Stage-Based Strategy for Scheduling Jobs Problem with Max-Min Fairness

Project for Algorithm and Complexity

Taoran Han (519021910911, hantaoran@sjtu.edu.cn), Longxuan Wei (519021910789, weilongxuan189034@sjtu.edu.cn), Log Creative (, logcreative-lzl@sjtu.edu.cn)

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

Abstract. This project introduces a solution of Scheduling Jobs Problem that is based on different stages of a single job. Moreover, it formalizes the max-min fairness in this problem with an approximation algorithm. It also proves that this problem is NP-Complete and analyzes this problem's complexity. Finally, it make some tests based on Toy Data and Real World Data.

Keywords: Geo-distributed Data Center Networks, Scheduling Algorithm, Max-Min Fairness, NP-Completeness.

1 Introduction

With the rapid growth on the scale of data clusters, there are more and more data online in different Geo-Distributed Data Centers. And the amount of data is so big that it is a emerging demand to schedule data within the bandwidth across data centers and take advantage of data locality at the same time. Moreover, the workload of computational resources requires to be distributed as fair as possible, in order to balance the cost in different areas.

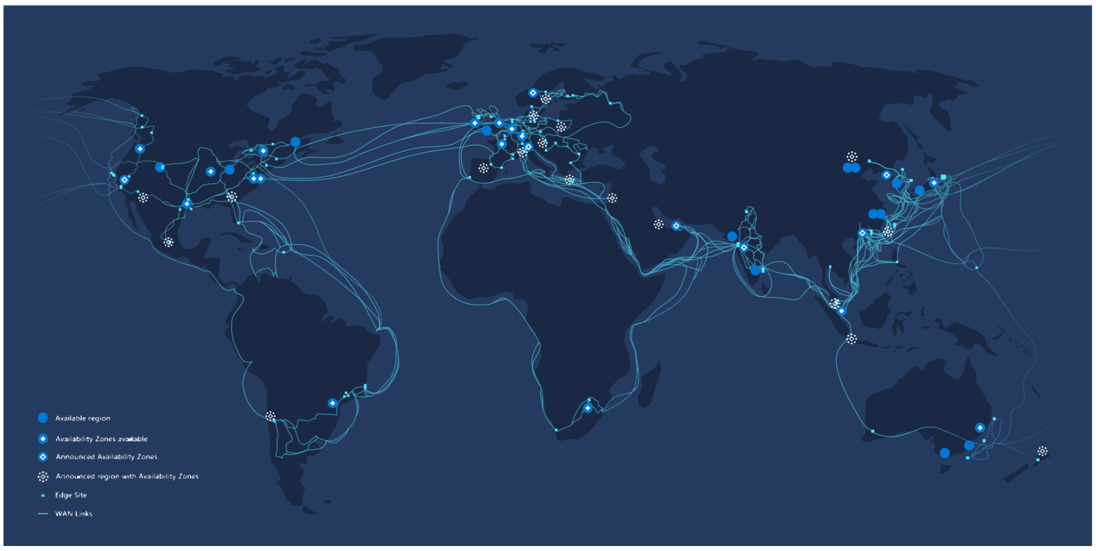
Available region
Availability Zones available
Announced Availability Zones
Announced region with Availability Zones
Edge Site
WAN Links

Fig. 1. Data centers and its connection in Azure global network[1]

2 Notation

In Scheduling Jobs across Geo-Distributed Data Centers, we have a set of jobs and a set of data centers. For each job, there are several tasks. Each task may require data from different centers, and it may also require some data from other tasks of the same job. For each data center, there are some slots to place tasks.

Our assignment is to assign all tasks into slots with the minimum average completion time of these jobs. And we need to make some notations first.

Obviously, we can first denote that there are K jobs and m data centers in total. For job i, the number of its tasks is t_i. For center i, the number of its slots is s_i. Then we denote the execution time of task i of job k as $e_{k,i}$. The bandwidth from center i to center j is denoted as $b_{i,j}$. Moreover, $d^j_{k,i}$ denotes the amount of data that task i of job k requires from data center j.

Furthermore, for every job i, there is a graph G that shows its precedence constraints. And we can use a square matrix T^i to represent it. For example, if task i_2 requires data of size 100 from task i_1, then the element $T^i_{1,2}$ will become 100. If there are no edges between task i_3 and i_4, then the element $T^i_{3,4}$ will become 0.

Here is a more specific example. If there are 3 tasks in job A, which we denote as t_1, t_2 and t_3, and t_2 needs 100MB data from t_1, t_3 needs 200MB data from t_2, t_3 needs 500MB data from t_1. Then the corresponding matrix will be:

$$\begin{pmatrix} 0 & 100 & 500 \\ 0 & 0 & 200 \\ 0 & 0 & 0 \end{pmatrix}$$

Now we have defined the corresponding notations of all the information we know. Finally we will define the variables that we want to optimize.

For every task i of job k, we create m binary variables $x^1_{k,i}, \cdots, x^m_{k,i}$. If $x^j_{k,i} = 1$, it means task i of job k is allocated to center j.

