

Task Scheduling and Transmission for Distributed Job Execution

Shangyi Shao
Nanyang Technological University
Singapore
shangyi001@e.ntu.edu.sg

Xueyan Tang
Nanyang Technological University
Singapore
asxytang@ntu.edu.sg

Abstract—Deploying data centers across geographically distributed locations has become common practice for exploiting data locality to meet strict latency and reliability requirements of various services. This paper studies a job scheduling problem for distributed job execution, in which the data partitions required by the tasks of jobs can be transferred between different sites leveraging network resources in the system. Our goal is to minimize the average job response time by jointly optimizing job scheduling and task transmission. We present a Workload-Aware Selection with Transmission algorithm, which coordinates job scheduling across different sites with task transmission and aims to provide a near-optimal solution. Our approach starts with a task transmission strategy tailored to optimize the response time of a single job. We then develop an integrated solution that jointly schedules multiple jobs along with their associated task transmissions. To balance performance and computational efficiency, we also present a fast heuristic approximation algorithm. Experiments with real job traces from Alibaba production clusters demonstrate the effectiveness of our proposed algorithms under various system utilization and bandwidth settings.

I. INTRODUCTION

Establishing multiple data centers distributed across geographically separated areas has become a prevailing strategy for data processing, by which jobs and tasks can be placed and processed at different data centers according to where their required input data are available. Through this approach, partial computation results are derived and exchanged among data centers in order to obtain the final result, which substantially reduces the volume of data that needs to be transferred and effectively improves network resource usage and service response speed [1].

The deployment of geo-distributed data centers has been applied to a wide range of applications, such as databases [2], [3], big data processing [4], [5], and machine learning [6], [7]. Meanwhile, this kind of geo-distributed computing systems introduces many new challenges, such as how to improve system efficiency and decrease the volume of data transmission. In a single data center scenario, the strategy for achieving the highest job processing efficiency (i.e., minimize the average job response time – the response time of a job is defined as the duration between its arrival time and its completion time) is to sort and execute jobs based on their computation loads (i.e., the total processing time of all tasks of each job) from low to high, known as the Shortest Remaining Processing Time (SRPT) strategy. Similar ideas can also be applied to the distributed job execution scenario where jobs are comprised of tasks to be processed at multiple data centers, but this can be far from

optimal. In geo-distributed job execution, the completion time of each job is determined by the completion time of its last task across all sites. Therefore, the execution order of jobs within individual sites leaves a huge space for optimization, especially when the task distributions of jobs among different sites are imbalanced. How to achieve optimal job scheduling in distributed job execution is an NP-hard problem.

What makes the problem even more complicated is how to utilize network resources in the system to further enhance job processing efficiency. Since the completion time of the last task decides the response time of a job, transferring the input data of tasks to other sites can potentially bring forward the execution of tasks and thereby improve the job's response time. However, there exists complex interdependency between the execution order of jobs and the transmission of tasks. Even the transmission of a single task may change the optimal execution order of jobs, while different job execution orders may, in turn, require different task transmissions to minimize the average job response time. As a result, it is highly challenging to find the optimal solution.

In this paper, we focus on developing job scheduling algorithms which utilize network resources to transfer tasks and improve job processing efficiency in geo-distributed systems. We present a novel job scheduling strategy, called Workload-Aware Selection with Transmission (WAST), designed to optimize average job response time. WAST computes the global execution order of jobs and a set of task transmissions between different sites in an integrated manner. More specifically, WAST searches for the task transmission strategy to achieve the best possible response time of each job by formulating an optimization problem and transforming it into multiple minimum-cost flow problems that can be efficiently solved by popular solvers (e.g., OR-Tools [8], Cplex [9], and GUROBI [10]), and iteratively schedules the job with the minimum response time until all jobs are scheduled. We also present a heuristic approximation algorithm that reduces the computational complexity without compromising the quality of scheduling much. We validate the effectiveness of the WAST algorithm and its approximation using publicly available job traces from Alibaba. Experimental results show that both algorithms can efficiently compute near-optimal job execution order and task transmissions, thereby significantly reducing job response times, especially in scenarios where the initial computation load distributions of jobs are highly skewed



Fig. 1: Motivating Example

across sites.

The rest of this paper is organized as follows. Section II presents a motivating example for our work. Section III describes the system model and problem definition. Section IV elaborates the design of our proposed algorithms. The experimental settings and results are discussed in Section V. Section VII summarizes the related work. Finally, Section VIII concludes the paper.

II. A MOTIVATING EXAMPLE

Fig. 1 provides a motivating example of the problem with four jobs J_1 , J_2 , J_3 and J_4 in three geo-distributed data centers (sites) S_1 , S_2 and S_3 . All jobs are submitted at time 0. For simplicity, we assume that each task of any job requires one computing slot and takes one time unit to process, and each site has one computing slot only. Fig. 1a shows the initial task distribution of each job among the three sites. The response time of a job is decided by the completion time of its last task across all sites. Obviously, different execution orders of the jobs can change the completion times of their tasks and result in different job response times. If the jobs are simply scheduled in the order of their indices (which, for example, represents their submission order), jobs J_1 , J_2 , J_3 and J_4 will complete at time 4, 8, 10 and 13, respectively, so the total job response time is $4 + 8 + 10 + 13 = 35$ time units. The optimal job execution order without task transmission is $\langle J_4, J_3, J_1, J_2 \rangle$ as shown in Fig. 1b, and the resultant total job response time is $3 + 6 + 9 + 13 = 31$ time units, which is 4 time units better than the first schedule.

Suppose that with utilization of network resources, the input data of one task is allowed to be transferred between sites and it takes one time unit to arrive at the destination. Under the job execution order of Fig. 1b, the optimal transmission can decrease the total job response time to 29 time units. For example, we can transfer one task of J_1 from site S_1 to S_2 , as shown in Fig. 1c. This task will arrive at site S_2 at time 1 when the slot of S_2 is processing a task of J_4 (and J_4 is scheduled earlier than J_1 in the execution order), so the task transferred can be treated as a local task of J_1 at site S_2 initially. Consequently, J_4 , J_3 , J_1 and J_2 will complete at time 3, 6, 8 and 12, respectively, and the total job response time is $3 + 6 + 8 + 12 = 29$ time units. In our example, the optimal solution is to transfer one task of J_4 from site S_1 to S_3 while

modifying the job execution order to be $\langle J_4, J_1, J_3, J_2 \rangle$, as shown in Fig. 1d. This results in the total job response time of $2 + 6 + 8 + 12 = 28$ time units, which is even better than the schedule in Fig. 1c.

The above example illustrates some interesting phenomena. First, job execution order and task transmission have mutual influence. The optimal execution order of jobs can change after task transmissions are performed: both execution orders in Figs. 1b and 1d achieve the optimal total job response times given the respective task transmissions, but J_1 and J_3 are swapped in the execution order after a task is transferred in Fig. 1d. Vice versa, different job execution orders have different optimal task transmissions: the optimal transmission changes after we modify the job execution order from $\langle J_4, J_3, J_1, J_2 \rangle$ (Fig. 1c) to $\langle J_4, J_1, J_3, J_2 \rangle$ (Fig. 1d). Second, the transmission in the optimal solution is from site S_1 to S_3 , while the site with the lowest load is S_2 . This indicates that the transmissions that minimize the total job response time can differ from the one that focuses on load balancing, so traditional load balancing techniques may not be preferable in this efficiency-targeted scenario.

III. SYSTEM MODEL

Consider a system consisting of m separate geo-distributed sites: $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$. Each site S_j has compute resources with capacity u_j , which can be interpreted as the volume of computing slots available. A set of n jobs $J_1, J_2, \dots, J_n$ are submitted to run in the system. Each job J_i consists of tasks that require distinct data partitions and can run in parallel. For simplicity, we assume that each task takes one computing slot to process. In big data processing frameworks, tasks of the same job typically process data partitions of the same size [4], [5], so they would take similar times to process. We denote the processing time of a task in each job J_i as l_i , which can be estimated by the historical execution records of the job or sampling-based learning, which proactively samples a small fraction of the tasks in the job to make an estimate [11]. A job is completed when all of its tasks are completed.

