Multi-job Scheduling with Max-Min Fairness based on Greedy Algorithm

Chang Chen (519021910994, ChenChang@sjtu.edu.cn), Renting Rui (519021910451, ruirenting@sjtu.edu.cn), Xin Xu (519021910726, xuxin20010203@sjtu.edu.cn)

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

Abstract. In this paper, we are asked to minimize the average completion time of all jobs while maintaining max-min fairness on computation resource. We formalize the problem by constructing an integer linear programming model and hence it is obviously NP-Complete problem. We adopt the greedy algorithm and set the least transfer time of every task as the assignment strategy. After determining the task assignment by greedy algorithm, we also construct a simulator to simulate the bandwidth allocation by max-min fairness and calculate the total completion time of all the jobs. We analyze the greedy algorithm and get the result of $O(max\{kn, xn^2/a\})$. To test the efficiency of our algorithm, we use toy data test and compare it to the baseline and find our greedy algorithm is obviously better. To better test the efficiency of your design by simulations, we generate data by writing a program which can randomly generate value of variables and a directed acyclic graph to represent the dependency of tasks. We simulate the generated data and produce a good outcome.
Keywords: multi-job scheduling, integer linear programming, greedy algorithm, random program

1 Introduction

1.1 Background

With the development of network, information passing is becoming more and more convenient, and the volume of data – from end users, sensors – has been growing exponentially. To efficiently managing data transferring, cluster system is created, in which a large quantity of data is stored in geographically distributed data centers far away. There are many cluster systems, such as Hadoop, Spark and Dryad. They usually run intensive jobs by generating significant workloads for data centers , providing services such as web search, consumer advertisements and product recommendations. These jobs consist of many tasks and each task demands a certain amount of data as well as an available computing slot to run in parallel. Only when all the tasks of one job have finished, the job is complete.

Usually, there are many geo-distributed data centers in this system. Each data center contains certain data sets which hold necessary data for some tasks to run and a fixed number of available computing slots for tasks to launch. Some data centers are connected through a fixed bandwidth denoting the maximal speed for data transferring between them while others are not connected. Every task should be launched in one of the slots among all data centers and get all the data it needs to execute.

It's clear that the total completion time of all jobs depends on the schedule of tasks launching. Since a job's completion time is determined by its last completed task across the data centers, finishing a portion of the job quickly at one data center does not necessarily result in faster overall job completion time, which increases the complexity of scheduling. So, the main focus of our paper is to find out the optimal scheduling method to minimize total completion time of all jobs in a geo-distributed data centers.

1.2 Our work

Beginning to solve the problem, we search the Internet and read many references to understand the background of the problem and the definition of many concepts, such as max-min fairness and so on. With discussion in the group, we finally come to the conclusion of the problem meaning. The goal of the problem is to assign the tasks to limited slots in different data centers in order to get the least average completion time of all jobs while maintaining max-min fairness on computation resource.

After understanding the problem, we formalize the multi-job scheduling problem with notations and formalize the max-min fairness. We use x_{ij} as the main decision variables as whether to assign $task\_i$ to $slot\_j$ and list several constraints about the slot limitation and bandwidth limitation. We set the expression of minimizing the completion time of all jobs as the target. Then we find it's integer linear programming which is NP problem hard.

With the preparation of understanding the problem and defining the difficulty, we begin to construct the algorithm. We first try the most plainest algorithm, enumerating all the situations where tasks are assigned to different slots. The outcome is not perfect because of the high time complexity. Then we convert to greedy algorithm and choose the least transfer time as the strategy to assign tasks and test the greedy algorithm by toy data. We find the outcome is better than the enumeration time.

Finally, we construct a simulation data to test the efficiency of our algorithm. To construct a simulation data, we write a little program which can randomly generate value of variables and DAG graph. Our simulation data consists of five parts: bandwidth.txt containing bandwidth between data centers; job.txt denoting the number of tasks of each job; resource.txt indicating where data sets locate; slots.txt containing the number of slots each data center holds and task.txt denoting demands of all tasks and execution time of them.

1.3 Assumption

To begin our module, there are some hypothesis as follows:

1. Processors(abstracted to be computation slots and we use the two terms interchangeably): The processor do not share memory and communications rely on message passing. Additionally, all processors are fully connected.
2. the data transfer time: We assume that the inner bandwidth is large enough so that the data transfer time between two slots(processors) in the same datacenter is zero.
3. All the slots in data centers are sufficient to enable all the tasks.
4. We assume all the tasks without dependency connection can execute in parellel.