Up to now, we've denoted all the notations we will use in our simulation. Then we need to add constrains to them.

All the variables we denoted are shown in Table 1.

Table 1. Symbel Table of Variables for Scheduling Jobs Problem

Variable	Meaning
K	The number of jobs
m	The number of data centers
t_k	The number of tasks of job k
s_i	The number of slots of center i
$e_{k,i}$	The execution time of task i of job k
$b_{i,j}$	The bandwidth from center i to center j
$d^j_{k,i}$	The amount of data that task i of job k requires from center j
$T^k_{i,j}$	The amount of data that task j of job k requires from task i of job k
q_k	The number of stages of job$k$
p^k_i	The number of tasks of stage i of job k
$x^j_{k,i}$	Whether task i of job k is allocated to center j
$c_{k,i}$	The completion time of task i of job k
C_k	The completion time of job k
ACT	The average completion time of all jobs

3 Formalization

3.1 Formalization and Test of a Single Example without Max-Min Fairness

To begin with, we want to simulate a simple example without max min fairness by using Cplex. This example is exactly same as the one introduced in Project-Data-AnalyticJobScheduling.pdf.

In this example, we have $K = 2$ and $m = 3$. There are 2 parallel tasks for job A and job B each and the execution time for every task is one second. The vector for the number of slots is $\begin{pmatrix} 2 & 2 & 1 \end{pmatrix}$.
The matrix for bandwidth is:

$$\begin{pmatrix} 1000 & 80 & 150 \\ 80 & 1000 & 120 \\ 100 & 160 & 1000 \end{pmatrix}$$

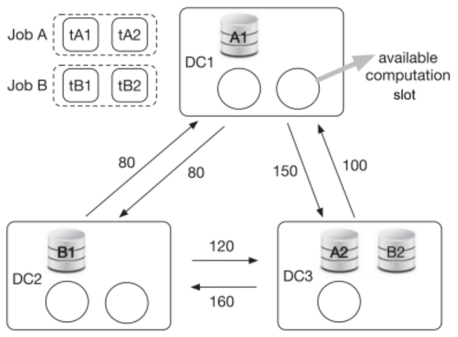

Fig. 2. An example of scheduling multiple jobs across geo-distributed data centers[2]

Moreover, for every job, there is a request matrix.

$$D_1 = \begin{pmatrix} 100\ 0\ 200 \\ 100\ 0\ 200 \end{pmatrix} D_2 = \begin{pmatrix} 0\ 200\ 200 \\ 0\ 200\ 300 \end{pmatrix}$$

For every job, there is also another matrix with binary elements which is corresponding to $x^j_{k,i}$. Their values need to be determined. Then we need to add constraints.
First, for every task i of job k, it can only be placed in one single center, which means that

$$\sum_{j=1}^{m} x^j_{k,i} = 1, \forall i \le t_i, \forall k \le K \tag{1}$$

Second, for every data center, the number of tasks there could not exceed the number of its slots. Therefore, we have

$$\sum_{k=1}^{K} \sum_{i=1}^{t_k} x^j_{k,i} \le s_j, \forall j \le m \tag{2}$$

With these constraints, we need to calculate the completion time for each job. Now let's first consider the completion time of a single task. This task may need data from several different centers, and different centers can transfer data to this job concurrently. Therefore, the completion of it will be:

$$c_{k,i} = \max_{1 \le j_2 \le m} \sum_{j_1=1}^{m} \frac{x^{j_1}_{k,i} \times d^{j_2}_{k,i}}{b_{j_2,j_1}} + e_{k,i} \tag{3}$$

Because every task of a job runs concurrently, we just need to calculate maximal completion time of tasks for every job, which means that $C_k = \max(c_{k,1}, \cdots, c_{k,t_k})$. And the average completion time

$$ACT = \frac{\sum_{k=1}^{K} C_k}{K} \tag{4}$$

is what we want to minimize.

Now we have analyzed all the data and constraints of this simple example, then we add data into the file example.dat and add constraints into the file example.mod. We can build a model and simulate this simple example. Here is the result we get:

3.2 Formalization for More Complex Scheduling Jobs Problem

In the single example, we have made part of formalization of the whole problem. However, there are some constraints that the single example doesn't have.
There are only 2 jobs with 2 tasks each in the single example. Moreover, for each job, all of its tasks can run concurrently. But in fact, there may be some precedence constraints inside a job. In part 2 we have defined a matrix to show these precedence constraints. Here we will concentrate on how to use these

```
averagetime = 2.8333;
JA = [[1 0 0]
             [1 0 0]];
JB = [[0 1 0]
             [0 0 1]];
timeA = 3;
timeB = 2.6667;
```

Fig. 3. Result from simulation of a simple example

constraints to calculate corresponding task's completion time.
If there exist any precedence constraints inside a job, then we can use a graph G to represent them. For example, if a job has the following graph:

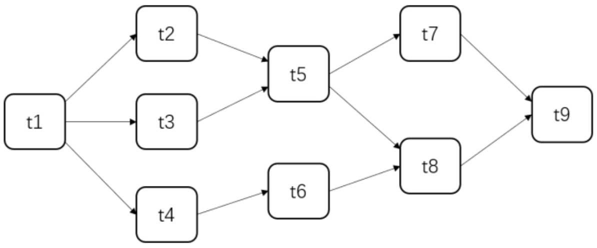

Fig. 4. An example of a job with precedence constraints

It means that this job has several stages. More importantly, only when an old stage is finished may a new stage start to be computed. In this case, stage 1 is $\{t1\}$, stage 2 is $\{t2, t3, t4\}$, stage 3 is $\{t5, t6\}$, stage 4 is $\{t7, t8\}$ and stage 5 is $\{t9\}$.
The strategy that we get stages from a graph is: At the beginning, every node is not visited. Then every time we choose the nodes that satisfy the following two requirements to form a new stage. First, these nodes have not been visited. Second, either they don't have parent node or all of their parent nodes have been visited.
The completion time of this job, obviously, is the summation of the completion times of its stages. If we define that for job k, there are q_k stages in total and the size of each stage s is p_s^k, then we have

$$C_k = \max(c_{k,1}, \cdots, c_{k,p_1^k}) + \max(c_{k,p_1^k+1}, \cdots, c_{k,p_1^k+p_2^k}) + \cdots + \max(c_{k,t_k-p_{q_k}^k+1}, \cdots, c_{k,t_k}) \tag{5}$$

Furthermore, in the simple example, for task i of job k, its completion time is the maximal transmission time from a data center to this task plus its execution time. However, after we add precedence constraints, we need to consider the transmission between two tasks and choose the maximal one. So the completion time becomes

$$c_{k,i} = \max\Big(\max_{1\le j_2\le m} \sum_{j_1=1}^{m} \frac{x_{k,i}^{j_1} \times d_{k,i}^{j_2}}{b_{j_2,j_1}} + e_{k,i}, \max_{1\le j_3\le t_k} \sum_{j_1=1}^{m}\sum_{j_2=1}^{m} \frac{x_{k,i}^{j_1} \times x_{k,j_3}^{j_2} \times T_{j_3,i}^{k}}{b_{j_2,j_1}} + e_{k,i}\Big) \tag{6}$$

where $T_{j_3,i}^k$ is the requested data amount from task j_3 to task i in job k.
Now there is still a factor that we haven't taken into consideration. In the simple example, there exists a bandwidth between every two data centers. However, in more complex problems, data may be not able to be transferred between some centers directly. In this case, data need to be transferred indirectly. For example, if the bandwidth between center A and center B is 100MB/s and the bandwidth between center B and center C is 100MB/s and there is no bandwidth between center A and center C. Then if we want to transform data between A and C, we need to transform it indirectly. First, we transform the data from A to B, then we transform the data from B to C. Moreover, we can calculate the equivalent bandwidth between A and C is 50MB/s.

Using the same strategy, we can calculate the corresponding bandwidth between every two data centers. Considering that the data transferring path need to be optimal, we will use the Floyd-Warshall's Algorithm. The calculating process is: First we calculate every bandwidth's reciprocal, which means the time to transfer 1MB of data through this bandwidth. Then we use Floyd-Warshall Algorithm to the whole matrix. By this way, we can get the optimal time, which is corresponding with optimal path. At last, we turn times into bandwidths by calculating reciprocal again. And we will get a new bandwidth matrix, in which there is also a bandwidth between every two data centers that can transfer data indirectly.

This assumption could be justified by the fact that we could construct a transfer task without the need for any computing resources to pass the data from the previous transfer. Because data could transfer on every link, the data could also been transferred indirectly in this way.

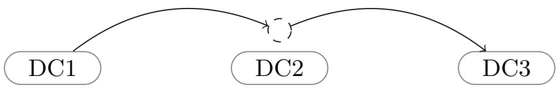

Fig. 5. Indirect transfer through a *transfer* task

3.3 Formalization for Scheduling Jobs with Max-Min Fairness

Finally, we will add description of max min fairness of this problem.

In order to minimize its completion time, every job may require different number of slots from different centers. However, the number of slots for every center is limited. Thus we need max min fairness to determine the allocation.

For each center, if the number of slots that is required from different jobs exceeds the number of slots it has, then we use max-min fairness.

To satisfy max-min on computing resources, we can first omit the constraints of slots. In this case we will get the number of requests, then we allocate limited slots according to max-min fairness. We will re-allocate the data center that has more requests than demand. For those data centers that has more demand than resources, we will fix the existing allocation in place for the next iteration. In the next iteration, the number of slots in those exceeded data centers will be limited, and apply the minimum average time on the current configuration until all limits on slots are satisfied and we get the optimal solution with minimum average time while maintaining max-min fairness. Figure 7 shows this process intuitively.

Now let's show that by using this strategy, we can reach an approximate max min fairness. For each iteration, we can view the overflowing centers as some small-size glasses with overflowing beer. And we can view the centers with empty slots as some large-size glasses with unfulfilled beer. Then we limit the number of slots in those overflowing centers, run the algorithm again and we will get another allocation. This process is similar to pour that overflowing beer into those glasses with capacity.