The data partitions required by the tasks of each job are originally distributed among the sites. Due to data locality consideration, each task is initially restricted to be executed at a certain site where its input data is available. Therefore, the computational demand of a job J_i on site S_j can be modeled

TABLE I: Summary of Notations

Notation	Definition
$\mathcal{S} = \{S_1, S_2, \dots, S_m\}$	Set of sites
$U = \{u_1, u_2, \dots, u_m\}$	Compute resource capacities of individual sites
$\{J_1, J_2, \dots, J_n\}$	Set of jobs
$D_{n \times m}$	Compute resource demands
$B_{m \times m}$	Network resource availability
$\{l_1, l_2, \dots, l_n\}$	Task processing times of individual jobs
$\{r_1, r_2, \dots, r_n\}$	Data partition sizes of individual jobs

as the number of its tasks that can run only on S_j initially. We represent the computational demands of all jobs across all sites as a matrix $D_{n \times m}$, where each element d_{ij} denotes job J_i 's computational demand on site S_j .¹

Suppose the sites in the system are connected by networks, which can be used to transfer data and hence tasks from one site to another. The network resources in the system is denoted by a matrix $B_{m \times m}$, where each element b_{ij} represents the bandwidth available on the link from site S_i to site S_j . b_{ij} is set to 0 if there is no direct link between S_i and S_j . Let r_k denote the size of a data partition of job J_k , then a task of J_k takes r_k/b_{ij} time to transfer from site S_i to site S_j (if $b_{ij} = 0$, this time is defined to be infinite). For simplicity, we assume that the transmissions on each link are carried out sequentially. That is, if there is an ongoing transmission on the link, a new transmission has to wait and its arrival time at the destination site will be delayed.

We study the problem of deriving an execution order of jobs and a set of task transmissions between sites, so that the total job response (completion) time of all jobs is minimized. The key notations used in this paper are summarized in Table I. For simplicity, we use $[x]$ to represent the set $\{1, 2, \dots, x\}$ for a positive integer x . Note that although the above model is formulated in a static setting, it can be easily extended to address online scenarios by dynamically capturing snapshots of job demand distributions and system resources.

IV. WORKLOAD-AWARE SELECTION WITH TRANSMISSION

A. Optimizing the Completion Time of a Single Job

We start by considering the subproblem of how to utilize network resources in the system to minimize the completion time of a single job. This subproblem can be formulated as follows: Given some workloads and task transmissions already scheduled in the system, and a job J_k with task processing time l_k , data partition size r_k and initial demand distribution $d_{k1}, d_{k2}, \dots, d_{km}$ across the m sites, we would like to determine a set of task transmissions for J_k so that its job completion time can be minimized.

We use a sequence of arrays $W_1, W_2, \dots, W_m$ to represent the workload already scheduled for each computing slot at each site, where the workload of a slot refers to the total processing time of all tasks assigned to it (i.e., the time when this

slot will become available). Each W_i is an array whose length is the number of computing slots at site S_i , and each element w_{ij} in W_i represents the scheduled workload of the j -th slot at S_i . In addition, we use a matrix $T_{m \times m}$ to represent the usage of network resources by the task transmissions already scheduled, where t_{ij} records the time when the link from site S_i to site S_j will complete the scheduled transmissions and become available.

To avoid the complex interaction between task transmissions and executions and simplify the problem model, we assume that every transferred task will arrive at its destination site before any computing slot at that site becomes available. This ensures that the transferred tasks can be processed continuously at their destination sites together with the tasks originally available there, which simplifies the modeling of job completion time. Based on this, we model the completion time of job J_k as follows. Suppose a matrix $X_{m \times m}$ represents the task transmissions of job J_k , where each element x_{ij} refers to how many tasks of J_k are to be transferred from site S_i to site S_j . Naturally, each x_{ij} is greater than or equal to 0. In particular, for each site S_i , we define $x_{ii} = 0$. We use an array Y of length m to represent the computational demands at the m sites after all the transmissions in X are performed. Each element y_i refers to the number of tasks to be executed at site S_i after the transmissions, and is given by:

$$\forall i \in [m], \quad y_i = d_{ki} + \sum_{j=1}^m x_{ji} - \sum_{j=1}^m x_{ij}. \quad (1)$$

Recall that we assume each task transmission to arrive at its destination site before any computing slot at that site becomes available. Thus, all the y_i tasks of job J_k to be executed at site S_i can be scheduled by iteratively assigning each task to the slot of the lowest workload in S_i . Let $w_{ij}^{(t)}$ denote the workload of the j -th slot in site S_i after t tasks of J_k are assigned to the slots at S_i . $w_{ij}^{(t)}$ can be computed iteratively as follows:

$$\forall j \in [u_i], \quad w_{ij}^{(t)} = \begin{cases} w_{ij}, & t = 0 \\ w_{ij}^{(t-1)} + l_k, & j = \operatorname{argmin}_{p \in [u_i]} w_{ip}^{(t-1)}, \\ w_{ij}^{(t-1)}, & j \neq \operatorname{argmin}_{p \in [u_i]} w_{ip}^{(t-1)}, \end{cases}$$

where in each iteration, a task is assigned to the slot of the lowest workload ($\operatorname{argmin}_{p \in [u_i]} w_{ip}^{(t-1)}$) and its workload then increases by l_k , while other slots are kept unchanged. Ties are broken arbitrarily if multiple slots have the same lowest workload. Based on this, the time when all y_i tasks complete execution at site S_i can be estimated as a function $f(W_i, y_i)$:

$$f(W_i, y_i) = \begin{cases} 0, & y_i = 0, \\ \min_{j \in [u_i]} (w_{ij}^{(y_i-1)} + l_k), & y_i > 0, \end{cases}$$

which gives the completion time of the last task at S_i . Note that function f 's inputs and outputs cannot be captured by any closed-form expression. The objective of optimizing the job completion time of J_k (i.e., $\max_{i \in [m]} f(W_i, y_i)$ — the completion time of its last task across all sites) can be achieved

¹Throughout this paper, we shall use capital letters to represent matrices as well as arrays, and use the corresponding small letters to stand for the elements therein.

by searching for the best task transmissions X between sites, which decide the computational demands Y at different sites and hence job J_k 's completion time.

It is essential to determine the bounds of X before starting the search. An excessively tight bound can limit the potential benefit of task transmissions in reducing the job completion time, whereas an excessively loose bound may give rise to task transmissions that fail to reach the destination sites in time, thereby introducing additional delays in the job completion time. Suppose x_{ij} tasks are to be transferred from site S_i to site S_j . Then, the completion time of the last task will be at least $t_{ij} + x_{ij} \cdot r_k / b_{ij} + l_k$ (when it is processed immediately after arrival), and we require this last task to finish before the job completion time:

$$\forall i \in [m], j \in [m], t_{ij} + \frac{x_{ij} \cdot r_k}{b_{ij}} + l_k \leq \max_{i \in [m]} f(W_i, y_i). \quad (2)$$

In addition, the number of tasks transferred out of a site cannot exceed the number of tasks it holds initially, which imposes another upper bound on $\sum_{j=1}^m x_{ij}$ at each site S_i :

$$\forall i \in [m], 0 \leq \sum_{j=1}^m x_{ij} \leq d_{ki}. \quad (3)$$

Constraints (2) and (3) collectively define the upper bounds of the elements in X . Therefore, we can define a program to search for task transmissions X under constraints (1), (2) and (3) to optimize the job completion time $\max_{i \in [m]} f(W_i, y_i)$. If multiple transmission matrices X 's produce the same minimum job completion time, we choose the X with the least total number of task transmissions. The rationale is to minimize the network resource usage and make more network resources available to other jobs.

