Scheduling Multi-Jobs across Geo-Distributed Datacenters with Max-Min Fairness

Ideas, Analysis and Realization

Xinran Li (519030910121, electron1@sjtu.edu.cn), Tang Peng (517020910038, 85704592@qq.com), Kuiyuan Huang (519030910183, huang_kuiyuan@qq.com)

Department of Computer Science,
Shanghai Jiao Tong University, Shanghai, China

Abstract. It has now become commonly accepted that the volume of data—from end users, sensors, and algorithms—has been growing exponentially, and mostly stored in geographically distributed data centers around the world. In this paper, we focus on the problem of assigning tasks belonging to *multiple* jobs across datacenters, with the specific objective of achieving *max-min fairness* across jobs sharing these datacenters, in terms of their job completion times, considering both data locality and network bandwidth across datacenters. To address this problem, we use algorithm on directed acyclic graph(*DAG*) as well as data structure and iteratively solve its single-objective subproblems, which can be transformed to equivalent linear programming (*LP*) problems to be efficiently solved. Then we realize our ideas by programming to handle some examples and finding solution.

Keywords: Geo-distributed Datacenter Networks, Multi-Jobs Scheduling, Max-Min Fairness.

1 Background and Introduction

It has become routine for large volumes of data to be generated, stored, and processed across geographically distributed datacenters. A single data analytic job typically proceeds in consecutive computation stages, each of which consisting of a number of computation tasks that are executed in parallel. To start a new computation stage, intermediate data from the preceding stage needs to be fetched, which may initiate multiple network flows. In the meantime, there may exist multiple jobs to proceed in a project.

When needed data is located across multiple geo-distributed datacenters, a naive and simple approach is to gather all the data to be processed locally within a single datacenter. Nevertheless, transferring such huge amounts of data across datacenters may be slow and inefficient, since bandwidth on interdatacenter network links is limited.

In this paper, we come up with an tasks assignment strategy which is designed to achieve max-min fairness across multiple jobs by using algorithms on *DAG*, as they compete for the limited slots of shared resources across multiple geo-distributed datacenters. At first in Sec. 2, we illustrate the problem systematacially. Then we formalize the problem with objective functions and do the hardness analysis. Furthermore, we with to minimize the job completion time across all concurrent jobs, while maintaining max-min fairness. In theoretical work in Sec. 3 and Sec. 4, we introduce methodologies such as *lexicographical minimization*, *convex optimization*, *unimodular matrix*, λ-*representation* to help to achieve our intention, which is the highlight idea in our work. In Sec. 5, we realize our idea with some examples and show our solutions and visualization.

2 Problem Description and Analysis

This section is about the definitions, formalization, and the hardness analysis of this problem. We sum up all definitions, variables, functions, theorems and lemmas arose from this part on in appendix **??**.

2.1 Multi-Job Scheduling with Max-Min Fairness

To start with, we introduce some concepts and definitions in this problem. In order to avoid tedious repetition, we omit the introduction of *Single Job Scheduling*, which is the basic case of multiple one.

Consider a set of jobs $\mathcal{K} = \{1, 2, \cdots, K\}$ submitted to the scheduler for tasks arrangement in a project. The needed input data of these jobs are distributed across a set of geo-distributed datacenters, represented

by $\mathcal{D} = \{1, 2, \cdots, J\}$, each of them in which connects with each other, constituting a network together. Each job $k \in \mathcal{K}$ has a set of tasks $\mathcal{T}_k = \{1, 2, \cdots, n_k\}$ is allowed to be launched on one available computing slots in datacenters. We use a_j to denote the capacity of available computing slots in datacenter $j \in \mathcal{D}$.

For each task $i \in \mathcal{T}_k$ of job k, the time it takes to complete consists of both the network transfer time, denoted by $c_{i,j}^k$, to fetch the input data if the task is assigned to datacenter j, and the execution time represented by $e_{i,j}^k$. The network transfer time is decided by both the amount of data to be read and the bandwidth on the link the data traverses. Let S_i^k denotes the set of datacenters where the input data of task i from job k are stored, called the source datacenters of this task for convenience. The task needs to read the input data from each of its source datacenters $s \in S_i^k$, the amount of which is represented by $d_i^{k,s}$. Let $b_{s,j}$ represent the bandwidth of the link from datacenter s to j $(s \neq j)$. The bandwidth in the same datacenter is considered as infinity, which indicates that the network transfer time is 0. Hence, the transfer time of task $i \in \mathcal{T}_k$, if assigned to datacenter j, is expressed as follows:

$$c_{i,j}^k = \begin{cases} 0, & \text{when } S_i^k = \{j\}; \\ \max_{s \in S_i^k,\ s \neq j} d_i^{k,s}/b_{s,j}, & \text{otherwise.} \end{cases} \tag{1}$$

The assignment of a task is represented with a binary variable $x_{i,j}^k$, indicating whether the job i of job k is assigned to datacenter j. Let ST_i^k and FT_i^k denote the start-time and finish-time of task i of job k. Respectively, the complete time of a task i is $FT_i^k - ST_i^k$. Intuitively, a job k completes when its slowest task finishes, thus the completion time of job k, represented by τ_k, is expressed as follows:

$$\tau_k = \max_{i \in \mathcal{T}_k,\ j \in \mathcal{D}} x_{i,j}^k (c_{i,j}^k + e_{i,j}^k) \tag{2}$$

As the computing slots in all the datacenters are shared by tasks from multiple jobs, we would like to obtain an optimal task assignment without exceeding the resource capacities by modifying $x_{i,j}^k$. That is, deciding the assignments of all the tasks, aiming to optimize the worst performance achieved among all the jobs with respect to their job completion times, and then optimize the next worst performance without impacting the previous one.

