer time of site 3 is $max((m+k)/B^1_{up}, (l-k)/B^2_{up}, (l+m)/B^3_{down})$. The transfer times of the three sites are all less than or equal to the transfer time in Fig. 5(a) and the computation times of the three sites do not change. If $k > l$, the transfer time of site 1 is 0, the transfer time of site 3 is $max((m+k)/B^1_{up}, (l+m)/B^3_{down})$, and the transfer time of site 2 is $max((k-l)/B^1_{up}, (k-l)/B^2_{down})$. In this case, the transfer times of the three sites are also all less than or equal to the transfer time in Fig. 5(a), and the computation times of the three sites do not change. Therefore, *Theorem* 3 is proven. □

Based on the above theorems, we propose Algorithm 1 to accelerate the wide-area data analytics by balancing the response time of each site. First, we can solve problem **P4** by some classical linear programming algorithms (e.g., Ellipsoid method, Interior point method, Simplex method, etc.) to determine the preliminary data transmission scheme (step 1). Thereafter, the algorithm calculates the response time $T^i_{e,m}$ for each site. $T^i_{e,m}$ is the sum of the transfer time of site i and the computation time of site i (step 2). Subsequently, it calls function $decreaseInputData()$, which attempts to adjust the data transmission scheme for reducing the map response time (step 3). The process is repeated until $r < \beta$ or T^m_e decreases no more, where β is the expected maximum difference ratio of the response time across all sites, and r is the actual difference ratio among the response time of all sites. For instance, Let t_2 and t_1 denote the maximum response time and the minimum response time, respectively. Then, the actual difference ratio r is $(t_2 - t_1)/t_1$. In this process, the algorithm attempts to reduce the response time of site max at the map stage by decreasing its input data, as in *Theorem* 2. The amount of decreased data of each step is $\gamma\%$ (step 12), and γ is the adjusting step size. Specifically, if site i transfers x^{max}_i data to site max in the original scheduling scheme, site i will decrease the volume of data to $x^{max}_i \times (1 - \gamma\%)$ in this process.

Thereafter, to reduce the response time of the job at the map stage further, we design another function $equalResponseTime()$. This function sorts the response times of all sites and divides each site into two groups, G_l and G_s, where each group has the same number of sites (step 19). Next, it matches the items of the two groups one by one and enables the matched sites to obtain the same response time (step 20). For example, we assume that sites i and j are matched, and site i has a larger response time. To obtain the same response time, site i needs to transfer some data to site j. Assuming that the amount of transfer data is y, the following equation is solved: $(A_i - y)/s_i \times t_{map} + T^i_{load} = T^j_{load} + x/max(B^i_{up}, B^j_{down}) + (A_j - y)/s_j \times t_{map}$. Thus, site i needs to transfer y GB of data to site j. Subsequently, we verify whether the data transmission scheme depicted in Fig. 5(a) exists in the calculated data transmission scheme (step 21). Similarly, the process is repeated until $r < \beta$ or T^m_e no longer decreases. Finally, the map task placement solution **x** is returned, which can achieve a reduced job response time.

4.2 SDTP at the Reduce Stage

At the reduce stage, the shuffle time T^i_{shuf} of site i can be formulated as Eq. 26. It is the maximum value among the download time of site i to obtain all input data ($T^i_{shuf,down}$) and the upload time of several sites that need to upload a certain ratio of data to site i ($T^j_{shuf,up}$). The reduce computation time of site i is presented in Eq. 27, and t_{red} is the execution time of a reduce task.

$$T^i_{shuf,down} = \sum_{j \in D, j \neq i} (I^j_{shuf} \times \alpha_i)/B^i_{down} \tag{24}$$

$$T^j_{shuf,up} = \sum_{i \in D, j \neq i} (I^j_{shuf} \times \alpha_i)/B^j_{up} \tag{25}$$

$$T^i_{shuf} = \max_{j \in D}(T^i_{shuf,down}, T^j_{shuf,up}), j \neq i, I^j_{shuf} \times \alpha_i \neq 0 \tag{26}$$

$$T^i_{red} = \left\lceil \frac{I_{shuf} \times \alpha_i}{S_i} \right\rceil \times t_{red}. \tag{27}$$

Hence, in this case, the task placement problem **P5** of the reduce stage is formulated as follow:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \quad \max_i \quad T^i_{shuf} + T^i_{red} \\ &s.t. \quad Constraints \quad (12), (13), (24), (25), (26), (27). \end{aligned} \tag{28}$$

Owing to the complexity of the above problem, we continue with the simplification. We also require that all reduce tasks are executed after the shuffle phase. Thus, the goal of the reduce stage is transformed into Eq. 29, the shuffle time of the job is denoted by Eq. 30, and Eq. 31 represents the reduce computation time of the job. The task

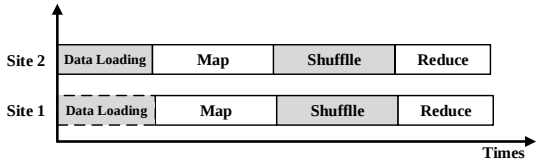


Fig. 6. Execution process of previous method.

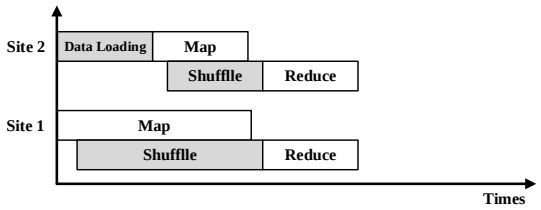


Fig. 7. Execution process of SDTP.

placement problem **P6** of the reduce stage can be formulated as follows:

$$min \quad T_{shuf} + T_{red} \tag{29}$$

$$T_{shuf} = max(T^i_{shuf}), i \in D \tag{30}$$

$$T_{red} = max(T^i_{red}), i \in D. \tag{31}$$

This is a linear programming problem, and it can be solved in polynomial time by existing solvers. However, the job response time of the above formulation is not sufficiently small. Owing to the insufficient computing capacities, the map tasks of one site cannot be executed at the same time. Only a part of the tasks can be executed simultaneously, and others have to wait. That is, the tasks will be executed on different waves. Once the tasks are completed, the intermediate data generated by those map tasks can be transferred to the other sites, which will execute corresponding reduce tasks using the generated intermediate data. That is, the map phase is CPU intensive, and the shuffle phase is I/O intensive, and the map computation phase may overlap with the shuffle phase. However, the shuffle phase of a job must start later than its map phase, and it cannot finish earlier than its map stage. This is because the shuffle phase must wait to transfer the intermediate data calculated by the map phase.