How we define a center's capacity of resources is necessary to emphasize here. It doesn't completely depend on the number of its slots. Because for those jobs, one slot may be not so popular as another. Those slots with higher popularity have lower capacity of resources oppositely. And those slots with lower popularity have higher capacity of resources. Therefore, those overflowing centers have more popular slots, correspondingly, they have lower capacity of resources. And we can regard them as glasses with smaller size.

Every time we re-allocate overflowing centers with the constraints of slots, then we lock their allocation, which means that we satisfy glasses of smaller size. We do this iteration again and again until there are no overflowing centers. For each iteration, we still maintain the claim of minimal average completion time, thus the final result with approximate max min fairness will not be too bad.

4 NP-Complete

This problem is a NP-Complete problem.[3]

The problem is to schedule all the tasks in the DAG to a number of worker nodes — while minimizing the completion time of the job. Given a set S of all n tasks which can be put into different DAG's, a time limit k, a relation $<$ on S which can be determined from the DAG's directed edges and a sequence of integers $c_0, c_1, \ldots, c_{t-1}$ with $\sum_{i=0}^{t-1} c_i = n$. Our goal is to find a function f from S to $\{0, 1, \ldots, t-1\}$ to make sure that $f^{-1}(i)$ has exactly c_i members and make the average cost be minimum.

Definition 1 (3-Satisfiability — 3-SAT). *Given a set of $\{x_i\}$, $1 \leq i \leq m$, and a collection of sets $D_1, \ldots, D_n$, where $m \leq 3n$, such that each D_i consists of exactly three of the elements x_i or $\overline{x}_i$.*

Given an instance of 3-SAT, we can construct the following instance of this problem. The tasks we shall denote:
x_{ij} and $\overline{x}_{ij}$ for $1 \leq i \leq m$ and $0 \leq j \leq m$;
y_i and $\overline{y}_i$ for $1 \leq i \leq m$;
D_{ij} for $1 \leq i \leq n$ and $1 \leq j \leq n$;
Thre relation ¡ is given by:

(i) $x_{ij} < \overline{x}_{i,j+1}$ and $\overline{x}_{ij} < \overline{x}_{i,j+1}$, for $1 \leq i \leq m$ and $0 \leq j < m$;
(ii) $x_{i,i-1} < y_i$ and $\overline{x}_{i,i-1} < \overline{y}_i$ for $1 \leq i \leq m$;
(iii) Considering D_{ij}, where $a_1a_2a_3$ is the binary representation of j.(Note that the case $a_1 = a_2 = a_3 = 0$ can't occur.) Let D_i consist of literals $z_{k_1}, z_{k_2}, z_{k_3}$, where each z independently stands for x or $\overline{x}$, in a fixed order. then for $1 \leq p \leq 3$, if $a_p = 1$, we have $\overline{z}_{k_p,m} < D_{ij}$, where $\overline{z}$ stands for $\overline{x}$ or x.

The limit of k is $m+3$, and constants c_i, $0 \leq i \leq m+2$ are:

$$\begin{aligned} c_0 &= m \\ c_1 &= 2m+1 \\ c_i &= 2m+2 (i \leq i \leq m) \\ c_{m+1} &= n+m+1 \\ c_{m+2} &= 6n \end{aligned}$$

We will show that the above instance has a solution if and only if the given instance of 3-SAT does. We imagine that x_i or $\overline{x}_i$ to be true if and only if x_{i0} or $\overline{x}_{i0}$ is executed at time 0. We know that the presence of the y's and $\overline{y}$'s forces exactly one of x_{i0} and $\overline{x}_{i0}$ to be executed at time 1. Then the requirement that $n+m+1$ jobs be executed at time $m+1$ is tantamount to the requirement that for each i, there is one j such that D_{ij} may be executed at that time. But this condition is equivalent to saying that the sum of items which D_i represents has value true when those of the x_i's and $\overline{x}_i$'s which were executed at time 0 are given the value true.

In any solution to the instance, we may not execute both x_{i0} and $\overline{x}_{i0}$ at time 0 for any i. Suppose we did, then since $c_0 = m$, there would be some j such that neither x_{i0} nor $\overline{x}_{i0}$ was executed at time 0. Then neither y_i nor $\overline{y}_i$ would be executed at or before time j, for example, y_i must be preceded by $x_{j0}, \ldots, x_{j,j-1}$, each executed strictly before the next in the sequence. The total number of jobs which could be executed at or before time j is thus seen to be:

(1) at most $m(2j+1)$ of the x's and $\overline{x}$'s ,that is $z_{0i}, \ldots, z_{ij}$ if z_{i0} was executed at time 0 and $z_{i0}, \ldots, z_{i,j-1}$
(2) at most $2(j-1)$ of the y's, specially $y_1, \overline{y}_1, \ldots, \overline{y}_{j-1}$

We may conclude that in any solution to this instance, exactly one of x_{i0} and $\overline{x}_{i0}$ is executed at time 0. Moreover, we can determine the exact jobs which are executed at each time, given which of x_{i0} and $\overline{x}_{i0}$ is executed at time 0. At time t we must execute z_{it} if z_{i0} was executed at time 0 and $z_{i,t-1}$ if not. Moreover, we must execute y_t at time t if x_{t0} was executed at time 0 and execute y_{t-1} at time t if x_{t0} was executed at time 1.