Max-Min Fairness According to the definition in Chapter 5 of *Ivan Marsic, Computer Networks: Performance and Quality of Service* [1], A sharing technique widely used in practice is called *max-min fair* share. Intuitively, a fair share allocates a user with a "small" demand what it wants, and evenly distributes unused resources to the "big" users. Formally, we define *max-min fair* share allocation to be as follows:

I. Resources are allocated in order of increasing demand

II. No source gets a resource share larger than its demand

III. Sources with unsatisfied demands get an equal share of the resource

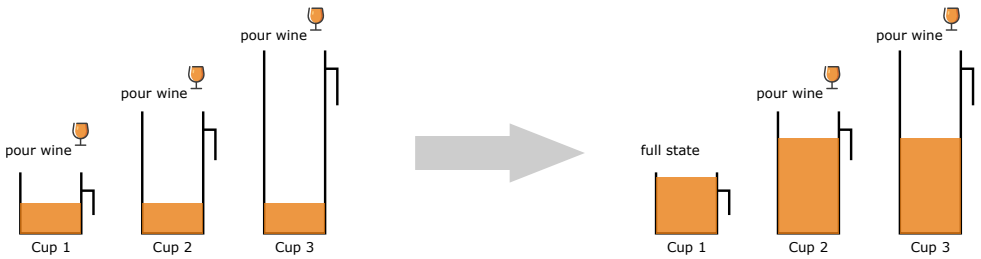

Fig. 1. Before the cup with minimum capacity is filled fully, we keep pouring wine in all cups at the same speed. Once there exists a new full-filled cup, we stop pouring wine in it and keep pouring wine in other cups at the same speed.

This formal definition corresponds to the following operational definition. Consider a set of sources $\{1, 2, \cdots, n\}$ that have resource demands $\{x_1, x_2, \cdots, x_n\}$. Without loss of generality, order the source demands so that $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_n$. Let the server have capacity C. Then, we initially give C/n of the resource to the source with the smallest demand, x_1. This may be more than what source 1 wants, perhaps, so we can continue the process. The process ends when each source gets no more than what it asks for, and, if its demand was not satisfied, no less than what any other source with a higher index got. We call such an allocation a *max-min fair* allocation, because it maximizes the minimum share of a source whose demand is not fully satisfied. We have a figure to show this idea (See Fig. 1).

The realization of *max-min fairness* will be shown later.

2.2 Formalization with notations

With the definitions above, we have our objective functions $\mathscr{F}$:

$$\min \frac{\sum_{i=1}^{n}(FT_i^k - ST_i^k)}{n} \tag{3}$$

and

$$\min \max_{1 \leqslant l,m \leqslant \mathcal{K}} (|\tau_m - \tau_l|) \tag{4}$$

To make the trade-off, we add a parameter $\mathscr{L}$ into them,

$$\mathscr{F} = \mathscr{L} \min \frac{\sum_{i=1}^{n}(FT_i^k - ST_i^k)}{n} + (1 - \mathscr{L}) \min \max_{1 \leqslant l,m \leqslant \mathcal{K}} (|\tau_m - \tau_l|) \tag{5}$$

By changing the value of $\mathscr{L}$, we can get the minimum value of our objective function $\mathscr{F}$.

2.3 Hardness Analysis

In this part, we show that this problem is a NP-complete problem.

Theorem 1. *The Multi-Jobs across Geo-Distributed Datacenters with Max-Min Fairness Scheduling Problem is NP-complete*

Proof. We prove this theorem by reduction from the Makespan Problem [6], which is NP-complete. The optimization version of the Makespan Problem is defined as follows: Let m ($m \geqslant 2$) be the number of machines and let n be the number of jobs. For each job $J_1, J_2, \cdots, J_n$ it has a process time $p_1, p_2, \cdots, p_n$. We need to find a function $F, \forall i \in 1, 2, \cdots, n, i \xrightarrow{F} j, j \in 1, 2, \cdots, m$. The optimization is to minimize: $\sum_{\substack{F(i)=j \\ j}}(p_i)$.

Since we have know that when $m \geqslant 2$ the Makespan problem is a NP-complete problem [6]. Then we show that for any instance of Makespan$(m, n, p_1, p_2, \cdots, p_n)$ we can reduce it to an instance of multi-Jobs across geo-distributed datacenters scheduling problem$(\sum_{j=1}^{J} a_j, \sum_{k=1}^{K} n_k, \tau_l^1, \tau_2^1, \cdots, \tau_i^k, \cdots, \tau_{n_k}^K)$ in polynomial time.

When allocating a job J_i to machine j, we assign a task i to one of slot in datacenter $s \in S_i^k$. With same execution time, $\tau_i^k > p_i$ for task i need waiting for the data transfer. For a process time, we can use $O(1)$ to create a piece of data and hence we can reduce from the Makespan problem instance by $O(n)$. So, we have the instance of multi-Jobs across geo-distributed datacenters scheduling problem now and the reduce complexity is polynomial time.

So, we have proved that any instance of Makespan problem can be reduced to a instance of multi-Jobs across geo-distributed datacenters scheduling problem. Hence in NP, multi-Jobs across geo-distributed datacenters scheduling problem has been proved as NP-complete.

3 Model and Formulation

Having given the definitions above in Sec. 2, we give our methodology to transfer this problem to a linear programming(LP) problem.

3.1 Lexicographical Minimization Transformation

Firstly, our objective can be rigorously formulated as a *lexicographical minimization* problem, with the following definitions as its basis.

Definition 1. *Let $\langle \upsilon \rangle_k$ denote the k-th ($1 \leqslant k \leqslant K$) largest element of $\upsilon \in \mathbb{Z}^K$, implying $\langle \upsilon \rangle_1 \geqslant \langle \upsilon \rangle_2 \geqslant \cdots \geqslant \langle \upsilon \rangle_K$. Intuitively, $\boldsymbol{\langle \upsilon \rangle} = (\langle \upsilon \rangle_1, \langle \upsilon \rangle_2, \cdots, \langle \upsilon \rangle_K)$ represents the non-increasingly sorted version of υ.*

Definition 2. *For any $\boldsymbol{\alpha} \in \mathbb{Z}^K$ and $\boldsymbol{\beta} \in \mathbb{Z}^K$, if $\langle \alpha \rangle_1 < \langle \beta \rangle_1$ or $\exists k \in \{2, 3, \cdots, K\}$ such that $\langle \alpha \rangle_k < \langle \beta \rangle_k$ and $\langle \alpha \rangle_i = \langle \beta \rangle_i, \forall i \in \{1, \cdots, k-1\}$, then $\boldsymbol{\alpha}$ is lexicographically smaller than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$. If $\langle \alpha \rangle_k = \langle \beta \rangle_k, \forall k \in \{1, 2, \cdots, K\}$ or $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$, then $\boldsymbol{\alpha}$ is lexicographically no greater than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \preceq \boldsymbol{\beta}$.*