Thus, we can formulate the job response time by overlapping the map computation and shuffle phases. $T_{shuf,load}$ represents the sum time of the first three phases of the job. Once the first wave of map tasks in a site has been completed, the intermediate data of those map tasks generated on this site and intermediate data can be transferred to other sites, which will execute its reduce tasks after obtaining all corresponding intermediate data. Therefore, the start time of the shuffle phase on site i ($T^i_{shuf,start}$) is equal to T^i_{load} + t_{map}, and T^i_{load} can be calculated by Algorithm 1. The sum time of the input data loading phase, map computation phase, and shuffle phase on site i is $T^i_{shuf} + T^i_{shuf,start}$. The time of the first three phases of the entire job is the maximum value among $(T^i_{shuf} + T^i_{shuf,start}), i \in D$ and it can be formulated as Eq. 33. Our objective is to minimize the job response time (Eq. 32).

$$min \quad T_{shuf,load} + T_{red} \tag{32}$$

$$T_{shuf,load} = \max_{i \in D}(T^i_{shuf} + T^i_{shuf,start}) \tag{33}$$

$$T^i_{shuf,start} = T^i_{load} + t_{map}, i \in D. \tag{34}$$

Algorithm 2: SDTP at the reduce stage

```
1  Calculate T^i_load, T^i_map by Algorithm 1;
2  Obtain the shuffle start time based on T^i_load, t_map;
3  Determine α by solving P7;
4  Calculate the new response time T^i_e of each site,
     maximum site m, minimum site s, and difference ratio
     r;
5  Call getRedTaskPlace(T^i_e, α, m, s) to obtain new α ;
6  return α;

7  Function getRedTaskPlace(T^i_e, α, m, s)
8      T = T^m_e;
9      while r ≥ β and T^m_e ≤ T do
10         T = T^m_e;
11         Update the reduce ratio α_m − α_m × γ%,
             α_s + α_m × γ%;
12         Calculate the new response time T^i_e of each site,
             maximum site m, minimum site s, and
             difference ratio r;
13     return α;
```

Therefore, the task placement problem **P7** can be formulated as follow:

$$\begin{aligned} min \quad & T_{shuf,load} + T_{red} \\ s.t. \quad & Constraints \quad (18), (26), (31), (33), (34). \end{aligned} \tag{35}$$

Considering the above characteristics, we design an algorithm known as SDTP for reducing the response time of the entire job. The basic concept is depicted in Fig. 7. In contrast, the usual geo-distributed data analytics execution process is presented in Fig. 6. The job can only start a new phase when the previous phases on all sites have been completed. Thus, massive resources are idle during job processing. SDTP starts the data processing as soon as possible and attempts to make full use of the resources of the sites.

Specifically, as shown in Algorithm 2, the input data loading time and map computation time at the map stage are first obtained for each site by **x**, which calculated by Algorithm 1 (step 1). Moreover, the start time of the shuffle phase is determined according to T^i_{load} and t_{map} (step 2). Thereafter, the model of problem **P7** is formulated. In problem **P7**, the map and shuffle phases will overlap. The linear programming problem **P7** is solved using a linear programming method to obtain the original reduce task placement solution (step 3). Thereafter, the algorithm obtains the difference ratio of the reduce tasks across all sites and attempts to decrease the job response time by decreasing the ratio of the reduce tasks on the bottleneck site. It calls the function $getRedTaskPlace()$, which attempts to mitigate part of the reduce tasks from the straggler site, which leads to the highest response time to the most vacant site with the lowest response time. The rate of each adjustment is $\gamma\%$. That is to say, the ratio of reduce tasks at the site m and site s change to $\alpha_m - \alpha_m \times \gamma\%$ and $\alpha_s + \alpha_m \times \gamma\%$, respectively.

In Algorithm 1, the iteration times of the **while** loop in lines 17 to 22 is a constant k, and the complexity of the sort in line 19 is $O(n \log n)$. Thus, the complexity of the function *equalResponseTime()* is $O(kn \log n)$. Problem **P4** is a linear programming problem, which can be solved by normal solvers in polynomial time. Specifically, we adopt

Algorithm 3: SDTP+

1 Build models $m1(), m2()$ to predict the map response time and reduce the response time;
2 Calculate **x** by solving problem **P4**;
3 Obtain the map response time $T^i_{e,m}$ by $m1()$, **x**, bottleneck site m, and difference ratio r;
4 Call $decreaseInputData(T^i_{e,m}, \mathbf{x})$ to obtain new **x** ;
5 Follow steps 2 to 5 in Algorithm 2 to obtain new $\boldsymbol{\alpha}$;
6 **return** $\mathbf{x}, \boldsymbol{\alpha}$;

the interior point method to solve problem **P4** with the time complexity $O(n^{3.5}L^2)$ [37], where n represents the number of variables, and L denotes the scale of the problem. Thus, the complexity of step 1 is $O(n^{3.5}L^2)$ and is higher than that of function *equalResponseTime()*. Therefore, the overall complexity of Algorithm 1 is $O(n^{3.5}L^2)$. In Algorithm 2, the most time-consuming process is calling Algorithm 1 for execution. Thus, the complexity of Algorithm 2 is also $O(n^{3.5}L^2)$.