1.4 Symbol table

Table 1. Symbol Table.

Symbol	Description
$\mathcal{J} = \{1, 2, 3, \cdots, J\}$	a set of parallel jobs needing scheduling
$\mathcal{D} = \{1, 2, 3, \cdots, K\}$	geo-distributed datacenters
$\mathcal{T}_j = \{1, 2, 3, \cdots, n_j\}$	a set of parallel tasks
s_k	the capacity of available computing slots
$c_{i,k}^j$	the task i assigned to datacenter k
$e_{i,k}^j$	the execution time of the same condition
S_i^j	the set of datacenters holding all the data that task i from job j needs
$d_i^{j,s}$	the amount of data that source datacenter s hasn't transfer to task i
$b_{s,i}^k$	the bandwidth between datacenter s to task i launched in datacenter k
$d_{i,b}^{j,s}$	the true bandwidth of a certain bandwidth $b_{s,i}^k$
$x_{i,k}^j$	the launch of task
τ_j	the completion time of job j
L_k	a sequence ordered by the value of $d_i^{j,s}$ increasingly
B_k	corresponding bandwidth sequence of L_k
$X_{m\times n}$	a binary variable matrix in ILP
$B_{m\times m}$	bandwidth matrix in ILP
$D_{n\times m}$	demand matrix in ILP
$C_{n\times m}$	target matrix of ILP

2 Problem Analysis

2.1 Formalization of the Problem and Max-Min Fairness

Formalization of the Problem Suppose there are a set of parallel jobs $\mathcal{J} = \{1, 2, 3, \cdots, J\}$ need scheduling. The input data of these jobs are distributed across a set of geo-distributed datacenters, represented as $\mathcal{D} = \{1, 2, 3, \cdots, K\}$. Every job $j \in \mathcal{J}$ is consisted of a set of parallel tasks $\mathcal{T}_j = \{1, 2, 3, \cdots, n_j\}$,

which are ready to be put in an available computing slots in these datacenters. For every datacenter $k \in \mathcal{D}$, we use s_k to represent the capacity of available computing slots.

For each task $i \in \mathcal{T}_j$ of job j, the time for the network to transfer the needed data to the task i assigned to datacenter k is denoted by $c_{i,k}^j$, and the execution time of the same condition is represented by $e_{i,k}^j$. Let S_i^j denotes the set of datacenters which holds all the data that task i from job j needs, so the task i would traverse all the source datacenter $s \in S_i^j$. In this condition, the amount of data that source datacenter s hasn't transfer to task i is represented as $d_i^{j,s}$, and let $b_{s,i}^k$ represents the bandwidth of the link between datacenter s to task i which is launched in the slot of datacenter k. We hypothesis that the bandwidth $b_{s,i}^k$ is periodically changed and is up to the current demand $d_i^{j,s}$, so we define $d_{i,b}^{j,s}$, in which b represents as the value of current bandwidth, as the actual amount of data pass between datacenter s to task i of a certain bandwidth $b_{s,i}^k$. And the sum of bandwidth between datacenter s and datacenter k is fixed, which equals to the total bandwidth between datacenter s to all the tasks in datacenter k. It is expressed as follows:

$$\sum_{i \in \mathcal{T}_j, j \in \mathcal{J}} b_{s,i}^k = b_{s,k} \tag{1}$$

With symbols defined above, we can get the method to calculate $c_{i,k}^j$ as follows:

$$c_{i,k}^j = \begin{cases} 0, & \text{when } S_i^j = \{k\}; \\ \max\limits_{s \in S_i^j, s \neq k} \sum\limits_{b \neq 0} \frac{d_{i,b}^{j,s}}{b_{s,i}^k}, & \text{otherwise.} \end{cases} \tag{2}$$