Definition 3. $\underset{\boldsymbol{x}}{\text{lexmin}} \boldsymbol{f}(\boldsymbol{x})$ *represents the lexicographical minimization of the vector $\boldsymbol{f} \in \mathbb{R}^N$, which consists of N objective functions of $\boldsymbol{x}$. The optimal solution $\boldsymbol{x}^* \in \mathbb{R}^K$ achieves the optimal $\boldsymbol{f}^*$, such that $\boldsymbol{f}^* = \boldsymbol{f}(\boldsymbol{x}^*) \preceq \boldsymbol{f}(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathbb{R}^K$.*

With these definitions, we are now ready to formulate our optimal task assignment problem among sharing jobs as follows:

$$\underset{\mathbf{x}}{\text{lexmin}}\ \boldsymbol{f} = (\tau_1, \tau_2, \cdots, \tau_K) \tag{6}$$

$$\text{s.t.} \quad \tau_k = \max_{i \in \mathcal{T}_k,\ j \in \mathcal{D}} x_{i,j}^k (c_{i,j}^k + e_{i,j}^k), \forall k \in \mathcal{K} \tag{7}$$

$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{T}_k} x_{i,j}^k \leqslant a_j, \forall j \in \mathcal{D} \tag{8}$$

$$\sum_{j \in \mathcal{D}} x_{i,j}^k = 1, \forall i \in \mathcal{T}_k, \forall k \in \mathcal{K} \tag{9}$$

$$x_{i,j}^k \in \{0, 1\}, \forall i \in \mathcal{T}_k, \forall j \in \mathcal{D}, \forall k \in \mathcal{K} \tag{10}$$

where where constraint (7) represents the completion time of each job k as aforementioned. Constraint (8) indicates that the total number of tasks to be assigned to datacenter j does not exceed its capacity a_j , which is the total number of available computing slots. Constraint (9) implies that each task should be assigned to a single datacenter.

With this idea, the objective is a vector $\boldsymbol{f} \in \mathbb{R}^K$ with K elements, each standing for the completion time of a particular job $k \in \mathcal{K}$. According to the previous definitions, we can find the $\boldsymbol{f}^*$. Then among all $\boldsymbol{f}$ with the same worst completion time, the first worst completion time in $\boldsymbol{f}^*$ is the minimum. In this way, solving this problem is to find an optimal assignment vector $\boldsymbol{x}^*$, with all the job completion times are minimized.

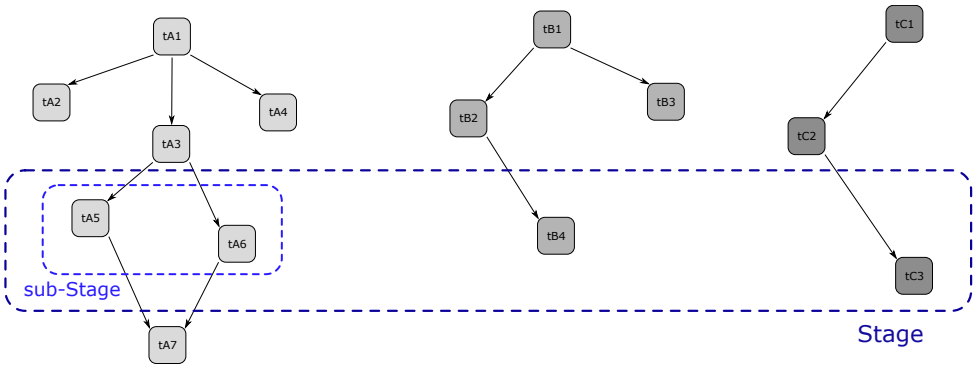

Fig. 2. Examples pf Stage and sub-Stage

Definition 4. *Since the relation between several tasks in a job, we use directed acyclic graph(DAG) to represent the relation of all jobs. In one job, if a task t_i is at the next layer of t_j, needing the data calculated by previous tasks, such as t_j here, it must wait for the completion of upper layer. We call the tasks of all jobs which are in the same layer of this DAG a* ***stage****, and call the tasks of one job which are in the same layer of the sub-DAG of this job a* ***sub-stage****(See Fig. 2). In one stage, all tasks can be executed in parallel intuitively.*

Using the concept of vector optimization, we consider the subproblem in a sub-stage of optimizing the worst performance as follows:

$$\min_{\boldsymbol{x}} \max_{k\in\mathcal{K}}(\tau_k) \tag{11}$$

$$\text{s.t.} \quad \text{Constraints}(7),(8),(9)\text{and}(10).$$

After adding constraint (7) in it, we have:

$$\min_{\boldsymbol{x}} \max_{k\in\mathcal{K}}(\max_{i\in\mathcal{T}_k,\ j\in\mathcal{D}} x_{i,j}^k(c_{i,j}^k + e_{i,j}^k)) \tag{12}$$

$$\text{s.t.} \quad \text{Constraints}(8),(9)\text{and}(10).$$

which is a primary step towards solving the original problem, to be elaborated in the next sections.

3.2 Separable Convex Objective Transformation

In the first step, we will show that the optimal solution for Problem (12) can be obtained by solving a problem with a separable convex objective function [2], which is represented as a summation of convex functions with respect to each single variable $x_{i,j}^k$.

We first show that the optimal solution of Problem (12) can be obtained by solving the following problem:

$$\operatorname*{lexmin}_{\mathbf{x}} \boldsymbol{g} = (\phi(x_{1,1}^1),\cdots,\phi(x_{i,j}^k),\cdots,\phi(x_{n_K,J}^K))$$

$$\text{s.t.} \quad \text{Constraints}(8),(9)\text{ and }(10).$$

where $\phi(x_{i,j}^k) = x_{i,j}^k(c_{i,j}^k + e_{i,j}^k), \forall i \in \mathcal{T}_k, \forall j \in \mathcal{D}, \forall k \in \mathcal{K}$, and $\boldsymbol{g}$ is a vector with the dimension of $M = |\boldsymbol{g}| = J\sum_{k=1}^K n_k$.

Let $\varphi(\boldsymbol{g})$ define a function of $\boldsymbol{g}$:

$$\varphi(\boldsymbol{g}) = \sum_{m=1}^{|\boldsymbol{g}|} |\boldsymbol{g}|^{g_m} = \sum_{m=1}^{M} M^{g_m}$$

where g_m is the m-th element of $\boldsymbol{g}$.