B. Transformation into ILP

The program in Section IV-A cannot be solved directly by popular optimization solvers owing to the discrete function f , which cannot be captured by any closed-form expression. In order to make the program solvable, we transform it into solving multiple Integer Linear Programs (ILP) that are supported by popular solvers. To facilitate presentation, we refer to the completion time of the last task at a site S_i as "the completion time at site S_i ". We compute and record the completion times $f(W_i, y_i)$ at each site S_i using the method introduced in Section IV-A, for increasing number of tasks y_i scheduled at site S_i starting from $y_i = 0$. The calculation ends at the lowest y_i value where the resulting $f(W_i, y_i)$ at site S_i exceeds the job completion time when all tasks are executed locally without transmission, i.e., $\max_{i \in [m]} f(W_i, d_{ki})$.

Note that these $f(W_i, y_i)$ values include all possible completion times at each site that J_k can produce, and they are non-decreasing with respect to y_i since adding a new task can never reduce the completion time at a site. With these $f(W_i, y_i)$ values, given any target job completion time z , the maximum number of tasks c_i that each site S_i can process (to keep its completion time within this

Algorithm 1 Optimizing the Completion Time of a Single Job

- 1: **Input:** A set of sites $\mathcal{S}$ with compute capacities U and network resources B ; a job J_k to be scheduled with task processing time l_k , data partition size r_k and computational demand distribution $d_{k1}, d_{k2}, \dots, d_{km}$; current workload $W_1, W_2, \dots, W_m$ and network usage T ;
- 2: **Output:** J_k 's optimal job completion time and task transmissions;
- 3: **Initialization:** Compute and record $f(W_i, y_i)$'s for each site S_i with increasing y_i until $f(W_i, y_i) \geq \max_{i \in [m]} f(W_i, d_{ki})$, using the method in Section IV-A; merge and sort all $f(W_i, y_i)$ values to form an array ψ ;
- 4: $low = 1$; $high = \psi.size$; $opt = 0$;
- 5: **while** $low < high$ **do**
- 6: $middle = \lfloor (low + high) / 2 \rfloor$;
- 7: $c_i = \max\{p \mid f(W_i, p) \leq \psi[middle]\}$, for each $i \in [m]$;
- 8: **Minimize** $\sum_{i=1}^m \sum_{j=1}^m x_{ij}$, **subject to:**
- 9: $\forall i \in [m], j \in [m], t_{ij} + \frac{x_{ij} \cdot r_k}{b_{ij}} + l_k \leq \psi[middle]$;
- 10: $\forall i \in [m], 0 \leq \sum_{j=1}^m x_{ij} \leq d_{ki}$;
- 11: $\forall i \in [m], y_i = d_{ki} + \sum_{j=1}^m x_{ji} - \sum_{j=1}^m x_{ij}$;
- 12: $\forall i \in [m], y_i \leq c_i$;
- 13: all x_{ij} 's and y_i 's are non-negative integers;
- 14: **if** the above ILP is feasible and admits a solution **then**
- 15: $opt = \psi[middle]$;
- 16: $X \leftarrow x_{ij}$'s in the solution of the above ILP;
- 17: $high = middle - 1$;
- 18: **else**
- 19: $low = middle + 1$;
- 20: **return** opt, X ;

target) can be derived as $c_i = \max\{p \mid f(W_i, p) \leq z\}$. Thus, the target z can be converted to an array of upper bounds $c_1, c_2, \dots, c_m$ for the elements $y_1, y_2, \dots, y_m$ in Y . Hence, the problem of determining "whether the job J_k can achieve a target job completion time z using task transmissions" can be transformed to determining whether task transmissions can be performed to bound the elements in $y_1, y_2, \dots, y_m$ by $c_1, c_2, \dots, c_m$. The latter can be formulated as an ILP, because its constraints are linear and the elements in X and Y are integers. Furthermore, the minimum job completion time that J_k can achieve must be among all the $f(W_i, y_i)$ values. Therefore, we can sort all values $f(W_1, 0), f(W_1, 1), \dots, f(W_1, c_1), f(W_2, 0), f(W_2, 1), \dots, f(W_2, c_2), \dots, f(W_m, 0), f(W_m, 1), \dots, f(W_m, c_m)$ to form an array ψ . Then, the minimum job completion time of J_k can be found by a binary search in ψ and solving multiple ILPs.

Algorithm 1 presents the details of optimizing the job completion time of job J_k . $\psi.size$ represents the number of elements in the array ψ . The algorithm first computes and records the possible completion times $f(W_i, y_i)$ at each site S_i using the method introduced in Section IV-A. These values are then merged and sorted to form a new array ψ which records the job completion times that J_k can possibly achieve. Next, a

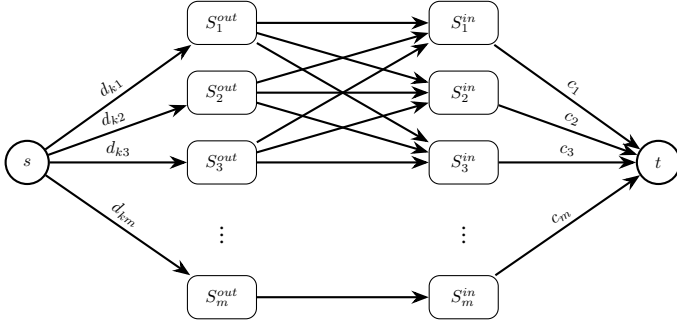


Fig. 2: The flow network

binary search is applied to the array ψ to find the optimal job completion time in ψ as well as the transmissions required. The parameters *low* and *high* in the algorithm refer to the boundary indices of the binary search. In each iteration, the *middle* index and bounds $c_1, c_2, \dots, c_m$ are updated, and the feasibility of an ILP is checked to decide whether the job completion time $\psi[middle]$ is achievable.

C. Transformation into Minimum-cost Flow Problem

Algorithm 1 solves the problem, but its computational complexity could raise concerns. ILP is NP-hard in general. If the solution space becomes huge (e.g., the job has thousands of tasks unevenly spread across tens of geo-distributed sites), solving the ILP can incur intolerable computational overhead. To improve the algorithm's efficiency, we further transform the ILP problem in lines 8-13 of Algorithm 1 to a minimum-cost flow problem in a bipartite flow network shown in Fig. 2. The flow network has a source node s , a sink node t , a set of m nodes on the left side and a set of m nodes on the right side. To facilitate exposition, we use S_i^{out} and S_i^{in} to represent the i -th node on the left and right sides, respectively. A set of m edges with cost 0 connects the source node s to the m nodes on the left, and the capacity of each edge (s, S_i^{out}) is set to d_{ki} . In addition, a set of $m \times m$ edges connects the m nodes on the left to the m nodes on the right. The capacity of each edge (S_i^{out}, S_j^{in}) where $i \neq j$ is set to the upper bound of x_{ij} given in line 9 of Algorithm 1 (where $\psi[middle]$ is the target job completion time) and the cost of the edge is set to 1. The capacity of each edge connecting each pair of nodes with the same index (S_i^{out}, S_i^{in}) is set to d_{ki} (the initial computational demand at site S_i) and the cost of the edge is set to 0. Finally, a set of m edges with cost 0 connect the m nodes on the right to the sink node t , and the capacity of each edge (S_i^{in}, t) is set to c_i (the upper bound of y_i for the target job completion time). The problem is to decide whether the bipartite flow network can transfer all $\sum_{i=1}^m d_{ki}$ units of flows from s to t under the given edge capacity constraints while minimizing the total cost.

Each edge (s, S_i^{out}) represents the initial number of tasks d_{ki} at site S_i . Each edge (S_i^{out}, S_j^{in}) where $i \neq j$ denotes the task transmissions from site S_i to site S_j . Due to the flow conservation constraint, the sum of flows going out of each node S_i^{out} is d_{ki} , ensuring that the number of task

transmissions out of site S_i does not exceed d_{ki} . Each edge (S_i^{out}, S_j^{in}) denotes the number of local tasks that are not transferred out of site S_i , so it refers to no transmission and the cost is set to 0. Each edge (S_i^{in}, t) represents the final number of tasks to be executed at site S_i after transmissions, which is required to fall below the upper bound c_i . It is easy to verify that this minimum-cost flow problem is equivalent to the ILP problem in Section IV-B. Since all the edge capacities in the flow network are integers, based on the Integral Flow Theorem [12], the solution (if it exists) is made up of integral units of flows on all edges. Existing algorithms (e.g., Successive Shortest Path Algorithm, Cost Scaling Algorithm [13]) for solving the minimum-cost flow problem in bipartite graphs have polynomial time complexities. They can solve the problem more efficiently than ILP.