To better represent the time for a job to complete, we assume a binary variable $x_{i,k}^j$ to denote the assignment of a task: when the task i of job j is assigned to a slot in datacenter k, $x_{i,k}^j$ is set to 1, otherwise, $x_{i,k}^j$ is set to 0, as expressed follows:

$$\sum_{k \in \mathcal{D}} x_{i,k}^j = 1, \forall i \in \mathcal{T}_j, \forall j \in \mathcal{J} \tag{3}$$

$$x_{i,k}^j \in \{0,1\}, \forall i \in \mathcal{T}_j, \forall j \in \mathcal{J}, \forall k \in \mathcal{D}. \tag{4}$$

Additionally, a job j completes only when its slowest task completes, so the completion time of job j, which is represented by τ_j, is determined by the maximum completion time among all its tasks, expressed as follows:

$$\tau_j = \max_{i \in \mathcal{T}_j, k \in \mathcal{D}} x_{i,k}^j (c_{i,k}^j + e_{i,k}^j) \tag{5}$$

Meanwhile, the number of tasks assigned to a certain datacenter k can't exceed its slot capacity s_k, represented as follows:

$$\sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{T}_j} x_{i,k}^j \leq s_k, \forall k \in \mathcal{D} \tag{6}$$

So, with all the definitions above, the average completion time among all the jobs represented by f can be expressed as follows, in which J is the number of jobs:

$$f = \frac{\sum\limits_{j \in \mathcal{J}} \tau_j}{J} \tag{7}$$

Max-Min Fairness Consider a datacenter with multiple slots and launch more than one task in it. Cases are that tasks in the same datacenter may compete for resources. For example, the total bandwidth between datacenter s to datacenter k is fixed, known as $b_{s,k}$ for all the tasks in datacenter k, but every task wants as much as possible bandwidth to them to complete data transition. How to allocate the total bandwidth $b_{s,k}$ to each task in datacenter k so that it reaches a rather fair condition? To tackle this problem, we will specify max-min fairness to better suit the module.

Recall the condition, for all the tasks assigned to the datacenter k, we reorder them into a sequence L_k by the value of $d_i^{j,s}$ increasingly, represented as follows:

$$L_k = (i_1, i_2, i_3, \cdots, i_r), \forall 1 \le m \le r-1, d_{i_m}^{j_m,s} \le d_{i_{m+1}}^{j_{m+1},s} \tag{8}$$

To maintain the max-min fairness, we should figure out a bandwidth sequence:

$$B_k = (b_{s,i_1}^k, b_{s,i_2}^k, b_{s,i_3}^k, \cdots, b_{s,i_r}^k), \forall 1 \le m \le r-1, b_{s,i_m}^k \le b_{s,i_{m+1}}^k \tag{9}$$

And it has properties as follows:

$$d_{i_p}^{j_p,s} = b_{s,i_p}^k, \exists h, s.t 1 \le h \le r, \forall 1 \le p \le h. \tag{10}$$

$$b_{s,i_q}^k = b_{s,i_{q+1}}^k, \forall h < q \le r-1. \tag{11}$$

$$\sum_{m=1}^{r} b_{s,i_m}^k = b_{s,k} \tag{12}$$

So, to get a max-min fairness allocation method, we can do steps as follows:

Firstly, divide $b_{s,k}$ by the number of tasks launched into datacenter k, and compare the answer with $d_{i_1}^{j_1,s}$. If $\frac{b_{s,k}}{k} \le d_{i_1}^{j_1,s}$, then allocation can end by allocating every task the bandwidth of $\frac{b_{s,k}}{k}$; else allocate the first task i_1 in L_k the bandwidth of $d_{i_1}^{j_1,s}$ and use the same method to allocate the remained bandwidth $d_{s,k} - d_{i_1}^{j_1,s}$ with the remained tasks $L_k - i_1$. It's clear that this algorithm is recursive.

2.2 Problem Complexity Analysis

This job scheduling problem can be described as follow:

Given a limit average completion time t, whether or not there is an assignment of tasks placement across geo-distributed data centers so that the final average completion time T of all jobs satisfying $T \le t$?

This decision problem is a NP-Complete Problem, and the proof is as follows:

Proof. Firstly we should prove it a NP Problem.

This statement is clearly true because the check program should launch all tasks and calculate completion time of each task to get the total finishing time of each job. Additionally, certificate should check limit conditions while traversing: dependency between tasks and whether the workload exceeds scheduling capacity, which can be done within constant steps. The critical part of certificate is traversing, which means the total running time of this program is in polynomial time.