5 Further Improvement of SDTP

In this section, the dynamic nature of the WAN bandwidth and the influence of the degree of parallelism are considered. The ever-changing WAN bandwidth and degree of parallelism in the parallel computing will seriously affect the job response time in wide-area data analytics. In this section, the computation time of tasks at each site is predicted by the nonlinear regression algorithm. Based on the more accurate time estimation, we formulate a suitable task scheduling scheme to optimize the job response time. This scheme can be generalized to dynamic situations.

5.1 Challenge of Accurate Time Estimation

A job with large input or large intermediate data can efficiently harness additional parallelism; in contrast, a job running on small input data or with less efficiently parallelizable operations will obtain few gains from extra parallelism. Therefore, it is necessary to formulate the relationship of the job running time to the size of the input data and degree of parallelism.

For recurring jobs, the running time and intermediate data sizes can be reasonable predicted [24], [38], [39]. In this section, we use the multiple nonlinear regression algorithm to predict the job running time. Based on these predictions, the job running time can be further reduced. As we calculate the task placement solution separately in each stage, we construct the related model for the running time in each individual stage. At the map stage, the task placement problem **P8** can be formulated as:

$$\begin{aligned} &\min_{\mathbf{x}} \quad \max_{i} \quad T^i_{load} + T^i_{map} \\ &s.t. \qquad Constraints \quad (5), (6), (16), (17), (18). \end{aligned} \tag{36}$$

At the reduce stage, the task placement problem **P9** can be formulated as:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \quad \max_{i} \quad T^i_{shuf} + T^i_{red} \\ &s.t. \quad Constraints \quad (12), (13), (14), (24), (25), (26). \end{aligned} \tag{37}$$

We first build the prediction models *m1()* and *m2()* to predict the map time and reduce the response time on each site (step 1). Next, we calculate the initial solution **x** by solving **P4** (step 2). Thereafter, we use the function $decreaseInputData(T^i_{e,m}, \mathbf{x})$ in Algorithm 1 to obtain a better transmission scheme at the map stage (step 4). In this function, we use the prediction model *m1()* to calculate the map computation time on each site. Subsequently, we follow Algorithm 2 from steps 2 to 5 to obtain the final task placement solution. In these steps, we also use the prediction model *m2()* to calculate the map computation time on each site.

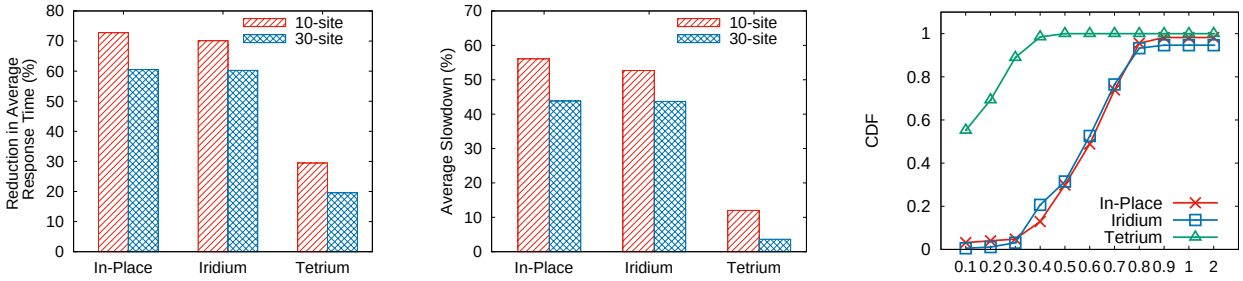
Algorithm 4: SDTP++

1 Calculate $\boldsymbol{x}, \boldsymbol{\alpha}$ by Algorithm 3;
2 Obtain the response time T^i_e of each site by $\boldsymbol{x}, \boldsymbol{\alpha}$;
3 Measure the new WAN bandwidth $B^i_{d,n}, B^i_{u,n}$ with the interval t_{inter};
4 **if** the change in the WAN $> \rho$ **then**
5 **if** $t_c <= t^{min}_{load}$ **then**
6 Obtain the new input data A_i by **x**, t_c;
7 Calculate $\boldsymbol{x}, \boldsymbol{\alpha}$ by Algorithm 3;
8 **if** $t_c > t^{min}_{load}$ and $t_c < t^{min}_{map}$ **then**
9 Get $\boldsymbol{\alpha}$ by getRedTaskPlace($T^i_{e,m}, \mathbf{x}, B^i_{d,n}, B^i_{u,n}, S_i$);
10 **return** $\boldsymbol{x}, \boldsymbol{\alpha}$;

5.2 Challenge of Dynamic Network Bandwidth

In addition to the scarcity, large variances exist in the WAN bandwidth. The dynamic WAN bandwidth will significantly affect the response time of the wide-area data analytic job. For example, one site may contain sufficient bandwidth and rich computing resources, and thus, it can execute a large number of computation tasks of job A. During the execution process of job A, the WAN bandwidth of the site decreases sharply and the site have to spend more time to transfer data to obtain the input data of job A. As a result, if the original task placement strategy is not changed, the job response time of job A will increase dramatically. Thus, it is necessary to consider the dynamic nature of the WAN bandwidth, especially for batch jobs with a long response time.

Focusing on the dynamic WAN bandwidth, we design a task placement update module, which provides a bandwidth detection component to detect the bandwidth of each site with a given interval. When the variation in the WAN bandwidth exceeds ρ, the module will change the task placement solution according to the job execution state and the variation in the WAN bandwidth. When the job is in the input data loading phase, we first calculate the data amount of each site, and subsequently determine a new task placement at the map and reduce stages using Algorithm 3 (steps 5 to 7). If the job is in the map computation phase, we continue to complete the map computation phase, and thereafter calculate a new task placement at the reduce stage using function $getRedTaskPlace()$ in Algorithm 3 (steps 8 to 9). In this algorithm, we use the prediction model to predict the reduction in the computation time of the job on different sites. If the current time t_c is larger than t^{min}_{map}, we do nothing, because the shuffle phase will transfer special data to the corresponding sites. A change in the



(a) Average response time. (b) Average slowdown. (c) CDF in slowdown when $|D| = 10$.