D. Optimizing the Completion Times of All Jobs

We have solved the problem of minimizing the completion time of a single job given the workloads and task transmissions already scheduled. We now consider how to deal with the problem when multiple jobs are submitted to run in the system. Since how a single job will utilize the network resources has been solved, the key problem that remains is how to decide an efficient job execution order. Inspired by the work from [14], we develop a Workload-Aware Selection with Transmission (WAST) algorithm, which aims to minimize the total job completion time by scheduling job execution and task transmission progressively across the system. In each iteration, WAST traverses all unscheduled jobs, searches for task transmissions for each job to minimize its completion time (accounting for the workloads and task transmissions already scheduled), and selects the job with the earliest possible completion time to add to the execution order.

Algorithm 2 shows the details of WAST. Initially, there is no scheduled workload or task transmission, and the set of unscheduled jobs $\mathcal{J}$ contains all jobs (line 3). In each iteration, the algorithm computes the optimal completion time of each job in $\mathcal{J}$ using Algorithm 1 with ILP transformed into solving the minimum-cost flow problem (lines 5-6). Then, the job J_{min} with the earliest completion time is selected and added to the job execution order, and its task transmissions X^{min} are saved (lines 7-8). Meanwhile, the scheduled workload W and network usage T are updated based on those arising from J_{min} , and J_{min} is removed from $\mathcal{J}$ (lines 9-14). The algorithm terminates when all jobs are scheduled and returns the job execution order and task transmissions of each job.

Additional optimizations can be applied to enhance the algorithm's computational efficiency. We can reduce the number of binary search operations for each job by decreasing the number of elements to consider in the array ψ (the job completion times that the job can possibly achieve). First, duplicate elements can be removed from ψ . Next, suppose that infinite bandwidth is available on the links between all site pairs so that any task transmission is instant. In this case, the best strategy that minimizes job completion time is to treat all m sites as one aggregate site and iteratively assign each task to

Algorithm 2 Workload-Aware Selection with Transmission

```
1: Input: A set of sites  $\mathcal{S}$  with compute capacities  $U$  and
   network resources  $B$ ; a set of jobs  $J_1, J_2, \dots, J_n$  with
   task processing times  $l_1, l_2, \dots, l_n$  and data partition sizes
    $r_1, r_2, \dots, r_n$ ; a computational demand matrix  $D$ ;
2: Output: Job execution order and task transmissions;
3: Initialization:  $\mathcal{J} = \{J_1, J_2, \dots, J_n\}$ ; scheduled workload
    $W_i = \mathbf{0}$ , for each  $i \in [m]$ ; network usage  $T = \mathbf{0}$ ;
4: while  $\mathcal{J} \neq \emptyset$  do
5:   for each job  $J_k \in \mathcal{J}$  do
6:     Call Algorithm 1 on  $J_k$  given  $W$  and  $T$  to obtain
        $J_k$ 's optimal completion time and the correspond-
       ing task transmissions  $X^k$ ;
7:   Find the job  $J_{min} \in \mathcal{J}$  of the earliest completion time;
8:   Append  $J_{min}$  to the execution order and save its task
       transmissions  $X^{min}$ ;
9:    $t_{ij} = t_{ij} + x_{ij}^{min} \cdot r_{min}/b_{ij}$ , for each  $i, j \in [m]$ ;
10:   $y_i = d_{ki} + \sum_{j=1}^n x_{ji}^{min} - \sum_{j=1}^m x_{ij}^{min}$ , for each  $i \in [m]$ ;
11:  for each  $i \in [m]$  do
12:    for each  $count \in [y_i]$  do
13:       $j = \operatorname{argmin}_{p \in [u_i]} w_{ip}$ ;
14:       $w_{ij} = w_{ij} + l_{min}$ ;
15:   $\mathcal{J} = \mathcal{J} \setminus J_{min}$ ;
16: return Job execution order and each job's task transmis-
   sions;
```

the slot with the lowest workload globally. The job completion time derived in this way is guaranteed to be no worse than the optimal solution in the case of finite bandwidth, so it is a lower bound on the optimal job completion time. As a result, all elements in ψ below this lower bound can be eliminated from consideration. Furthermore, in each iteration, suppose that job J_{min} has the minimum completion time t^* among all jobs examined before examining an unscheduled job J_i . If the above lower bound of J_i 's completion time is higher than or equal to t^* , there is no need to examine J_i in this iteration because it cannot achieve an earlier job completion time than t^* . Otherwise, if the lower bound of J_i 's completion time is lower than t^* , t^* can be set as an upper bound when searching in J_i 's array ψ of possible job completion times, since we are only concerned about whether J_i can achieve an earlier job completion time than t^* . In summary, when examining an unscheduled job J_i , we can remove all elements in ψ that fall below the lower bound of its completion time or exceed t^* .

E. Recursive Approximation

Algorithm 2 attempts to minimize the average job completion time in a step-by-step manner by solving multiple minimum-cost flow problems. We also present an alternative recursive approximation method called WAST_a which provides better computational efficiency. The outer loop of WAST_a is the same as WAST, and we only present the inner loop in Algorithm 3 which optimizes the completion time of a single job J_k . For simplicity, WAST_a assumes that, a site can either receive J_k 's tasks from other sites

Algorithm 3 Recursive Approximation of Optimizing the Completion Time of a Single Job

```
1: Input: A set of sites  $\mathcal{S}$  with compute capacities  $U$  and
   network resources  $B$ ; a job  $J_k$  to be scheduled with task
   processing time  $l_k$ , data partition size  $r_k$  and compu-
   tational demand distribution  $d_{k1}, d_{k2}, \dots, d_{km}$ ; current
   workload  $W_1, W_2, \dots, W_m$  and network usage  $T$ ;
2: Output:  $J_k$ 's job completion time and transmissions;
3: Initialization: Copy current workload:  $W'_i = W_i$ , for each
    $i \in [m]$ ;  $X = \mathbf{0}$ ;  $q = 0$ ;
4: for each  $i \in [m]$  do
5:   for each  $count \in [d_{ki}]$  do
6:      $j = \operatorname{argmin}_{p \in [u_i]} w_{ip}$ ;
7:      $w_{ij} = w_{ij} + l_k$ ;
8:      $q = \max\{q, w_{ij}\}$ ;
9: while true do
10:  Find a task whose completion time =  $q$ ;
11:  Suppose the task is currently assigned to the  $a$ -th slot
     of site  $S_g$ ;
12:  if site  $S_g$  has ever received  $J_k$ 's tasks then
13:    return  $q, X$ ;
14:   $q^* = q$ ,  $h = -1$ ;
15:  for each  $i \in [m]$  and  $i \neq g$  do
16:    if  $t_{gi} + r_k/b_{gi} \leq \min_{p \in [u_i]} w_{ip}$  and
        $\min_{p \in [u_i]} w_{ip} + l_k < q^*$  then
17:       $q^* = \min_{p \in [u_i]} w_{ip} + l_k$ ;
18:       $h = i$ ;
19:  if  $h == -1$  then
20:    return  $q, X$ ;
21:  else
22:     $x_{gh} = x_{gh} + 1$ ;
23:     $t_{gh} = t_{gh} + r_k/b_{gh}$ ;
24:     $w_{ga} = w_{ga} - l_k$ ;
25:     $v = \operatorname{argmin}_{p \in [u_h]} w_{hp}$ ;
26:     $w_{hv} = w_{hv} + l_k$ ;
27:     $q = \max_{i \in [m]} f(W'_i, d_{ki} + \sum_{j=1}^m x_{ji} - \sum_{j=1}^m x_{ij})$ ;
```