Then, we should prove it a NP-complete problem:

We will prove it by reducing Scheduling Jobs across Geo-Distributed Data Centers from Integer Linear Programming Problem as follows:

For Integer Linear Programming, suppose that there is a variable matrix $X_{m\times n}$ with binary value $1, 0$. Then, there are also a bandwidth matrix $B_{m\times m}$ and a demand matrix $D_{n\times m}$. The limit constraints are:

$$\sum_{1\le i\le m} x_{ij} = 1 \tag{13}$$

$$\sum_{1\le j\le n} x_{ij} = 1 \tag{14}$$

$$x_{ij} \in \{1, 0\}, \forall 1 \le i \le m, 1 \le j \le n \tag{15}$$

Define a target matrix $C_{n\times m}$, $c_{ij} = \max\{\frac{d_{i1}}{b_{j1}}, \frac{d_{i2}}{b_{j2}}, \frac{d_{i3}}{b_{j3}}, \ldots, \frac{d_{im}}{b_{jm}}\}$. With this definition above, we can define the target function as follows:

$$\min \sum diag(C \times X). \tag{16}$$

For Scheduling Jobs Across Geo-Distributed Data Centers, suppose that there is m data centers and every data center only has one slot. And there are n jobs, each job only has one task to do.

Additionally, there is no dependency between tasks so that all tasks can run independently. We define that for the variable matrix X:

$$x_{ij} = \begin{cases} 1 & \text{task } j \text{ is launched in slot } i \\ 0 & \text{others} \end{cases} \tag{17}$$

And for the bandwidth matrix B, the value of b_{ij} means the total bandwidth between data center i and data center j. Without loss of generality, we hypothesis that the bandwidth from data center i to data center j is the same with the bandwidth from data center j to data center i. It's easy to know that matrix B is symmetrical. For the demand matrix D, the value of d_{ij} is the resources that task i demands from data center j. For target matrix C, the value of c_{ij} means the completion time of task i if task i is launched in data center j. So, the target function is to minimize the sum of each job's completion time.

With these constructions above, we can easily know that this integer linear programming solves if and only if this scheduling jobs problem solves. Since Integer Linear Programming Problem is a NP-Complete problem, Scheduling Jobs across Geo-Distributed Data Centers is also a NP-Complete problem.

3 Greedy Algorithm

We divide the problem to two parts: the algorithm to determine task assignment and simulate the procedure of bandwidth assignment and task completion and calculate the final execution time. In terms of the task assignment, we initially try the enumeration algorithm, but the outcome is not perfect. Then we convert to greedy algorithm and choose two assignment strategies: priority allocation for jobs with **higher demand** or priority allocation for jobs with **a longer chain of tasks**. In terms of simulation, we adopt several stages to simulate the procedure: bandwidth allocation by max-min fairness, task execution and reallocation until all the jobs are finished. At last, we calculate the total execution time. As Figure1, we can get the whole algorithm.