Fig. 8. Reduction in average response time, the average slowdown and CDF in slowdown compared to other approaches under different numbers of sites.

task placement will seriously affect the reduce response time. The specific steps of the algorithm are presented in Algorithm 4.

In Algorithm 4, there is no loop, and the most time-consuming process is calling Algorithm 3 for execution. Thus, the complexity of Algorithm 4 is dominated by Algorithm 3. In Algorithm 3, the function *decreaseInputData()* is called for execution. In decreaseInputData(), the iteration times of the **while** loop in lines 8 to 13 is a constant k and the nested **while** loop in lines 11 to 12 is iterated p times, where p is the number of sites. Hence, the complexity of function *decreaseInputData()* is $O(kp)$. It is also lower than the complexity of solving the problem **P4** in step 2. To this end, the complexity of Algorithm 4 and Algorithm 3 is $O(n^{3.5}L^2)$.

6 Performance Evaluation

In this section, we discuss the comprehensive evaluations that were conducted to measure the performance of our methods using the Google dataset [40], [41] and the Alibaba dataset [42].

6.1 Experiment Settings

We construct two wide-area analysis environments with 10 and 30 geo-distributed heterogeneous sites, respectively. The resource capabilities are set according to Amazon EC2. More precisely, the bandwidth of each inter-site link ranges from 100 Mbps to 2 Gbps, and the number of slots on each site ranges from 10 to 100. Moreover, by default, we set the ratio between the intermediate data and input data q as 0.5, the excepted difference ratio among response times of all sites β as 0.1, and the adjusting step size γ as 5. The execution time of a map task t_{map} ranges from 10 to 120s and the execution time of a reduce task t_{red} ranges from 5 to 60s.

We use realistic trace data sets from Google and Alibaba to emulate the geo-distributed data analytic jobs. The Google trace [40], [41] collects the information of machines, jobs, and tasks in a data center with 12.5k machines. The events of the machines, jobs, and tasks are all described by one or more records. Each record generally contains meta-information such as the timestamp, ID, event type, and resource request. The Alibaba trace [42] was published by the Alibaba Group in 2018. It contains records regarding 4k machines over a period of eight days. This trace includes many types of batch workloads, most of which are DAG jobs. The machines in Google trace and Alibaba trace are randomly divided into 30 and 10 sites, respectively. Thus, the workload of each site is combined by the tasks distributed on corresponding machines.

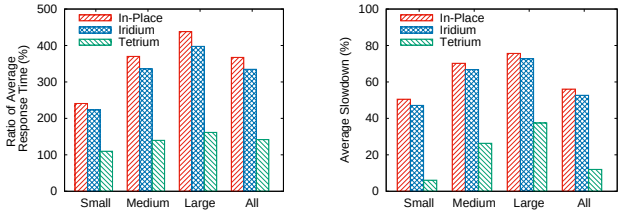
We compare our approach with the following methods in our evaluations.

- **In-Place:** The default Spark approach which runs tasks locally according to the input data placement and assigns tasks evenly to all sites in the shuffle phase.
- **Iridium:** A recent method that improves the job response time by shuffle-optimized reduce task placement for geo-distributed jobs.
- **Tetrium:** A state-of-the-art approach in recent years, which aims to optimize the placement of the input data and reduce tasks, as well as improve the job response time.

6.2 Performance of Our Approach

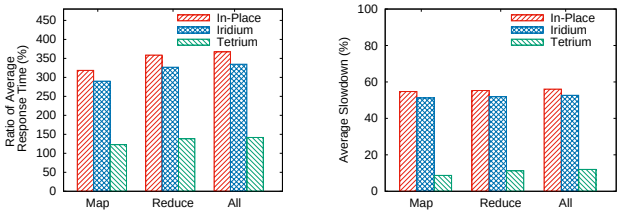
We first evaluate the performance of our approach, namely SDTP, by comparing it with several classical task placement approaches in terms of the average response time and average slowdown. We present the results of the reduction in the average response time and reduction in the average slowdown compared to various approaches. The slowdown is defined as the reduction ratio of the response time of a single job compared to that of other approaches. For instance, the response time of job A using In-place is t_1 and the response time of job A using SDTP is t_2; thus, the slowdown of job A compared to the response time using In-place is $(t_1 - t_2)/t_1$. The average slowdown is the sum of all slowdowns of each job divided by the number of jobs.

Fig. 8(a) presents the improvement of SDTP on the average job response time under different numbers of sites. From this figure, we can find that SDTP outperforms the other baseline methods significantly. In particular, when the number of sites is 10, our method reduces the average job response time of all job types by 72%, 70%, and 29% compared to In-Place, Iridium, and Tetrium, respectively. Thus, our approach can effectively reduce the job response time. When the number of sites is 30, our method reduces the average job response time of all job types by 61%, 60%, and 19% compared to In-Place, Iridium, and Tetrium, respectively. The reductions in the average response time under the 10-site setting are more significant than those under the 30-site setting. This is because more sites result in more computing resources being required to process the job, and the overlapping time of the map computation and



(a) Average response time. (b) Average slowdown.

Fig. 9. Ratio of average response time and the average slowdown on diverse scales.



(a) Average response time. (b) Average slowdown.

Fig. 10. Ratio of average response time and the average slowdown on different components.

shuffle phases is reduced. Thus, the reduction in the average response time decreases with the increase in the number of sites.

Fig. 8(b) also presents the reduction in the slowdown compared to In-Place, Iridium, and Tetrium when the number of sites is varied. When the number of sites is 10, our approach can reduce the average response time for each job by 56% compared to the In-Place method. Thus, SDTP can reduce the job response time for most of the wide-area data analytic jobs. Similarly, with the increase in the number of sites, the reduction in the average slowdown decreases. This is also because with the increase in the number of sites, the sum of the computing resources required to process the job increases, and the overlapping time of the map computation and shuffle phases decreases. Thus, the reduction in the average slowdown decreases when the number of sites increases. Furthermore, the reductions in the average response time are less than the reductions in the average slowdown according to the different baselines. This demonstrates that SDTP is more effective for jobs with a long response time.