Lemma 1. *$\varphi(\cdot)$ preserves the order of lexicographically no greater($\preceq$), i.e. $\boldsymbol{g}(\boldsymbol{x}^*) \preceq \boldsymbol{g}(\boldsymbol{x}) \Longleftrightarrow \varphi(\boldsymbol{g}(\boldsymbol{x}^*)) \leqslant \varphi(\boldsymbol{g}(\boldsymbol{x}))$.*

Here we put the proof of this lemma in appendix 5.2.

Based on Lemma 1, we have

$$\operatorname*{lexmin}_{\mathbf{x}} \boldsymbol{g} \Longleftrightarrow \min_{\boldsymbol{x}} \varphi(\boldsymbol{g}) = \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{T}_k}\sum_{j\in\mathcal{D}} M^{\phi(x_{i,j}^k)}$$

where the objective function $\varphi(\boldsymbol{g})$ is a summation of the term $M^{\phi(x_{i,j}^k)}$, which is a convex function of the single variable $x_{i,j}^k$.

Therefore, solving Problem (12) is equivalent to solving the following problem with a separable convex objective:

$$\min_{\boldsymbol{x}} \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{T}_k}\sum_{j\in\mathcal{D}} M^{\phi(x_{i,j}^k)} \tag{13}$$

$$\text{s.t.} \quad \text{Constraints}(8),(9)\text{and}(10).$$

3.3 Totally Unimodular Linear Constraints

In this part, we investigate the coefficient matrix of linear constraints (8) and (9).

A m-by-n matrix is *totally unimodular* [3,5], if it satisfies two conditions:

I. any of its elements belongs to {-1,0,1}

II. any row subset $R \subset \{1, 2, \cdots, m\}$ can be divided into two disjoint sets, R_1 and R_2, such that $|\sum_{i \in R_1} a_i j - \sum_{i \in R_2} a_i j| \leqslant 1, \forall j \in \{1, 2, \cdots n\}$.

Lemma 2. *The coefficients of constraints (8) and (9) form a totally unimodular matrix.*

Here we put the proof of this lemma in appendix 5.2.

3.4 Equivalent LP Transformation

Exploiting the problem structure of totally unimodular constraints and separable convex objective, we can use the λ-representation technique [4] to transform Problem (10) to a linear programming problem that has the same optimal solution.

For a single integer variable $y \in \mathcal{Y} = \{0, 1, \cdots, Y\}$, the convex function $h : \mathcal{Y} \to \mathbb{R}$ can be linearized with the λ-representation as follows:

$$h(y) = \sum_{s \in \mathcal{Y}} h(s)\lambda_s, \ y = \sum_{s \in \mathcal{Y}} s\lambda_s \sum_{s \in \mathcal{Y}} \lambda_s = 1, \ \lambda_s \in \mathbb{R}^+, \ \forall s \in \mathcal{Y}.$$

In our problem, we apply the λ-representation technique to each convex function $h^k_{i,j} = M^{\phi(x^k_{i,j})}$: $\{0, 1\} \to \mathbb{R}$ as follows:

$$h^k_{i,j}(x^k_{i,j}) = \sum_{s \in \{0,1\}} M^{s(c^k_{i,j}+e^k_{i,j})}\lambda^{k,s}_{i,j} = \lambda^{k,0}_{i,j} + M^{c^k_{i,j}+e^k_{i,j}}\lambda^{k,1}_{i,j}$$

which removes the variable $x^k_{i,j}$ by sampling at each of its possible value $s \in \{0, 1\}$, weighted by the newly introduced variables $\lambda^{k,s}_{i,j} \in \mathbb{R}^+, \forall s \in \{0, 1\}$ that satisfy

$$x^k_{i,j} = \sum_{s \in \{0,1\}} s\lambda^{k,s}_{i,j} = \lambda^{k,1}_{i,j} \sum_{s \in \{0,1\}} \lambda^{k,s}_{i,j} = \lambda^{k,0}_{i,j} + \lambda^{k,1}_{i,j} = 1$$

Furthermore, with linear relaxation on the integer constraints (10), we obtain the following linear programming problem:

$$\min_{\boldsymbol{x},\boldsymbol{\lambda}} \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{T}_k} \sum_{j \in \mathcal{D}} (\lambda^{k,0}_{i,j} + M^{c^k_{i,j}+e^k_{i,j}}\lambda^{k,1}_{i,j}) \tag{14}$$

$$\text{s.t.} \quad x^k_{i,j} = \lambda^{k,1}_{i,j}, \ \lambda^{k,0}_{i,j} + \lambda^{k,1}_{i,j} = 1, \ \lambda^{k,0}_{i,j}, \lambda^{k,1}_{i,j}, x^k_{i,j} \in \mathbb{R}^+ \ (\forall k \in \mathcal{K}, i \in \mathcal{T}_k, j \in \mathcal{D})$$

$$\text{Constraints(5)and(6).}$$

Theorem 2. *An optimal solution to Problem (14) is an optimal solution to Problem (11).*

Proof. The property of total unimodularity ensures that an optimal solution to the relaxed *LP* problem (14) has integer values of $x^k_{i,j}$, which is an optimal solution to problem (13), and thus an optimal solution to problem (12) as demonstrated. Moreover, Problem (11) and (12) are equivalent forms.

Therefore, the optimal assignment that minimizes the worst completion time among all the jobs can be obtained by solving Problem (14), a *LP* problem.

4 Optimizing the Worst Completion Time

In this section, we continue to solve our original multi-objective problem (6) by coming up with an algorithm in the perspective of stages defined above to minimize the next worst completion time repeatedly.

We want all jobs to work concurrently at the beginning. In every stage G, which is a set of sub-stages, represented as g, we realize arrangement with *max-min fairness* by iteratively finding the longest-executing-time sub-stage g^*, for this means the bottleneck in a stage. Besides, after solving the subproblem g^*, it is known that the optimal worst completion time is achieved by sub-stage g^*, whose slowest task $i*$ is assigned to datacenter j^*. We then fix the computed assignment of the slowest sub-stage g^*, which means that the corresponding schedule variable $x_{i,j}^g$ is removed from the variable set $\boldsymbol{x}$ for the next round. Also, since task i^* is to be assigned to datacenter j^*, it is intuitive that all the assignment variables associated with it should be fixed as zero and removed from $\boldsymbol{x}$: $x_{i^*,j^*}^{g^*}, \forall j \neq j^*, j \in \mathcal{D}$. Considering the parallelism of other sub-stages, The resource capacity constraints should be updated as $\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{T}_k,(k,i)\neq(k^*,i^*)} x_{i,j^*} \leqslant a_{j^*} - 1$.