or send J_k 's tasks to other sites, but not both. Under this assumption, each task will be transferred at most once, because the destination site of any transmission will not send any task out. WAST_a first computes the completion time of J_k when all its tasks are executed locally without transmission as $q = \max_{i \in [m]} f(W_i, d_{ki})$ (line 8). Then, WAST_a starts a loop, where in each iteration it locates a task whose completion time is equal to q and tries to transfer it to another site to bring its completion earlier. The transmission is again required to arrive at the destination site before any computing slot at that site becomes available. In the case that multiple sites satisfy the requirement, the algorithm selects the one where the transferred task can be completed the earliest as the destination site (lines 14-18). After the task transmission is identified, the scheduled workload W , network usage T and J_k 's job completion time q and task transmissions X are updated accordingly (lines 22-27). Then, the algorithm moves onto the

next iteration to locate another task whose completion time is equal to the new job completion time and look for its possible transmission. If the task whose completion time equals q is at a site that has received J_k 's tasks in previous iterations (line 12) or cannot find a destination site to have its completion time reduced (line 19), the algorithm terminates and returns all the transmissions identified in previous iterations and the job completion time q (lines 13 and 20). Note that this algorithm is not guaranteed to produce the optimal completion time for job J_k because it searches for task transmissions one at a time. However, it is easier to implement and takes much less time to compute the task transmissions.

F. Complexity Analysis

The outer loop of the WAST algorithm (Algorithm 2) has n iterations, in which it computes the optimal job completion times of all remaining jobs in $\mathcal{J}$ (which decreases by 1 in each iteration) using Algorithm 1. Therefore, Algorithm 2 calls Algorithm 1 for a total of $n(n+1)/2$ times. With ILP transformed into a minimum-cost flow problem, Algorithm 1 solves multiple minimum-cost flow problems to derive a job's optimal completion time, and the number of problems to be solved depends on the job's task number, task distribution, data partition size as well as network resources available. Consider the worst case that a job J_k 's tasks concentrate at a single site initially and the network resources are adequate to ensure that these tasks can be transferred to any other site in time. In this case, all the tasks can be executed at any site, and the total number of possible job completion times for J_k is $F \cdot m$ (where F is the number of tasks in J_k), which requires $\log(F \cdot m)$ rounds in the binary search. The time complexity to solve the minimum-cost flow problem depends on the algorithm applied. Our flow network has $2m + 2$ nodes and $m^2 + 2m$ edges. The Successive Shortest Path Algorithm (using a binary heap) has the worst-case computational complexity of $O(m^3 \log m)$ [13]. Therefore, assuming that all jobs have the same task number F , the worst-case computational complexity of Algorithm 2 is $O(n(n+1)/2 \cdot \log(F \cdot m) \cdot m^3 \log m) = O(n^2 m^3 \log F \log^2 m)$.

The outer loop of the approximation algorithm WAST_a is the same as that of Algorithm 2. In each iteration, WAST_a calls Algorithm 3 to optimize the completion time of every remaining job in $\mathcal{J}$. Algorithm 3 first places the tasks of the job locally, and then iteratively searches for destination sites to transfer tasks whose completion times are equal to the current job completion time. In the worst case, all tasks of the job are to be transferred to other sites to improve the job completion time (for example, all tasks of the job concentrate at one site initially whose scheduled workload is already the highest among all sites), and each task traverses all sites to find the best destination site, so the worst-case computational complexity is $O(F \cdot m)$. Therefore, if Algorithm 2 calls Algorithm 3 instead of Algorithm 1, the worst-case complexity is $O(n(n+1)/2 \cdot F \cdot m) = O(n^2 F m)$. Since no minimum-cost flow problem needs to be solved, this computational complexity is much lower than using Algorithm 1. We shall

empirically evaluate the performance of these two algorithms in the experiments.

G. Deployment

In this section, we discuss how to apply the algorithms introduced above to online scenarios where jobs arrive dynamically for execution. The algorithm can be deployed at a central scheduler of the geo-distributed system. The scheduler keeps track of the system's resource capacities (including the compute resources and the network resources) as well as the demand distribution of each job across all sites. Once the job execution order and the task transmissions of each job are derived, the scheduler propagates them to relevant sites. Whenever a computing slot becomes available at a site S_i , S_i selects the first job in the job execution order that has pending tasks locally and assigns the slot to one task of this job. For each link from a source site to a destination site, the task input data of jobs are transferred based on the task transmission decisions. The transmissions follow the job execution order, so that the tasks of the jobs to be executed earlier are transferred earlier. The central scheduler recomputes the job execution order and the task transmissions of jobs whenever a new job arrives. To do so, it takes a snapshot of the pending task distributions of all jobs as their computational demand distributions. The recomputation derives a new job execution order and new task transmissions for operation. Existing scheduled transmissions not yet completed are withdrawn, and the network usage T is reset to 0 for each recomputation.

V. EXPERIMENTS

Simulation experiments using read-world job traces are conducted to evaluate the performance of applying our scheduling algorithms to online scenarios as discussed in Section IV-G.

A. Datasets

We use cluster job traces sourced from the Alibaba Cluster Trace Program [15], which contains cluster traces of different years from real production. We pre-process the raw trace files of years 2017 and 2018, extract the key information including the arrival time, task number, average memory consumption and task processing times of each job. A continuous sequence of 10,000 jobs is randomly selected from each trace to serve as the dataset driving our simulations. The job characteristics of selected sequences are shown in Fig. 3 and Fig. 4. Both

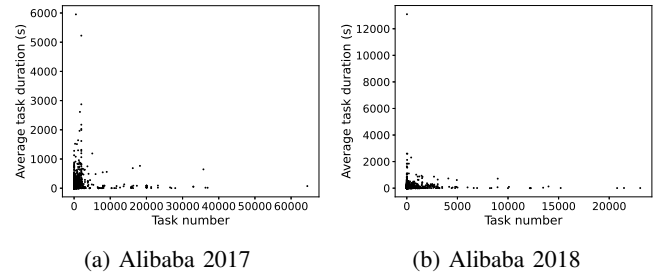


Fig. 3: Job characteristic distribution of two datasets

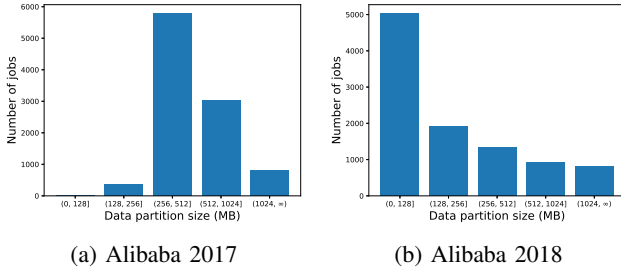


Fig. 4: Data partition size distribution of two datasets

Characteristics	Alibaba 2017	Alibaba 2018
Average task number per job	266.01	94.70
Average task processing time (s)	129.14	82.41
Average data partition size (MB)	661.15	453.31

TABLE II: Job characteristics of two datasets

sequences exhibit a heavy-tailed distribution, in which a small subset of jobs accounts for a disproportionately large share of the total computation load. We take the average memory consumption of each job as its data partition size. The average memory consumption records in the data traces are normalized values with respect to the memory capacity of the servers in the cluster. We derive the exact data partition size assuming that each server has 128GB RAM, under which the largest data partition is about 4GB. The average task number per job, average task processing time and average data partition size of all jobs are shown in Table II.