Fig. 8(c) presents the CDF in the slowdown compared to other approaches when the number of sites is 10. It can be observed that the slowdown mainly ranges from 0.1 to 0.8. Besides, compared with Iridium and In-Place, SDTP can reduce the response time of almost all jobs by at least 10%, and can reduce the response time of 50% jobs by 10% to 70% compared to Tetrium. That is to say, SDTP can effectively reduce the response time of most of the wide-area data analytic jobs compared with other approaches.

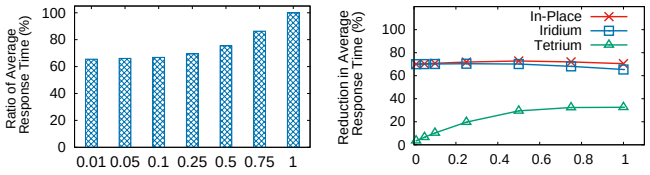
Thereafter, we evaluate the improvement in the average response time on jobs of different scales compared to In-Place, Iridium, and Tetrium. We classify all jobs as small-scale, medium-scale or large-scale jobs according to the volume of input data that they required. If the amount of input data of a job is no greater than 60 GB, it is regarded as a small-scale job. If the amount of input data of a job is greater than 60 GB and no greater than 600 GB, it is classified as a medium-scale job. Otherwise, it is a large-scale job. In this experiment, the number of sites is set to 10.

Fig. 9(a) presents the ratio of average response time between baselines and our approach on diverse scales. This figure demonstrates that the response time increases with the increase of job scale. That is, our approach is more effective for time-consuming jobs. This is because with the increase in the scale of the jobs, SDTP can use more idle resources to transfer data and process tasks. At the map stage, the time-consuming jobs usually result in a greater difference in the sum time of the data transmission and map task execution across all sites. Thus, SDTP can use idle resources to balance the sum time of the data transmission and map task execution across all sites, thereby reducing the time more. At the reduce stage, larger jobs result in longer map computation and shuffle times, and the job response time can be reduced further by overlapping the map computation and shuffle phases.

Fig. 9(b) shows the slowdown of the job response time with the increase in the scale of the jobs. The reduction in the average slowdown ranges between 37% and 75% for large jobs. Thus, our approach can effectively reduce the job response time when the job scale is large. The reduction in the average slowdown decreases with the increase in the job scale. This is because with the growth in the amount of input data, the response time of the jobs increases and the time that SDTP can optimize becomes large.

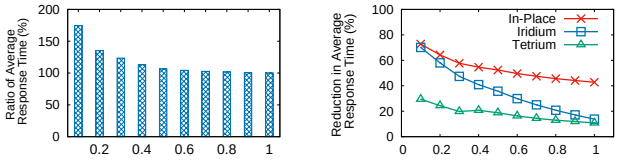
Thereafter, we evaluate the influence of the average response time on different components compared to In-Place, Iridium, and Tetrium. Fig. 10(a) presents the ratio of average response time between baselines and our approach with different components. When using only Algorithm 1 at the map stage, the ratios of average response time under In-Place, Iridium, and Tetrium compared to SDTP are 318 %, 290%, and 123%, respectively. When using only Algorithm 2 at the reduce stage, the reduction in the average job response time is greater than the reduction in the average job response time when using Algorithm 1. That is, Algorithm 2 can reduce job response time more than Algorithm 1. When using Algorithms 1 and 2 simultaneously, the reduction in the average job response time is greater than the reduction in the average job response time when using only one algorithm. That is, using Algorithms 1 and 2 simultaneously results in less job response time. Furthermore, the reduction in the average job response time using Algorithms 1 and 2 simultaneously is less than the sum of the reduction in the average job response time when using Algorithm 1 and Algorithm 2 separately. This is because optimizing the map stage with Algorithm 1 definitely changes the input of the reduce stage, which decreases the performance of Algorithm 2, since Algorithm 1 makes the data distribution more balanced among different sites.

Fig. 10(b) depicts the reduction in the average slowdown compared to the other approaches. It can be observed that the reduction in the average slowdown when using Algorithm 2 is slightly greater than the slowdown when using Algorithm 1. This demonstrates that Algorithm 2 is more effective in reducing the job time. Furthermore, when using Algorithms 1 and 2 simultaneously, the reduction in the average slowdown is greater than the reduction in the average slowdown when using only one algorithm.



(a) Ratio of average response time. (b) Reduction in average response time.

Fig. 11. Influence of ratio of q.



(a) Ratio of average response time. (b) Reduction in average response time.

Fig. 12. Influence of number of slots.

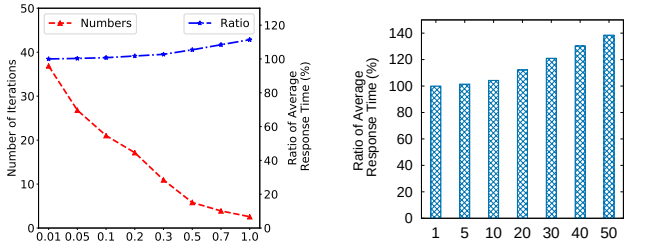
6.3 Impact of Varied Parameters

In this section, we quantify the impact of diverse parameters on SDTP, including the ratio of the intermediate data to the input data at the map stage q, the number of slots, the expected maximum difference ratio β and the adjusting step size γ.

Fig. 11(a) depicts the influence of q. The figure indicates the ratio of the response time to T for different q values, where T is the response time when $q = 1$. It can be observed that the job response time increases with the increase in q. This is because a larger q will produce more intermediate data. Transmitting intermediate data at the shuffle phase and handling intermediate data at the reduce stage can both increase the overall response time.