4.1 Algorithm

To illustrate our ideas, we write a pseudo code here.(See Algorithm 1)

In our design, to satisfy the *max-min fairness*, next stage must wait for upper stage, which indicates we strictly arrange tasks in stages. Therefore, to arrange several stages here, we enqueue each stage and arrange them in order.

Algorithm 1: Performance-Optimal Tasks Assignment among sub-Stages with Max-Min Fairness

Input: $d_i^{k,s}, b_{s,j}, c_{i,j}^k, e_{i,j}^k, a_j$
Output: $x_{i,j}^k$

Initialize $G' = G$;
while $G' \neq \varnothing$ **do**
  solve LP problem (14) $\Rightarrow x_{i^*,j^*}^{g^*} = \underset{x_{i,j}^g \in \boldsymbol{x}}{\text{argmax}}\ \phi(x_{i,j}^g)$;
  Fix $x_{i^*,j}^{g^*}$;
  Update capacity of $s \in S_i^g$;
  Set $\phi(x_{i,j}^{g^*}) = x_{i^*,j^*}^{g^*}(c_{i^*,j^*}^{g^*} + e_{i^*,j^*}^{g^*})$;
  Remove g^* from G';
end

Algorithm 2 is to treat with all stages and complete the project of assigning all of the jobs.

Algorithm 2: Performance-Optimal Tasks Assignment among Stages

Input: $x_{i,j}^k, \mathcal{K}, \mathcal{D}$
Output: $\tau_i, \mathcal{T}_k$

Initialize DAG with given jobs information;
foreach *stage* **do**
  Solve assigning stage with Algorithm 1;
  Record the $\tau_i, \mathcal{T}_k$;
end

4.2 Complexity Analysis

Assume that the input size of tasks is n, total data slots number is m. In the worst case, we need to split them into n stages, indicating that we need to go through this loop for n times.

In every loop, we solve a *LP* problem, which takes exponential time $O(2^{n+m})$ if we use simplex algorithm, then we update and delete our data and elements in $O(n \cdot m)$ time.

Totally, we can solve this problem in $O(n \cdot m \cdot 2^{n+m})$.

For space complexity, we store the main process information $x_{i,j}^k$ in three-dimension-matrix X, which is the largest data to store. Thus the space complexity is $O(n \cdot m)$

However, with the use of totally unimodular technique, this problem may be simplified. And in our realization with *Python*, we use a ready-made library function to solve the most difficult *LP* constraint equations, improving the efficiency of solving this problem. We use extra space in our code to store many other information to help us analysis the efficiency better.

5 Performance Evaluation

Having proved the theoretical optimality and efficiency of our scheduling solution, we proceed to implement it with *Python* on given example, *ToyData.xlsx*, and demonstrate its effectiveness in optimizing job completion times in visual format. Moreover, we present an extensive array of large-scale simulation results to demonstrate the effectiveness of our algorithm in achieving max-min fair job performance.

5.1 Implementation of Sample File

Design and Implementation We use *Python* to write and realize our idea(See *allocateMain.py*). As mentioned above, we deal with the tasks in form of *DAG*, and store the necessary data in various forms of data structures. We implement the algorithm 1 in function *AllocStage*, which calls function *liner* to solve *LP* sub-problem and calls *Find_k* to g^* (the symbol may be different from the paper mentioned ones for processing convenience).

Experimental Setup We read information from given sample (See *ToyData.xlsx* for more details) into some text files and set parameters according to known information. The bandwidths visualization is as follows (See figure 3):

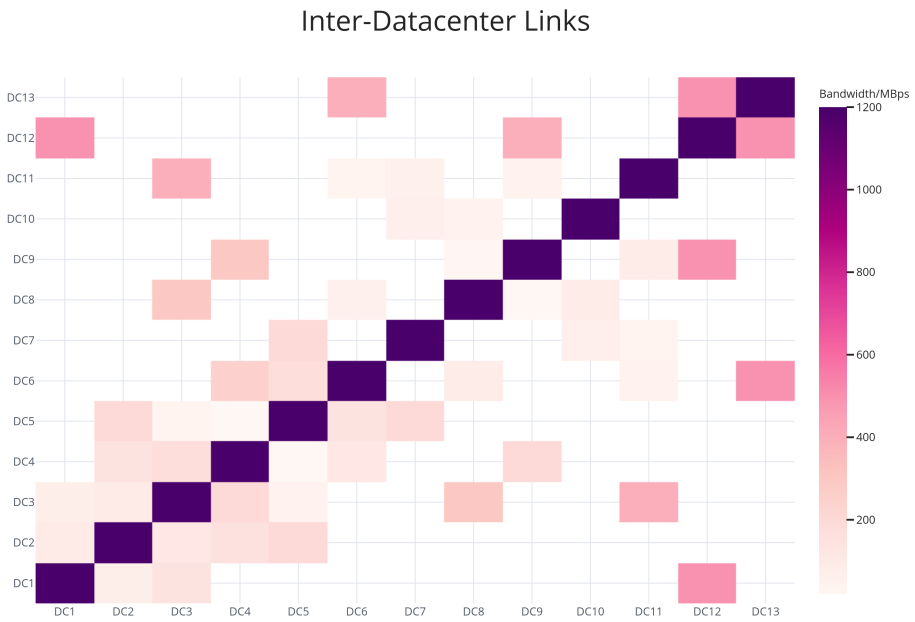

Fig. 3. Bandwidths Visualization of *ToyData.xlsx*

Experimental Results After running the code, we have the results about start-time, finish-time and allocating decision of every task (See figure 4).