B. System Configuration

We simulate a distributed system of m sites, each with the same number of computing slots. Each task is assumed to take up one computing slot to run and release it after completion. Due to the switching overhead consideration, we assume that the system does not allow preemption during task execution. For each job, the initial distribution of its tasks among the sites is generated by a Zipf distribution: the numbers of tasks placed at the m sites are proportional to $1, \frac{1}{2^\alpha}, \frac{1}{3^\alpha}, \dots, \frac{1}{m^\alpha}$, where α is a Zipf parameter. A higher α implies a more skewed distribution (a higher proportion of tasks are concentrated at fewer sites), whereas $\alpha = 0$ indicates a uniform distribution. In the experiments, we test four Zipf distributions with α set to 0, 0.5, 1 and 2 to simulate different task distributions of jobs. After the task distribution is generated, we shuffle the order of the sites when deploying each job's task distribution, so that the total workload at each site is roughly balanced. We scale the arrival times of the jobs in the datasets to simulate different levels of overall system utilization. We assume that the system is fully connected, i.e., a direct link exists between any two sites. The available bandwidth is set to 24Mbps, 48Mbps or 96Mbps for all links. We experiment with different numbers of sites, different numbers of computing slots per site, and different levels of system utilization. The results show similar performance trends. Due to space limitations, we focus on presenting the results for $m = 10$ sites, 32 computing slots per site, and system utilization = 50%, 60%, 70% and 80%.

C. Scheduling Policies

In Section IV, we have introduced the WAST algorithm and its fast approximation WAST_a. For comparison, we also simulate the following baseline algorithms.

- **First-Come-First-Serve (FCFS)**: FCFS simply schedules jobs in the order of their arrival times. When a computing slot becomes available, it is assigned to the job that arrives the earliest and has pending tasks.
- **Shortest Remaining Processing Time (SRPT)**: SRPT schedules jobs based on their estimated completion times in the case that each job runs alone in the system. More specifically, it estimates the completion time of a job J_k as $\max_{j \in [m]} \lceil d_{kj}/u_j \rceil \cdot l_k$, and executes pending jobs based on this estimation from low to high.
- **Workload-Aware Greedy Scheduling (SWAG)** [14]: SWAG greedily schedules jobs iteratively by estimating their completion times based on scheduled workloads without using network resources to transfer tasks between sites. Our WAST algorithm degenerates to SWAG when network resources are not available in the system.
- **GEODIS** [16]: GEODIS models job scheduling in geo-distributed data centers as a joint optimization of task placement and data access. It employs a heuristic scheduling algorithm which is triggered when any data center completes its scheduled workload. The objective of scheduling is to minimize the makespan of job execution (the latest job completion time) by selectively migrating data over faster network links when local servers are overloaded.

In the experiments, we use OR-Tools [8] to solve the minimum-cost flow problems in WAST. All algorithms (except FCFS) take the average task processing time of each job as input for scheduling and assume that the processing times of all tasks of the job are equal to the average. The simulation, on the other hand, uses the real processing times of individual tasks recorded in the trace to derive the job response time.

D. Experimental Results

Fig. 5 shows the performance of all algorithms in terms of average job response time. Due to space limitations, we only present the results for the Alibaba 2018 dataset (the results for the Alibaba 2017 dataset are similar). Note that the y-axis is plotted on a logarithmic scale. The results indicate that the average job response times of FCFS and GEODIS are one to two orders of magnitude higher than the other algorithms. This is because FCFS does not optimize job execution order and GEODIS is primarily designed to minimize the makespan.

Figs. 6 and 7 zoom into the performance of the remaining algorithms for both datasets. Compared to SRPT and SWAG, the average job response time of WAST and WAST_a is reduced by up to 53.3% for the Alibaba 2017 trace and 42.1% for the Alibaba 2018 trace. The improvement becomes more pronounced when the task distribution of jobs across sites is more skewed and when higher bandwidth is available, as more network resources can be effectively utilized to mitigate

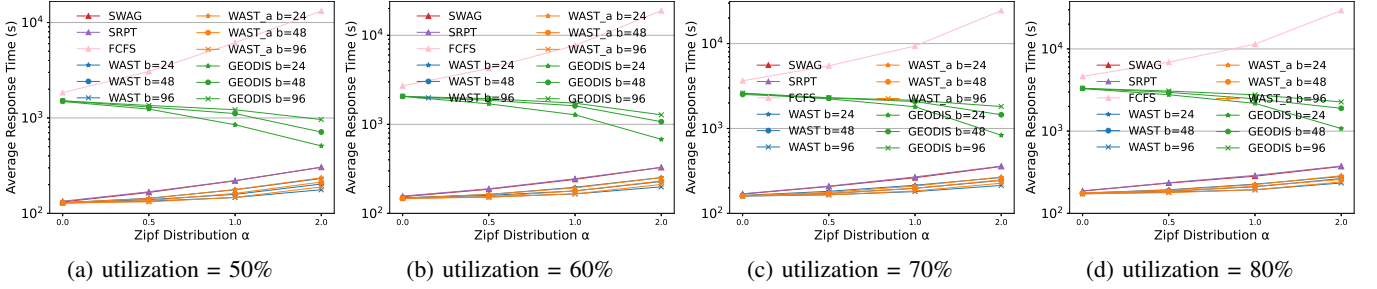


Fig. 5: Average Job Response Time Compared with GEODIS, Alibaba 2018

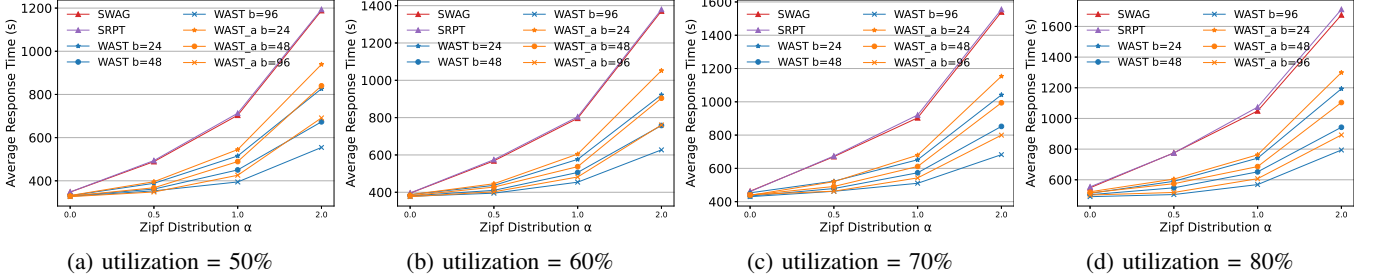


Fig. 6: Average Job Response Time, Alibaba 2017

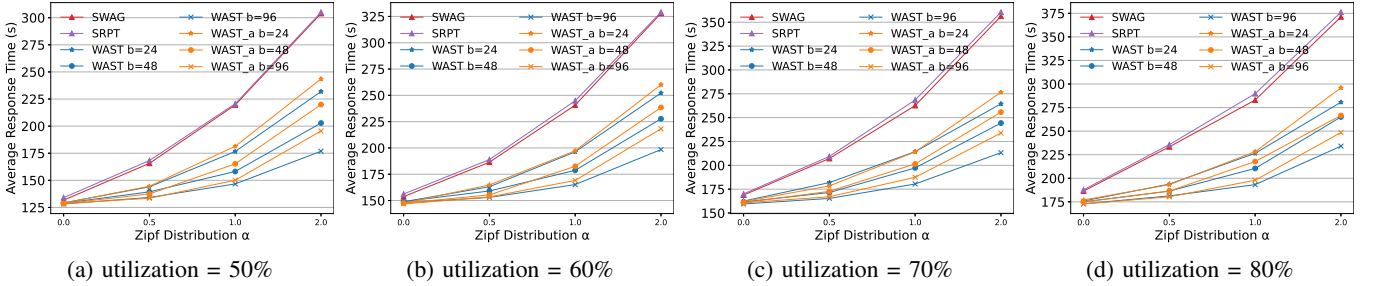


Fig. 7: Average Job Response Time, Alibaba 2018

the imbalance in the task distribution of jobs under these conditions. Moreover, our WAST algorithm performs better for the Alibaba 2017 trace than the Alibaba 2018 trace. This is because the higher average task number per job in Alibaba 2017 leads to more tasks accumulated at some particular sites under highly uneven task distributions, resulting in increased response times if task transmission is not conducted. This, in turn, allows the WAST algorithm to offload more tasks from peak-load sites, enhancing the advantage of WAST. In addition, Figs. 6 and 7 show that although WAST_a compromises the quality of scheduling and performs slightly worse than WAST, it still significantly outperforms the baseline algorithms.