At time m + 1 we can execute the m remaining x's and $\overline{x}$'s and the one remaining y or $\overline{y}$. Since $c_{m+1} = m+n+1$, we must be able to execute n of the D's if we are to have a solution. We observe that for each pair $D_{i,j}$ and $D_{i,j'}$, $j \neq j'$, there is at least one k such that x_{km} precedes D_{ij} and $\overline{x}_{km}$ precedes $D_{ij'}$, or vice versa. Since we have already proven that exactly one of x_{km} and $\overline{x}_{km}$, can be executed by time m, it follows that for each i, at most one of $D_{i1}, \ldots, D_{i7}$ can be executed at time m + 1.

Moreover, if we assign the truth value true to x_k if and only if x_{k0} was executed at time 0, then there will be one of $D_{i1}, \ldots, D_{i7}$ executable at time m + 1 if and only if D_i takes the value true under this assignment of values to the variables. We conclude that a solution to the instance exists if and only if the original product of sums is satisfiable.[4]

$$\text{3-SAT} \leq_P \text{DAG JOB SCHEDULING}$$

where 3-SAT is a NP-Complete problem, so is DAG JOB SCHEDULING.

5 Algorithm & Complexity

The Floyd-Warshall's Algorithm on the bandwidth matrix costs $O(m^3)$, where m is the number of data centers.

For any arbitrary job DAG, to get the stage level of each task, we conduct the following algorithm:

Algorithm 1: DAG Stage Separation

```
Input: Request Matrix $D$
Output: Stage Separation for every job
1 Reduce $D$ to $D'$ with task items only;
2 Turn $D'$ into adjacency list $G$ on positive element;
3 Get topological order of $G$ by recording to PRE of Depth-First Search;
4 Run Breadth-First Search on $G$ in topological order to get the stage level, updating the level in a
    top-down method;
5 Group task by (job, level);
```

The algorithm is efficient on parse graph with the use of adjacency list, with the total running time of $O(|V|^2 + |E|)$. In this case, $|E| = \Theta(|V|)$, thus $O(|V|^2)$ is the time complexity in this stage, where $|V| = \#tasks = \sum_{k=1}^{K} \sum_{i=1}^{t_k} 1$ is the number of all tasks.

With the stages of each job known, it is time to write down the constraints on elapsed time on each stage. It is discovered that using built-in function `max` in CPLEX is slow. We turned the max function into inequalities.

Assume that $t_{k,i}$ is denoted as the transmission time from a data center for task i in job j and $t'_{k,i}$ is the transmission time between two tasks for the same task. Equation (6) could be turned into

$$c_{k,i} \geq t_{k,i} + e_{k,i} \tag{7}$$

$$c_{k,i} \geq t'_{k,i} + e_{k,i} \tag{8}$$

$$t_{k,i} \geq \sum_{j_1=1}^{m} \frac{x_{k,i}^{j_1} \times d_{k,i}^{j_2}}{b_{j_2,j_1}}, \qquad \forall 1 \leq j_2 \leq m \tag{9}$$

$$t'_{k,i} \geq \sum_{j_1=1}^{m} \sum_{j_2=1}^{m} \frac{x_{k,i}^{j_1} \times x_{k,j_3}^{j_2} \times T_{j_3,i}^{k}}{b_{j_2,j_1}} + e_{k,i}, \qquad \forall 1 \leq j_3 \leq t_k \tag{10}$$

It is obvious that to optimze the average cost, $c_{k,i}$ is required to touch the tightest bound in Constraints (7) and (8), which satisfies the original equality of (6).

Assume that $c'_{k,s}$ is the cost of stage s in job k, then it is the longest cost among all tasks in this stage. And Equation (5) is turned into

$$C_k = \sum_{s=1}^{q_k} c'_{k,s} \tag{11}$$

$$c'_{k,s} \geq c_{k,i}, \quad \forall i : 1 + \sum_{l=1}^{s-1} p_l^k \leq i \leq \sum_{l=1}^{s} p_l^k \tag{12}$$

since the task in every stage could be non-continuous, it is just a representation to show that it is the maximum elapsed time in every stage.

To perform the method mentioned in Section 3.3, we conduct the iteration algorithm as follows:

The iteration time will not exceed m, thus the optimization costs $O(mT)$, where T is the computing cost on Linear Optimization using Gurobi (often 3s).

As a result, the total time complexity of our algorithm is:

$$O(m^3) + O(n) + O(mT)$$

where $n = \#tasks$. Our algorithm performs 10s on Toy Data.

Algorithm 2: Iterative Max-Min Fairness Algorithm

```
1  Initialize Optimize Model with basic constraints (1), (7 – 10), (11 – 12);
2  ConstrSlot ← [];
3  repeat
4      foreach slot do
5          if slot∈ConstrSlot then
6              add constraint on mono slot similar to Constraint (2);
7          else
8              fix the allocated task on slot.
9          end
10         Optimize Model;
11         append the overflow slots to ConstrSlot;
12     end
13 until original Constraint (2) is satisfied;
```

6 Test on Toy Data

We firstly test the algorithm by using Cplex. However, because we use too many "max" functions, there is a very large gap between the result and the objective. Then we improve our algorithm and test it by using gurobi. And we get nice results.