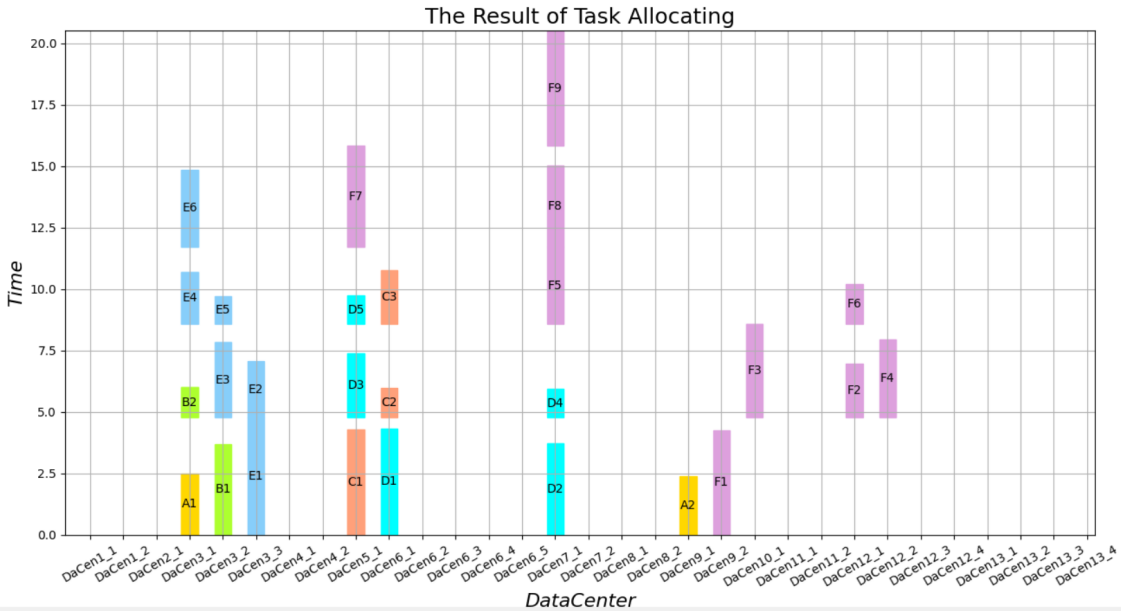

Fig. 4. Allocation Result of *ToyData.xlsx*

In this figure, we could witness assignments of every task in each job clearly. Every task is represented by a columnar rectangle with label to show its name, the bottom and top of a rectangle indicate the start-time and finish-time of the task respectively. For the axis, we list every slot in each datacenter, and the location of a rectangle indicates where it should be allocated.

After calculation, we can conclude the respective executing time of each job $k \in \{A, B, C, D, E, F\}$ are 2.5000, 6.0144, 10.7644, 9.7311, 14.8573 and 20.5229. Meanwhile, the average executing time of these jobs is 10.7317 and the variance of execution time is 40.5403. Since the number of tasks of these jobs are different, the max-min fairness cannot be accurately reflected. However, our algorithm gets a excellent result.

5.2 Implementation of Real-word Sample

Searching Information Online We find a real-word sample by searching online, a geographically denser set of DCs on EC2, Google Cloud [7]. It has plenty of virtual machines located in 11 geographically distributed cities over the world. The bandwidths between them are shown as figure 5.

Compared to the intradatacenter network where the available bandwidth is around 1 Gbps, bandwidth on inter-datacenter links are much more limited: almost all inter-continental links have less than 70 Mbps of available bandwidth. This confirms the observation that transferring large volumes of data across datacenters is likely to be time-consuming.

For no slots and jobs list given, we create an objective list to implement on this sample.

Suitability for Testing For its origin source of it, the data is reliable and credible. The bandwidths of them are more similar to the real-world condition. From the information we get, the datacenters are geo-distributed and easy to be transform to *DAG*. Thus, it can be applied to our research, and it's suitable for testing.

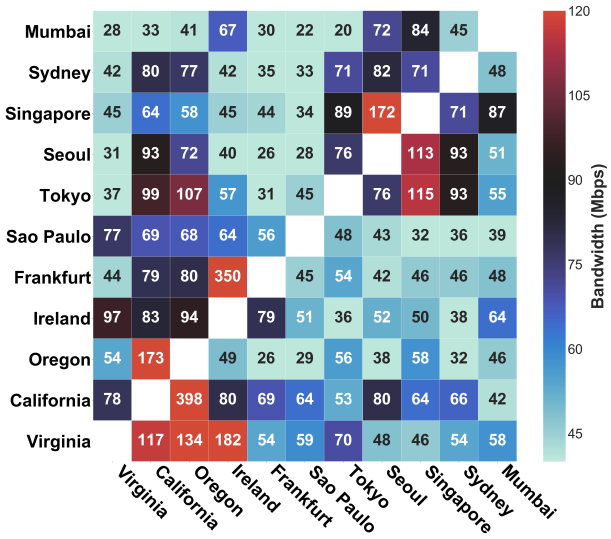

Fig. 5. Bandwidths of *DCs* on *EC2*, Google Cloud

Experimental Results After running the code, we have the results about start-time, finish-time and allocating decision of every task (See figure 6).