Figs. 8 and 9 show the 30th, 60th, 90th percentiles and the highest job response times for both datasets, under the Zipf distribution with $\alpha = 1$. For each percentile, the longest job response time among algorithms is normalized to 1, and the job response times of other algorithms are expressed relative to it. It can be seen that jobs in the lower percentiles have little improvement when the WAST and WAST_a algorithms are

applied, since these jobs are small and have already achieved nearly best response times under the SRPT and SWAG strategies which favor small jobs. On the other hand, WAST and WAST_a can significantly enhance the performance of large jobs whose response times are relatively longer by transferring their tasks to mitigate their imbalanced task distributions. The highest job response time can be improved by more than 70%, as shown in Fig. 8.

Tables III and IV show the average computation time per recomputation of the SWAG, WAST and WAST_a algorithms under various bandwidth settings, slot numbers per site, Zipf parameters α and system utilization. Due to space limitations, we present the results for the Alibaba 2018 trace under 60% and 80% utilization only. Both tables demonstrate that WAST_a effectively reduces the computation time compared with WAST, which can be up to 87.3%. Furthermore, it can be seen that the average computation time of WAST is about an order of magnitude higher than that of SWAG, while the average computation time of WAST_a is of the same order of magnitude as SWAG, demonstrating the effectiveness of

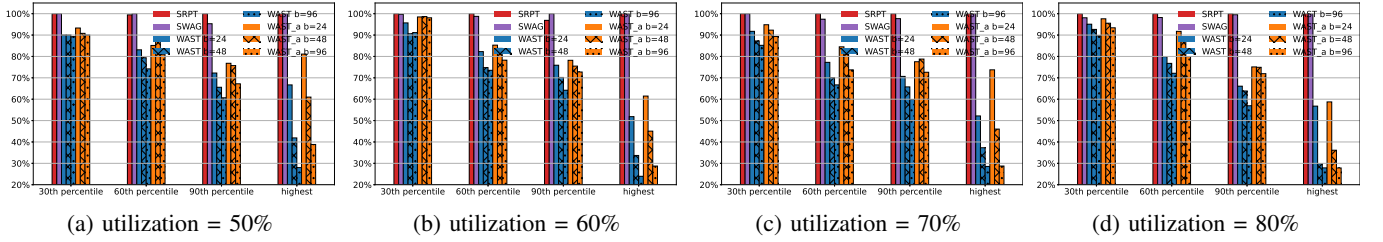


Fig. 8: Job Response Time at Different Percentiles, Zipf Distribution $\alpha = 1$, Alibaba 2017

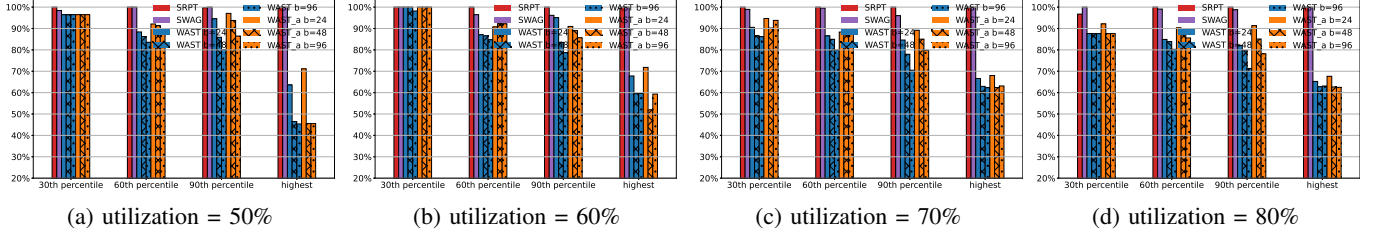


Fig. 9: Job Response Time at Different Percentiles, Zipf Distribution $\alpha = 1$, Alibaba 2018

TABLE III: Average computation time (second), Alibaba 2018 trace, 60% utilization

slot	α	b = 24Mbps			b = 48Mbps		b = 96Mbps	
		SWAG	WAST	WAST_a	WAST	WAST_a	WAST	WAST_a
32	0.0	0.0157	0.1209	0.0201	0.1127	0.0199	0.1005	0.0198
	0.5	0.0184	0.1323	0.0323	0.1167	0.0297	0.1023	0.0274
	1.0	0.0253	0.1805	0.0453	0.1327	0.0468	0.1301	0.0424
	2.0	0.0475	0.3014	0.0756	0.2283	0.0872	0.1936	0.0877
64	0.0	0.0306	0.2293	0.0378	0.2310	0.0399	0.2283	0.0376
	0.5	0.0365	0.2797	0.0585	0.2467	0.0653	0.2330	0.0563
	1.0	0.0502	0.3646	0.0883	0.3281	0.0781	0.2867	0.0845
	2.0	0.0885	0.6217	0.1666	0.5365	0.1299	0.4369	0.1482
128	0.0	0.0772	0.6290	0.0899	0.6169	0.0928	0.6057	0.0922
	0.5	0.0876	0.7265	0.1127	0.7205	0.1228	0.6601	0.1341
	1.0	0.1302	0.9427	0.1459	0.9012	0.1565	0.8432	0.1781
	2.0	0.2068	1.5751	0.2247	1.4975	0.2317	1.3203	0.2493

TABLE IV: Average computation time (second), Alibaba 2018 trace, 80% utilization

slot	α	b = 24Mbps			b = 48Mbps		b = 96Mbps	
		SWAG	WAST	WAST_a	WAST	WAST_a	WAST	WAST_a
32	0.0	0.0326	0.2950	0.0411	0.2782	0.0398	0.2543	0.0393
	0.5	0.0402	0.3212	0.0785	0.2879	0.0616	0.2549	0.0547
	1.0	0.0521	0.4283	0.1078	0.3196	0.1100	0.2513	0.0905
	2.0	0.0964	0.4826	0.1503	0.5172	0.1785	0.4501	0.1746
64	0.0	0.0667	0.5337	0.0863	0.5363	0.0891	0.5340	0.0818
	0.5	0.0842	0.6638	0.1378	0.5871	0.1483	0.5706	0.1266
	1.0	0.1078	0.8532	0.2476	0.7262	0.1848	0.6623	0.2028
	2.0	0.1816	1.2361	0.3728	1.1003	0.2484	0.9437	0.2913
128	0.0	0.1665	1.3935	0.1963	1.4098	0.2024	1.4332	0.2087
	0.5	0.1947	1.7591	0.2664	1.6433	0.2899	1.6000	0.3193
	1.0	0.2715	2.1885	0.3210	2.0776	0.3473	1.8640	0.3722
	2.0	0.4260	3.9332	0.4998	3.6373	0.5136	3.0193	0.5459

WAST_a in practice.

VI. DISCUSSION

Heterogeneous tasks. In real-world workloads, a job may consist of tasks with heterogeneous processing times and data sizes. Our proposed model and algorithms are capable of handling heterogeneous tasks by replacing the parameters l_k and d_k with task-specific values. Although this complicates the

modeling of computation and transmission process, it does not compromise the effectiveness of our proposed algorithms.

Limited network resources. Inter-datacenter bandwidth is often limited and costly compared to intra-datacenter links. Therefore, the scheduler must avoid excessive network resource consumption. We suggest that the algorithm should exclusively utilize the spare network bandwidth, which is identified based on current system conditions. The specific method for quantifying this spare bandwidth via network monitoring is beyond the scope of this paper.