Fig. 11(b) illustrates the reduction in the average response time with different q values compared to In-Place, Iridium, and Tetrium. It can be observed that, with the increase in q, the reduction in the average response time increases compared to Tetrium, whereas the reduction in the average response time is relatively stable compared to In-Place and Iridium. The reason for this is that, with the increase in q, the amount of intermediate data increases. SDTP can effectively reduce the response time at the reduce stage by overlapping the map computation and shuffle phases compared to Tetrium. As the job response time of In-Place and Iridium is very long, SDTP can make full use of the idle resources to reduce the job response time, and thus, the reduction in the average response time is always larger than those of In-Place and Iridium.

Different numbers of slots will also affect the job response time. Fig. 12(a) indicates that the reduction in the average job response time decreases with the increase in the number of slots. In this experiment, the number of slots on each site ranges from 100 to 1000 when the ratio of the slot number is 1. When the ratio of the slot number is 0.1, the number of slots on each site is equal to the number of slots when the ratio of the slot numbers is 1 multiplied by 0.1. Thus, the number of slots increases with the increase in the ratio of the slot numbers. The figure indicates the ratio of the response time to T with different ratios, where T is the response time when the ratio of slot numbers is 1. It can be observed that the job response time is reduced with the increase in the number of slots. When a site has more slots, it can process the map and reduce tasks more rapidly, and thus, the time of the map and reduce computation phases is decreased.



(a) Influence of β. (b) Influence of γ.

Fig. 13. The influence of β and γ.

The reduction in the average response time with different numbers of slots compared to In-Place, Iridium, and Tetrium is illustrated in Fig. 12(b). It can be observed that the reduction in the average response time decreases with more slots. This is because when sites have more slots, the map and reduce computation times decrease, and thus, the time that SDTP can improve is limited, particularly at the reduce stage.

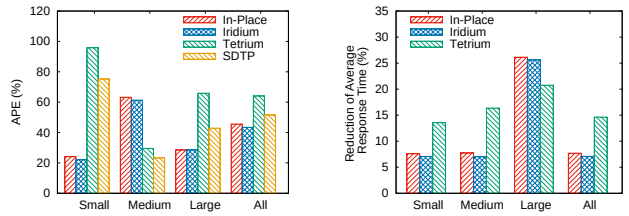
Finally, we measure the influence of the expected maximum difference ratio β and the adjusting step size γ. Fig. 13(a) shows the influence of β. We count the number of iterations and demonstrate the ratio of the response time to T for different β values, where T is the response time when $\beta = 0.01$ at the map stage. With the increase of β, the average response time grows rapidly. Specifically, the average response time when $\beta = 1$ is 10% higher than that when β is set as 0.01. That is to say, the trend of the reduction of the response time increases with the decline of β. On the other hand, the smaller value of β, the more iterations the algorithms required meaning the longer execution time. To better trade-off the efficiency and performance of algorithm 1 at the same time, we assign β as 0.1. When $\beta = 0.1$, the average response time is only 0.75% less than that when $\beta = 0.01$. Moreover, the average iteration amounts for scheduling one job when $\beta = 0.1$ is about 20 times less than that when $\beta = 0.01$.

Fig. 13(b) shows the influence of adjusting step size. It presents the ratio of the average response time T for different γ, where T is the average response time when $\gamma = 1$. With the increase of γ, the average response time also increases. This indicates that using smaller γ results in less job response time.

6.4 Impact of Parallelism

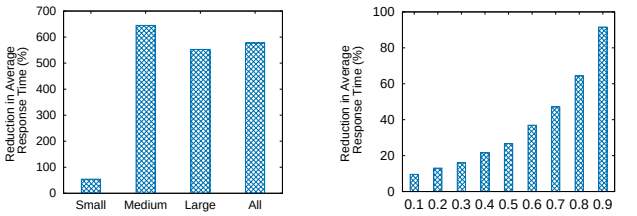
Considering the impact of parallelism in parallel computing, we first evaluate the accuracy of our prediction method on the response time at different stages. Thereafter, we modify our algorithms and other benchmarks by computing more accurate computation time with our prediction method, and then analyze the deviation of the unmodified approaches from the actual values. Finally, we evaluate the performance of SDTP+.

We measure the time of multiple queries with different data amounts and degrees of parallelism running on Spark using BigDataBench [25]. Based on the results, we use



(a) Absolute percentage errors. (b) Average response time.

Fig. 14. Absolute percentage errors and reduction in average response time with prediction computation time.



(a) Range from 0.1 to 2 GB/s. (b) Different fluctuation ranges.

Fig. 15. Reduction in average response time when WAN bandwidth changed.

the multiple linear regression algorithm to construct the prediction model for the computation time in each stage. The results demonstrate that the R_2 statistics are all larger than 0.9, where R is the correlation coefficient. The value of the F-statistic is larger than the value according to the F distribution table. The probabilities p corresponding to the F-statistics are all less than 0.0001. That is, a strong correlation exists between the amount of input data and degree of parallelism, and thus, the prediction model is effective.

To quantify the impact of the degree of parallelism, we analyze the absolute percentage error (APE) of different approaches. The APE is calculated as $APE = |T - T_p|/T * 100\%$. For example, T is the average response time of the In-Place algorithm, and T_p is the average response time when the computation time is calculated by the prediction model in the In-Place algorithm. Fig. 14(a) presents the APEs of the different algorithms for varying job scales. The APEs of the algorithms are all greater than 20%.

Thus, if we only use the times of the map and reduce tasks and the number of computation slots to calculate the computation time at different stages, the final results of the job response time will differ significantly from the actual job response time. Moreover, the APE of Tetrium is larger than those of the other approaches because Tetrium considers the influence of the heterogeneity of computing resources on different sites. The computation time for each site in Tetrium is equal to $V/S_i \times t$, where V is the amount of data and t is the execution time of a single task.