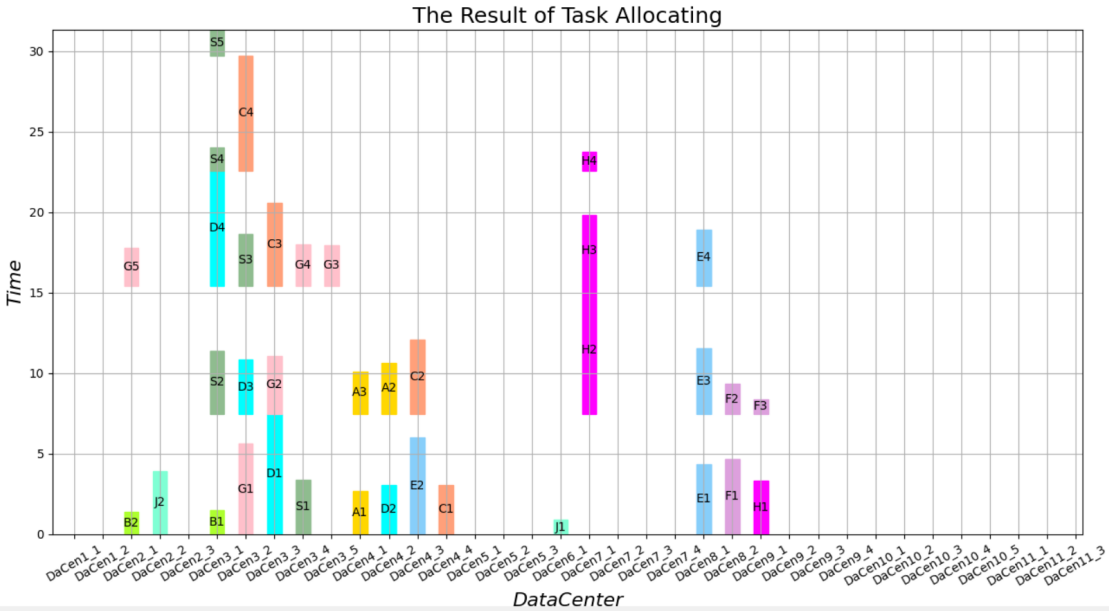

Fig. 6. Allocation Result of *DCs* on *EC2*, Google Cloud

The visualization is defined similar to the previous one.

Acknowledgements and Concluding Remarks

Summary

In this paper, we have conducted a theoretical study of the tasks scheduling problem with max-min fairness among competing data analytic jobs, whose input data are distributed across geo-distributed datacenters. With tasks from multiple jobs competing for the computing slots in each datacenter.

Max-Min Fairness To achieve the objective of max-min fairness, we first formulated a lexicographical minimization problem to optimize all the job completion times, then we transformed it into an equivalent linear programming (LP) problem. we designed and implemented a algorithm to assign tasks across these datacenters with max-min fairness achieved. Last but not the least, we have implemented our idea through some examples, Witnessed the rationality of our algorithm.

Scalability Furthermore, as in our real-world experiment part, we have measured the running times with different problem scale created by ourselves in our simulations to evaluate the scalability of our algorithm. And the result is also good.

Acknowledgements

After searching a lot of information and having enough understanding of the problem, we try to apply amount of mathematical methodologies on this problem. Finally, we choose the LP to satisfy the goal of minimizing the worst completion time and max-min fairness. More significantly, we apply *lexicographical minimization*, *convex optimization* to transform our constraint conditions to LP problem, and we use *unimodular matrix*, λ*-representation* to simplify the process of solving LP problem.

We believe that there exist many other more efficient and superexcellent algorithms as well as methods arisen by other outstanding people to solve this problem. This problem is applying among many fields not only in computer science, but also in others such as Network Heterogeneity and Economic-Based Resource Allocation. Thus, keep thinking of solving this problem is meaningful and worthwhile.

We gain a lot through the whole experimental process. We have gotten known with a lot of useful mathematical knowledge which cannot be received from offline class teaching. Coming up with a idea to solve a problem together, discussing with the merit and demerit of an algorithm, and realizing our idea are very exciting. We also improve our coding skill and ability to write papers.

Thanks to the instructors, *Xiaofeng Gao* and *Lei Wang*, for giving the lectures and teaching materials in this semester, making us learned a lot about algorithm, which is very helpful for our study in computer science.

References

1. Ivan Marsic.: Computer Networks: Performance and Quality of Service, ch. 5, p. 309–366. Rutgers University, New Brunswick, New Jersey, Location (2013)
2. Wikiwand, https://www.wikiwand.com/en/Convex_optimization. Last accessed 28 May 2021
3. B.Korte, J.Vygen.: Combinatorial Optimization: Theory and Algorithms. 3rd edn, p. 104. ser. Algorithms and Combinatorics. Location (2000)
4. R. Meyer.: "A Class of Nonlinear Integer Programs Solvable by A Single Linear Program". SIAM Journal on Control and Optimization, vol. 15, no. 6, pp. 935–946. Springer, Heidelberg (1997).
5. Youtube Video at Colorado State University, https://www.youtube.com/watch?v=3BPHGeWuAyA&ab_channel=HenryAdams. Last accessed 28 May 2021
6. T.C.E. Cheng, Q. Ding.: The time dependent machine makespan problem is strongly NP-complete. (1999)
7. USENIX Homepage, https://www.usenix.org/. Last accessed 28 May 2021

Appendix

Proof of Lemma 1

We first consider $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{Z}^K$ that satisfies $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$. If we use the integer $\tilde{k}(1 \leqslant \tilde{k} \leqslant K)$ to represent the first non-zero element of $\langle \boldsymbol{\alpha} \rangle - \langle \boldsymbol{\beta} \rangle$, we have $\langle \boldsymbol{\alpha} \rangle_k = \langle \boldsymbol{\beta} \rangle_k, \forall \tilde{k} \leqslant k \leqslant K$ and $\langle \boldsymbol{\alpha} \rangle_{\tilde{k}} < \langle \boldsymbol{\beta} \rangle_{\tilde{k}}$. Assume $\langle \boldsymbol{\alpha} \rangle_{\tilde{k}} = m$, then $\langle \boldsymbol{\beta} \rangle_{\tilde{k}} \geqslant m + 1$.

$$\varphi(\boldsymbol{\alpha}) = \sum_{k=1}^{K} K^{\langle \boldsymbol{\alpha} \rangle_k} + K^{\langle \boldsymbol{\alpha} \rangle_{\tilde{k}}} + \sum_{k=\tilde{k}+1}^{K} K^{\langle \boldsymbol{\alpha} \rangle_k}$$

$$\leqslant \sum_{k=1}^{\tilde{k}-1} K^{\langle \boldsymbol{\alpha} \rangle_k} + K^{\langle \boldsymbol{\alpha} \rangle_{\tilde{k}}} + (K = \tilde{k}) K^{\langle \boldsymbol{\alpha} \rangle_{\tilde{k}}}$$

$$= \sum_{k=1}^{\tilde{k}-1} K^{\langle \boldsymbol{\alpha} \rangle_k} + (K + 1 - \tilde{k}) \cdot K^{\langle \boldsymbol{\alpha} \rangle_{\tilde{k}}}$$

$$< \sum_{k=1}^{\tilde{k}-1} K^{\langle \boldsymbol{\alpha} \rangle_k} + K \cdot K^m,$$

where the first inequality holds as $\langle \boldsymbol{\alpha} \rangle_{\tilde{k}} \geqslant \langle \boldsymbol{\alpha} \rangle_k, \forall \tilde{k} + 1 \leqslant k \leqslant K$.