Replicated data inputs. Cloud-based environments often replicate data chunks across multiple geo-distributed servers in order to reduce service response latency and data transfer overhead [17], [18]. Our method can be integrated with algorithms that determine task assignments under replicated data inputs. Specifically, we first execute the task assignment algorithm to determine an initial execution site for each task, and then employ our proposed algorithm(s) to derive the job execution order and task transmissions. Notably, if a destination site is found to already host a replica of the required data partition, the corresponding transmission is not needed, and the task can be processed locally at the destination.

VII. RELATED WORK

The task (job) scheduling problem in geo-distributed environments has been widely studied in many aspects. Some research focuses on utilizing the game theory to minimize the system entity cost by assuming that users in the system have different interests, and thus they form a game model [19], [20]. Many efforts aim to minimize the bandwidth usage or the transmission time/cost in the system. For example, [21] focuses on optimizing the transmission time of data along the given paths, Gaia [22] tries to minimize the transmission cost between machine learning clusters while maintaining the model's performance, and ELIS [23] aims to save the

network bandwidth usage on both intra-DC and inter-DC links while ensuring the QoS of microservices. Previous studies have also made progress on developing efficient distributed job scheduling algorithms. However, they typically rely on simplified system assumptions which limit their applicability in real practice [14], [16]–[18], [24]–[26]. For example, Hung *et al.* [14] presents a workload-aware strategy which schedules jobs based on their estimated completion times under given task placements, but it neglects to utilize network resources to move tasks. Guan *et al.* [17], Zhao *et al.* [18] study the task assignment problem given that the input data of each task can be replicated across multiple geo-distributed data centers, and they do not consider using network resources either. Hu *et al.* [24] and Pu *et al.* [25] introduce algorithms considering task transmissions, but they focus on improving the efficiency of a single job and do not study the scheduling strategy across jobs. Chen *et al.* [26] presents a scheduling algorithm that incorporates transmissions but imposes strict constraints that the number of tasks is no more than the number of computing slots in the system. Convolbo *et al.* [16] coordinates scheduling with transmission but focuses on minimizing the makespan of job execution instead of the response times of individual jobs. Different from the above work, we study task scheduling and transmission for optimizing job response times without making those simplifying assumptions.

VIII. CONCLUSION

In this paper, we have proposed a WAST algorithm and a fast approximation to improve job scheduling efficiency in geo-distributed systems with the usage of network resources. WAST constructs a near-optimal job execution order with task transmissions progressively. In each round, WAST searches for the job with the minimum completion time among all unsettled jobs given the scheduled workloads and task transmissions of previous rounds. The sub-problem of minimizing a single job's completion time is converted into solving multiple minimum-cost flow problems, which can be computed efficiently. We also present a fast approximation to WAST, which optimizes each job's completion time by transferring its tasks of the latest completion time to other sites recursively. We have evaluated WAST and its approximation using real job traces from Alibaba. The experimental results show that WAST can effectively reduce the average job response time, especially when the task distributions of jobs are highly skewed.

ACKNOWLEDGMENT

This research is supported by the Ministry of Education, Singapore, under its AcRF Tier 2 (Awards MOE-T2EP20121-0005, MOE-T2EP20125-0001) and Tier 1 (Award RG15/25).

REFERENCES

- [1] S. Tang, Y. Yu, H. Wang, G. Wang, W. Chen, Z. Xu, S. Guo, W. Gao, A survey on scheduling techniques in computing and network convergence, *IEEE Communications Surveys & Tutorials* 26 (1) (2023) 160–195.
- [2] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al., *Spanner: Google's globally distributed database*, *ACM Transactions on Computer Systems (TOCS)* 31 (3) (2013) 1–22.
- [3] C. Li, H. Chen, S. Zhang, Y. Hu, C. Chen, Z. Zhang, M. Li, X. Li, D. Han, X. Chen, et al., *Bytegraph: a high-performance distributed graph database in bytedance*, *Proceedings of the VLDB Endowment* 15 (12) (2022) 3306–3318.
- [4] K. Shvachko, H. Kuang, S. Radia, R. Chansler, *The hadoop distributed file system*, in: *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, Ieee, 2010, pp. 1–10.
- [5] J. G. Shanahan, L. Dai, *Large scale distributed data science using apache spark*, in: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 2323–2324.
- [6] N. Hogade, S. Pasricha, *A survey on machine learning for geo-distributed cloud data center management*, *IEEE Transactions on Sustainable Computing* 8 (1) (2022) 15–31.
- [7] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, J. S. Rellermeyer, *A survey on distributed machine learning*, *Acm computing surveys (csur)* 53 (2) (2020) 1–33.
- [8] L. Perron, V. Furnon, *Or-tools*, <https://developers.google.com/optimization/>, google Optimization Tools (2023).
- [9] IBM, *IBM ILOG CPLEX optimization studio*, <https://www.ibm.com/products/ilog-cplex-optimization-studio> (2024). URL <https://www.ibm.com/products/ilog-cplex-optimization-studio>
- [10] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, <https://www.gurobi.com> (2024).
- [11] A. Jajoo, Y. C. Hu, X. Lin, N. Deng, *Slearn: A case for task sampling based learning for cluster job scheduling*, *IEEE Transactions on Cloud Computing* 11 (3) (2022) 2664–2680.
- [12] E. L. Lawler, *Combinatorial optimization: networks and matroids*, Courier Corporation, 2001.
- [13] P. Kovács, *Minimum-cost flow algorithms: an experimental evaluation*, *Optimization Methods and Software* 30 (1) (2015) 94–127.
- [14] C.-C. Hung, L. Golubchik, M. Yu, *Scheduling jobs across geo-distributed datacenters*, in: *Proceedings of the Sixth ACM Symposium on Cloud Computing*, 2015, pp. 111–124.
- [15] Alibaba, *Alibaba cluster trace program*, <https://github.com/alibaba/clusterdata> (2017). URL <https://github.com/alibaba/clusterdata>
- [16] M. W. Convolbo, J. Chou, C.-H. Hsu, Y. C. Chung, *Geodis: towards the optimization of data locality-aware job scheduling in geo-distributed data centers*, *Computing* 100 (1) (2018) 21–46.
- [17] Y. Guan, X. Tang, *On task assignment and scheduling for distributed job execution*, in: *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, IEEE, 2022, pp. 726–735.
- [18] H. Zhao, X. Tang, P. Chen, J. Yin, S. Deng, *Data-locality-aware task assignment and scheduling for distributed job executions*, *IEEE Transactions on Services Computing* 18 (5) (2025) 2701–2713.
- [19] X. Chen, *Decentralized computation offloading game for mobile cloud computing*, *IEEE Transactions on Parallel and Distributed Systems* 26 (4) (2014) 974–983.
- [20] N. Hogade, S. Pasricha, H. J. Siegel, *Energy and network aware workload management for geographically distributed data centers*, *IEEE Transactions on Sustainable Computing* 7 (2) (2022) 400–413.
- [21] L. Chen, S. Liu, B. Li, *Optimizing network transfers for data analytic jobs across geo-distributed datacenters*, *IEEE Transactions on Parallel and Distributed Systems* 33 (2) (2021) 403–414.
- [22] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, O. Mutlu, *Gaia: Geo-Distributed machine learning approaching LAN speeds*, in: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, USENIX Association, 2017, pp. 629–647.
- [23] J. Shi, K. Fu, J. Wang, Q. Chen, D. Zeng, M. Guo, *Adaptive qos-aware microservice deployment with excessive loads via intra-and inter-datacenter scheduling*, *IEEE Transactions on Parallel and Distributed Systems* 35 (9) (2024) 1565–1582.
- [24] Z. Hu, B. Li, J. Luo, *Flutter: Scheduling tasks closer to data across geo-distributed datacenters*, in: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, IEEE, 2016, pp. 1–9.
- [25] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, I. Stoica, *Low latency geo-distributed data analytics*, *ACM SIGCOMM Computer Communication Review* 45 (4) (2015) 421–434.
- [26] L. Chen, S. Liu, B. Li, B. Li, *Scheduling jobs across geo-distributed datacenters with max-min fairness*, *IEEE Transactions on Network Science and Engineering* 6 (3) (2018) 488–500.