The reduction in the average response time on different job scales compared to the In-Place, Iridium, and Tetrium methods is illustrated in Fig. 14(b). It can be observed that the improvement in the average response time of all jobs is between 7% and 14%. Furthermore, when the scale of jobs is large, the improvement in the average response time of all jobs is between 20% and 26%. That is, the reduction in the average response time increases with the increase in the amount of data. Note that, due to the limitations of our experiment environment, the size of input data used in this experiment is no more than 100GB per site, which means that many large-scale jobs are not considered. However, as shown in Fig. 9(a), the larger size of input data, the greater advantages our algorithms have over other benchmarks. If these large-scale jobs are considered in this experiment, the advantages of our algorithms will become greater.

6.5 Dynamic Bandwidth

Fig. 15(a) presents the influence of the dynamic WAN links on the job response time. The changed bandwidths of all sites are randomly generated between 0.1 and 2 GB/s. We measure the reduction in the average response time of SDTP++ compared to SDTP+ when the WAN bandwidth changed on different job scales. It can be observed from Fig. 15(a) that there is a large reduction in the average response time. Thus, if the WAN bandwidth is varied while the task placement remains unchanged, the job response time will become very long. When the scale of the job is medium or large, the reduction in the average response time is very large compared to that of small-scale jobs. Thus, SDTP++ is more effective for time-consuming jobs.

The reduction in the average response time with different WAN bandwidths is illustrated in Fig. 15(b). In this figure, the value of the abscissa δ is the difference in the WAN bandwidth. For example, when δ is 0.1, the WAN bandwidth B_i of each site range from $B_i \times 0.9$ to $B_i \times 1.1$. It can be observed that the reduction in the average response time increases with an increase in δ. The reason for this is that when δ increases, a link with sufficient bandwidth may become the bottleneck link, and the original data transmission scheme will cause a longer transmission time.

7 Conclusions and Future Work

Cloud service providers and research institutes deploy data centers or edge clusters globally, which generate large volumes of data across geo-distributed locations. We have proposed a novel scheduling mechanism known as SDTP for wide-area data analytics. This method attempts to balance the data transfer and task computation, and begins the tasks as early as possible. Moreover, SDTP provides more accurate time estimation and can be generalized to dynamic situations. The evaluation results demonstrate that SDTP can outperform existing state-of-the-art methods and significantly improves the job response time.

In future work, we plan to realize our method on popular big data frameworks. There are mainly two challenges to do so. Firstly, in the current big data frameworks (e.g. Hadoop, Spark, etc.), the tasks of each stage are executed when all tasks obtain their required input data. However, in our approach, we assume that a site can execute task computation once it gets its required input data. Thus, how to realize a new task scheduling component to satisfy our requirement is challenging. Secondly, with the sites distributed across different regions, task failures are more likely to occur due to the unstable wide-area network. The task failures may lower the performance of our methods. Thus, coping with task failures or persistent data transmission is an open problem.

ACKNOWLEDGMENT

This work is partially supported by National key research and development program under Grant No 2018YFE0207600, National Natural Science Foundation of China under Grant No. U19B2024 and No. 62002378, and Tianjin Science and Technology Foundation under Grant No. 18ZXJMTG00290.

REFERENCES

[1] Y. Huang, Y. Shi, Z. Zhong, Y. Feng, J. Cheng, J. Li, H. Fan, C. Li, T. Guan, and J. Zhou, "Yugong: Geo-distributed data and job placement at scale," in *Morgan Kaufmann/ACM VLDB, Los Angeles, USA, August 26-30*, 2019.

[2] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching lan speeds," in *USENIX NSDI, Boston, USA, April 9-11*, 2017.

[3] C. Hung, G. Ananthanarayanan, L. Golubchik, M. Yu, and M. Zhang, "Wide-area analytics with multiple resources," in *ACM EuroSys, Porto, Portugal, April 23-26*, 2018.

[4] "Private conversation with datacenter operators of one of the largest public cloud providers," anonymized, Tech. Rep., 2016.

[5] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," in *IEEE INFOCOM, Paris, France, April 29 - May 2*, 2019.

[6] V. Farhadi, F. Mehmeti, T. He, T. F. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 779–792, 2021.

[7] H. Wang, D. Niu, and B. Li, "Turbo: Dynamic and decentralized global analytics via machine learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1372–1386, 2020.

[8] Q. Pu, G. Ananthanarayanan, P. Bodík, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *ACM SIGCOMM, London, United Kingdom, August 17-21*, 2015.

[9] J. Jiang, R. Das, G. Ananthanarayanan, P. A. Chou, V. N. Padmanabhan, V. Sekar, E. Dominique, M. Goliszewski, D. Kukoleca, R. Vafin, and H. Zhang, "Via: Improving internet telephony call quality using predictive relay selection," in *ACM SIGCOMM, Florianópolis, Brazil, August 22-26*, 2016.

[10] Z. Hu, D. Li, and D. Guo, "Balance resource allocation for spark jobs based on prediction of the optimal resource," *Tsinghua Science and Technology*, vol. 25, no. 4, pp. 487–497, 2020.

[11] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *ACM SIGCOMM, Budapest, Hungary, August 20-25*, 2018.

[12] S. Liu, L. Chen, B. Li, and A. Carnegie, "A hierarchical synchronous parallel model for wide-area graph analytics," in *IEEE INFOCOM, Honolulu, USA, April 15-19*, 2018.

[13] Z. Hu, B. Li, and J. Luo, "Time-and cost-efficient task scheduling across geo-distributed data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 705–718, 2018.

[14] W. Xiao, W. Bao, X. Zhu, and L. Liu, "Cost-aware big data processing across geo-distributed datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, pp. 3114–3127, 2017.

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

[16] H. Hu, Y. Wen, T. Chua, and X. Li, "Cost-optimized microblog distribution over geo-distributed data centers: Insights from cross-media analysis," *ACM Transactions on Intelligent Systems and Technology*, vol. 8, no. 3, pp. 1–18, 2017.