$$\varphi(\boldsymbol{\beta}) = \sum_{k=1}^{K} K^{\langle \boldsymbol{\beta} \rangle_k} + K^{\langle \boldsymbol{\beta} \rangle_{\tilde{k}}} + \sum_{k=\tilde{k}+1}^{K} K^{\langle \boldsymbol{\beta} \rangle_k}$$

$$> \sum_{k=1}^{\tilde{k}-1} K^{\langle \boldsymbol{\beta} \rangle_k} + K^{\langle \boldsymbol{\beta} \rangle_{\tilde{k}}} + (K - \tilde{k}) \cdot 0$$

$$\geqslant \sum_{k=1}^{\tilde{k}-1} K^{\langle \boldsymbol{\beta} \rangle_k} + K \cdot K^m.$$

Given that $\sum_{k=1}^{\tilde{k}-1} K^{\langle \boldsymbol{\alpha} \rangle_k} = \sum_{k=1}^{\tilde{k}-1} K^{\langle \boldsymbol{\beta} \rangle_k}$, we have proved that $\varphi(\boldsymbol{\alpha}) < \varphi(\boldsymbol{\beta})$.

If $\boldsymbol{\alpha} = \boldsymbol{\beta}$, which means that $\langle \boldsymbol{\alpha} \rangle_k = \langle \boldsymbol{\beta} \rangle_k, \forall 1 \leqslant k \leqslant K$, it is trivially true that $\varphi(\boldsymbol{\alpha}) = \sum_{k=1}^{K} K^{\langle \boldsymbol{\alpha} \rangle_k} = \sum_{k=1}^{K} K^{\langle \boldsymbol{\beta} \rangle_k} = \varphi(\boldsymbol{\beta})$. Thus, we have proved $\boldsymbol{\alpha} \preceq \boldsymbol{\beta} \Rightarrow \varphi(\boldsymbol{\alpha}) \leqslant \varphi(\boldsymbol{\beta})$.

We further prove $\varphi(\boldsymbol{\alpha}) \leqslant \varphi(\boldsymbol{\beta}) \Rightarrow \boldsymbol{\alpha} \preceq \boldsymbol{\beta}$ by proving its contrapositive: $\neg(\boldsymbol{\alpha} \preceq \boldsymbol{\beta}) \Rightarrow \varphi(\boldsymbol{\alpha}) > \varphi(\boldsymbol{\beta})$. $\neg(\boldsymbol{\alpha} \preceq \boldsymbol{\beta})$ implies $\boldsymbol{\alpha} \neq \boldsymbol{\beta}$ and the first non-zero element of $\langle \boldsymbol{\alpha} \rangle - \langle \boldsymbol{\beta} \rangle$ is positive, which further indicates the first non-zero element of $\langle \boldsymbol{\beta} \rangle - \langle \boldsymbol{\alpha} \rangle$ is negative, *i.e.*, $\boldsymbol{\beta} \prec \boldsymbol{\alpha}$. Thus, the contrapositive is equivalent to $\boldsymbol{\beta} \prec \boldsymbol{\alpha} \Rightarrow \varphi(\boldsymbol{\beta}) \leqslant \varphi(\boldsymbol{\alpha})$, which has already been proved previously using the exchanged notations of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

With $\boldsymbol{\alpha} \preceq \boldsymbol{\beta} \Longleftrightarrow \varphi(\boldsymbol{\alpha}) \leqslant \varphi(\boldsymbol{\beta})$ holding for any $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of the same dimension, we complete the proof by letting $\boldsymbol{\alpha} = \boldsymbol{g}(\boldsymbol{x}^*)$ and $\boldsymbol{\beta} = \boldsymbol{g}(\boldsymbol{x})$.

Proof of Lemma 2

Let $\mathsf{A}_{m \times n}$ denote the coefficient matrix of all the linear constraints (8) and (9), where $m = J + \sum_{k=1}^{K} n_k$, representing the total number of the constraints, and $n = J \sum_{k=1}^{K} n_k$, denoting the dimension of the variable $\boldsymbol{x}$.

It is obvious that any element of $\mathsf{A}_{m \times n}$ is either 0 or 1, satisfying the first condition. For any row subset $R \subset \{1, 2, \cdots, m\}$ we can select all the elements that belong to $\{1, 2, \cdots, J\}$ to form the set R_1. As such, R is divided into two disjoint sets, R_1 and $R_2 = R - R_1$. It is easy to check that for coefficient matrix of constraint (8), the summation of all its rows, represented by rows $\{1, 2, \cdots, J\}$, is a $1 \times n$ vector with all the elements equal to 1. Similarly, for coefficient matrix of constraint (9), the summation of all its rows, represented by rows $\{J + 1, J + 2, \cdots, J + \sum_{k=1}^{K} n_k\}$, is also a $1 \times n$ vector whose elements are 1. Hence, we can easily derive that $\sum_{i \in R_1} a_{ij} \leqslant 1, \sum_{i \in R_2} a_{ij} \leqslant 1, \forall j \in \{1, 2, \cdots, n\}$. Eventually, we have $|\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij}| \leqslant 1, \forall j \in \{1, 2, \cdots, n\}$, and the second condition is satisfied.

In summary, we have shown that both conditions for total unimodularity are satisfied, thus the coefficient matrix $\mathsf{A}_{m \times n}$ is totally unimodular.

Summary of Symbols

Below are the symbols we defined in the process of mathematical derivation.

Table 1. Summary of Symbols

Symbol	The meaning of symbol
$\mathcal{K}$	the set of jobs
K	number of jobs
$\mathcal{D}$	the set of datacenters
J	number of datacenters
$\mathcal{T}_k$	the set of tasks for job k
n_k	number of tasks for job k
a_j	the capacity of available computing slots in datacenter j
$c_{i,j}^k$	the network transfer time of task i
$e_{i,j}^k$	the execution time of task i
S_i^k	the set of datacenters where the input data of task i are stored
$d_i^{k,s}$	needed data amount of task i from datacenter j
$b_{s,j}$	the bandwidth of the link from datacenter s to j
τ_k	the completion time of job k
$x_{i,j}^k$	whether task i is assigned to datacenter j
ST_i^k	the start-time of task i
FT_i^k	the finish-time of task i
G	Stage
g	sub-Stage