The result without max-min fairness is shown in Figure 6, where the left part shows the allocation of tasks and the right part shows the connection between the nodes (red indicates a longer time on transferring). And the result with the iteration process with the max-min allocation on computing resources is shown in Figure 7, where the upper part is the iteration process and the bottom part shows the change in average time (red line is the time without max-min fairness).

7 Test on Real World Data

Real world should satisfy that the total number of slots is larger than or equal to the total number of tasks, which means that $\sum_{i=1}^{m} s_i \geq \sum_{i=1}^{K} t_i$. In order to meet the requirement of this problem, we generate a certain amount of data sets according to certain constraints to test and check our algorithm. In our test sets, all of the data are generated by random numbers.

For bandwidth, because it takes almost no time to transfer data in the same data center, its bandwidth is set to be much larger than other bandwidth values. Other bandwidth is set according to 50% generation probability. For the required data amounts between tasks and datasets, the corresponding demand path is generated according to 20% probability. For every job's tasks, the generation probability of each directed acyclic edge is set to be 20%. Other data in the data set is set according to its possible and reasonable range to ensure that it conforms to the real situation.

Figure 8 shows the simulation result with the double data scale. The running time is 15s, which indicates that our algorithm is efficient enough.

8 Conclusion

Throughout this report, we have introduced a strategy to solve the Scheduling Jobs problem.

We define the notations of this problem first and bring in a simple example without max-min fairness. Then we make formalization for more complex problem, including precedence constraints inside a job itself and indirect transmission between centers. Furthermore, we describe an approximation algorithm to satisfy max-min fairness. The core idea of it is to view overflowing centers as resources with lower capacity and view centers with empty slots as resources with higher capacity.

Moreover, we have proved that the Scheduling Jobs problem is a NP-Complete problem. To prove this, we need to reduce in to a similar and known NPC problem, which is the 3-SAT problem. Because that the max-min fairness is a condition which is to optimize the result after finding out a result. And this does not affect the judgment of NPC, so it is not mentioned in the process of proof. Through the induction of the problem and the explanation of the problem with the 3-SAT example, it can be proved that it is NP-Complete.

We conduct the algorithm based on the analysis of the constraints. Using DAG topological sort, the sparse graph could be read efficiently in the form of adjacency list. With the help of turning the `max` equation into inequalities and using Gurobi, the algorithm could run in a feasible time. Finally, the iteration algorithm could bound the iteration time within the number of data centers.

The test on Toy Data indicates an average time of 10.301s among all jobs, while maintaining max-min fairness of computing resources. The workload among all data centers are balanced.

Acknowledgements

Firstly, thank Mrs. Gao and Mr. Wang for teaching us algorithm and complexity course. We have learned many amazing algorithms through this course without which we can not finish this project. Then thank Prof. Bo Li from Department of Computer Science and Engineering at The Hong Kong University of Science of Technology for motivating this problem. Finally thank TAs Yihao Xie and Haolin Zhou for assistance of this project and algorithm and complexity class.

References

1. Microsoft: Azure global network (2021)
2. Chen, L., Liu, S., Li, B., Li, B.: Scheduling jobs across geo-distributed datacenters with max-min fairness. IEEE Transactions on Network Science and Engineering **6**(3) (jul 2019) 488–500
3. Kwok, Y.K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. **31**(4) (December 1999) 406–471
4. Ullman, J.: Np-complete scheduling problems. Journal of Computer and System Sciences **10**(3) (1975) 384–393