[17] L. Chen, S. Liu, B. Li, and B. Li, "Scheduling jobs across geo-distributed datacenters with max-min fairness," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 488–500, 2019.

[18] W. Li, X. Yuan, K. Li, H. Qi, and X. Zhou, "Leveraging endpoint flexibility when scheduling coflows across geo-distributed datacenters," in *IEEE INFOCOM, Honolulu, USA, April 15-19*, 2018.

[19] S. Liu, W. Hao, and B. Li, "Optimizing shuffle in wide-area data analytics," in *IEEE ICDCS, Atlanta, USA, June 5 – 8*, 2017.

[20] L. Luo, D. Guo, W. Li, T. Zhang, J. Xie, and X. Zhou, "Compound graph based hybrid data center topologies," *Frontiers of Computer Science*, vol. 9, no. 6, pp. 860–874, 2015.

[21] C. Hung, G. Ananthanarayanan, P. Bodík, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *IEEE/ACM SEC, Bellevue, USA, October 25-27*, 2018.

[22] W. Li, D. Guo, A. X. Liu, K. Li, H. Qi, S. Guo, A. Munir, and X. Tao, "Coman: Managing bandwidth across computing frameworks in multiplexed datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 5, pp. 1013–1029, 2018.

[23] D. Guo, J. Xie, X. Shi, H. Cai, C. Qian, and H. Chen, "HDS: A fast hybrid data location service for hierarchical mobile edge computing," *ACM/IEEE Transactions on Networking*, vol. 29, no. 3, pp. 1308–1320, 2021.

[24] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *ACM SIGCOMM, Beijing, China, August 19-23*, 2019.

[25] http://prof.ict.ac.cn/.

[26] H. Zheng and J. Wu, "Joint scheduling of overlapping mapreduce phases: Pair jobs for optimization," *IEEE Transactions on Services Computing*, pp. 1–11, 2018.

[27] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "Clarinet: Wan-aware optimization for analytics queries," in *USENIX OSDI, Savannah, USA, Novermber 2-4*, 2016.

[28] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, K. Karanasos, J. Padhye, and G. Varghese, "Wanalytics: Geo-distributed analytics for a data intensive world," in *ACM SIGMOD, Victoria, Australia, May 31- June 4*, 2015.

[29] K. Kloudas, R. Rodrigues, N. M. Preguiça, and M. Mamede, "Pixida: optimizing data parallel jobs in wide-area data analytics," in *Morgan Kaufmann/ACM VLDB, Kohala Coast, Hawai'i, August 31 - September 4*, 2015.

[30] 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 *USENIX NSDI, Oakland, USA, May 4-6*, 2015.

[31] H. Wang and B. Li, "Mitigating bottlenecks in wide area data analytics via machine learning," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 155–166, 2020.

[32] A. P. Iyer, A. Panda, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica, "Monarch: Gaining command on geo-distributed graph analytics," in *USENIX HotCloud, Boston, USA, July 9*, 2018.

[33] A. C. Zhou, S. Ibrahim, and B. He, "On achieving efficient data transfer for graph processing in geo-distributed datacenters," in *IEEE ICDCS, Atlanta, USA, June 5-8*, 2017.

[34] 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 Transactions on Parallel and Distributed Systems*, vol. 31, no. 2, pp. 279–293, 2020.

[35] T. Magrino, J. Liu, N. Foster, J. Gehrke, and A. C. Myers, "Efficient, consistent distributed computation with predictive treaties," in *ACM EuroSys, Dresden, Germany, March 25-28*, 2019.

[36] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *USENIX NSDI, San Francisco, USA, December 11*, 2008.

[37] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *ACM STOC, Washington, USA, April 30 - May 2*, 1984.

[38] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *USENIX NSDI, Santa Clara, USA, March 16-18*, 2016.

[39] A. Gounaris, G. Kougka, R. Tous, C. T. Montes, and J. Torres, "Dynamic configuration of partitioning in spark applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 1891–1904, 2017.

[40] https://commondatastorage.googleapis.com/clusterdata-2011-2/, 2011.

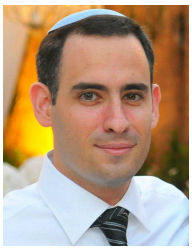
[41] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *ACM SOCC, San Jose, USA, October 14-17*, 2012.

[42] https://github.com/alibaba/clusterdata, 2018.

Yiting Chen received MS degree in School of information and communication from Guilin university of electronic technology of China, Guilin, China, in 2017. She is currently working toward Ph.D in 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 received his B.S, M.S. and Ph.D degree at the school of systems engineering from National University of Defence Technology, Changsha, China, in 2013, 2015 and 2019, respectively. He is currently a lecturer in the school of systems engineering, National University of Defense Technology, Changsha, China. His research interests include probabilisitic data structures and data analysis.

Deke Guo 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, and is also with the College of Intelligence and Computing, Tianjin University. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks. He is a senior member of the IEEE and a member of the ACM.



Ori Rottenstreich is an assistant professor at the department of Computer Science and the department of Electrical Engineering of the Technion, Haifa, Israel. Previously, he was a Postdoctoral Research Fellow at Princeton University. Ori received his B.Sc. degree in Computer Engineering and Ph.D. degree in Electrical Engineering from Technion.

Jie Wu is the associate vice provost for international affairs with Temple University. He also serves as the chair and Laura H. Carnell professor with the Department of Computer and Information Sciences. Prior to joining Tempe University, he was a program director with the US National Science Foundation and was a distinguished professor with Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including the IEEE Transactions on Service Computing and the Journal of Parallel and Distributed Computing. He was general co-chair/chair of the IEEE Mobile Adhoc and Sensor Systems 2006, the IEEE International Parallel & Distributed Processing Symposium 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society distinguished visitor, ACM distinguished speaker, and chair of the IEEE Technical Committee on Distributed Processing (TCDP). He is a fellow of the IEEE.