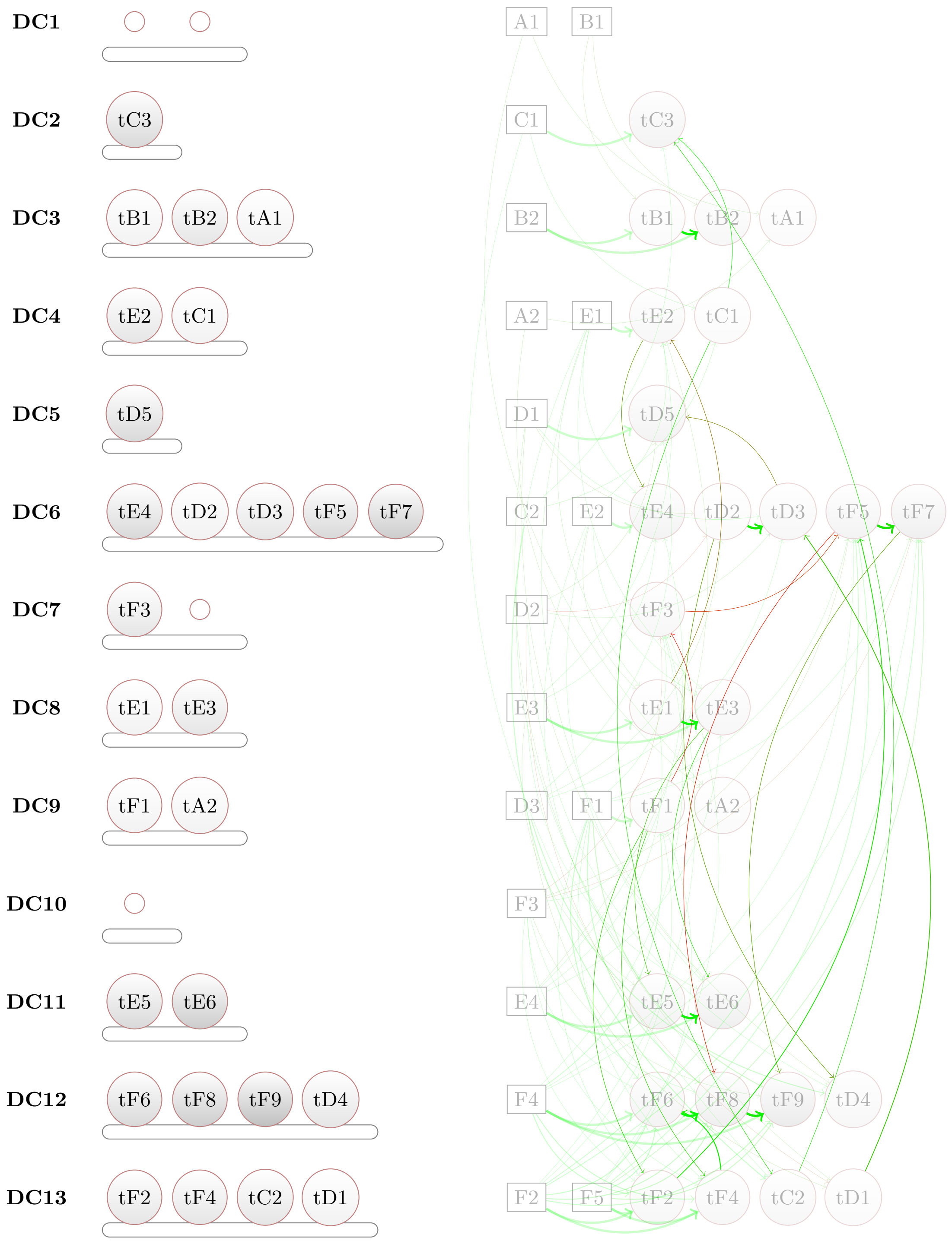

Fig. 6. Toy Data Allocation without max-min

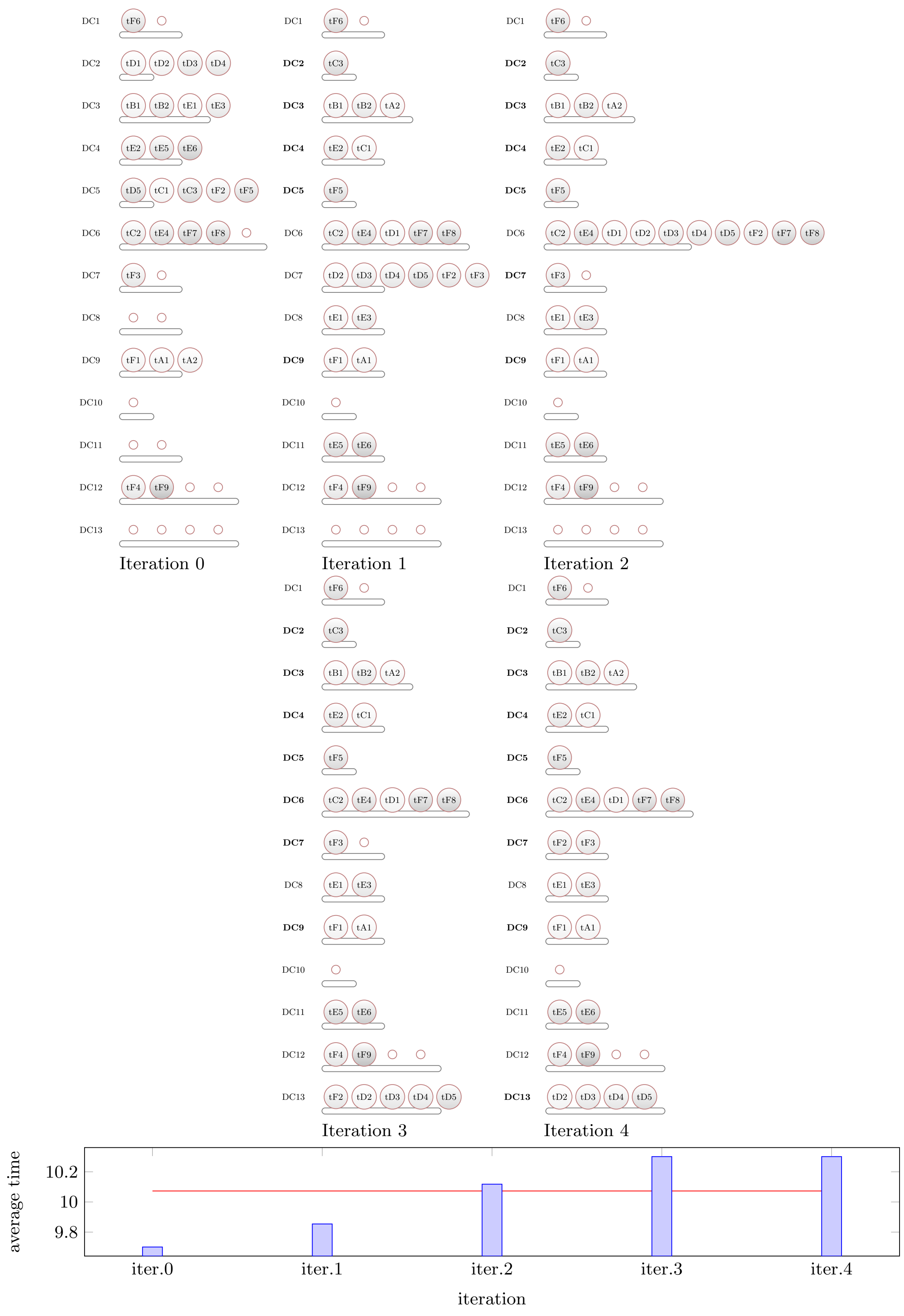

Fig. 7. Max-min iteration

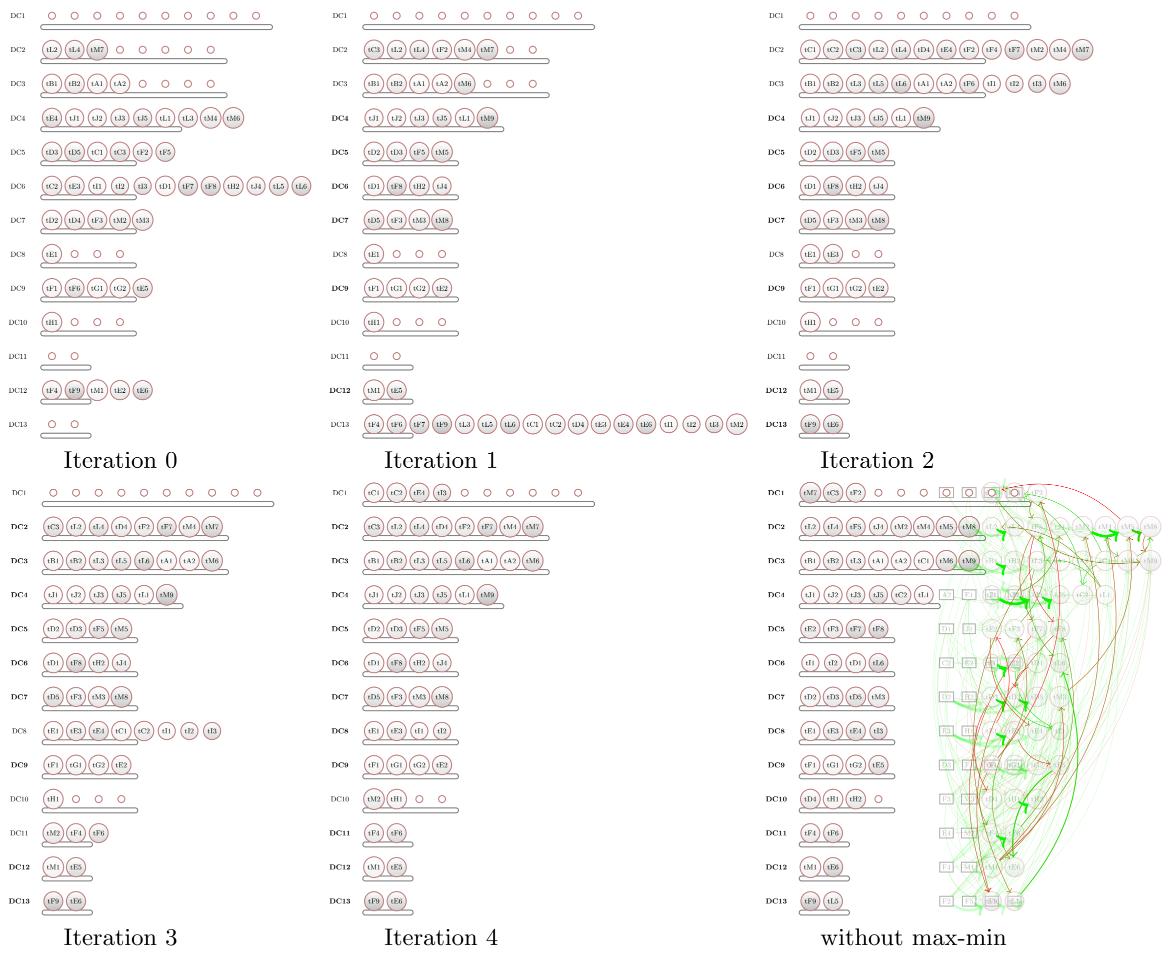

Fig. 8. Real World Simulation