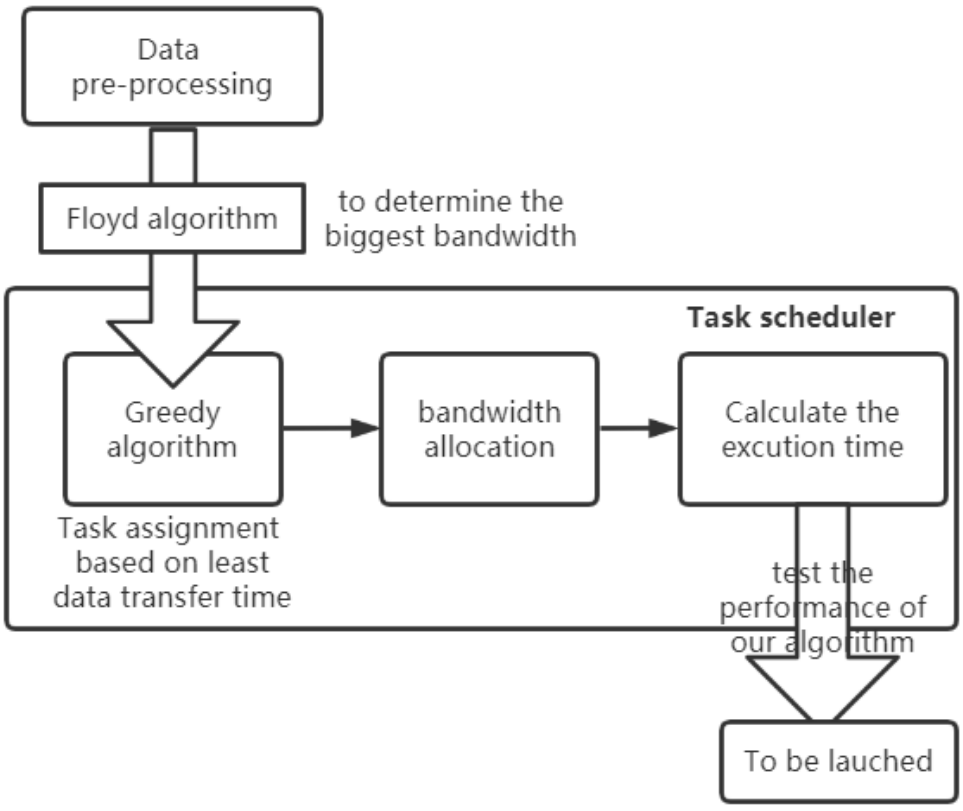

Fig. 1. Flowchart of the algorithm

3.1 Data pre-processing

In face of the raw data, we process the data into data structures which we need for the following algorithm. We read the **toydata.csv** and acquire data center details at first. We get the number of data centers and the number of slots in every data center. Then we acquire the information about bandwidth between different data centers. We construct a two dimensional arrays **bandwidth**[·][·] to record the bandwidth from $datacenter_i$ to $datacenter_j$. Then we acquire the job list details. We construct a two-dimensional array **TaskDemand**[·][·] to record how much data $task_i$ ask from $datacenter_j$. Moreover, under consideration of task dependency, we also build a two-dimensional array. If $task_i$ asks data from $task_j$, we set $A[i][j]$ as one. We also record the execution time of different tasks for calculating the overall execution time.

data structure	description
struct data center	record the slot numbers of one data center and its' all serving demands
bandwidth[·][·]	the bandwidth between two different data centers
struct demand	record the asker and receiver of the demand and whether it's satisfied
struct task	all the task's demands, task dependency and the state of task
struct job	all the job's tasks, start time, finish time and the state of the job

After we pre-process the data after importing, we determine optimal bandwidth between data centers.To get the least completion time, we hope that the bandwidth between data centers achieves the biggest. So we choose the Floyd algorithm to get the biggest bandwidth since Floyd algorithm is a simple way to realize. The following algorithm exhibits how the floyd algorithm assigns the bandwidth.

```
for i← 1to n do
    for j← 1to n do
        for k← 1to n do
            if bandwidth[i][k]+bandwidth[k][j] is less than bandwidth[i][j] then
                bandwidth[i][j]← bandwidth[i][k] + bandwidth[k][j] end
            end
        end
    end
    return bandwidth
```

3.2 Task assignment

To arrange tasks, we initially adopt the enumeration algorithm: enumerate all the situations where tasks are assigned to different data centers respectively since we suppose all the slots are sufficient for all the tasks. However, when we test the algorithm, we find the time cost is too large to produce the correct answer.

Then we put forward a new algorithm, greedy algorithm, to determine the task assignment in advance by adding some constraints as assignment strategy.

The first greedy strategy is to sort all the tasks' demands by their value and give priority to tasks with high demands. Tasks with high demands are assigned to the source data center of the demand. The reason why we choose this strategy is that the bandwidth transfer between slots in one data center is far greater than the bandwidth transfer between different data centers. Giving priority to big demands can sufficiently decrease the transfer time when the tasks with big demands exist on other slots.

Considering the tasks' dependency, we can construct a directed acyclic graph to portray this dependency, where nodes represent tasks and edges describes that one task at the end of edges needs data from the task at the start end of edges. The second greedy strategy, which takes tasks' dependency in consideration, is to sort the tasks' chains and give priority to a long chain of tasks. This strategy avoids the situation where the long chain cannot have the chance to satisfy demands until all the other

tasks are executed. If we satisfy the job with long chain, we can exploit bandwidth as large as possible.

Algorithm 1: Task Assignment

Input: Tasks,demands,bandwidth
Output: The result of task assignment,$dc[\cdot]$

```
Sort Task[] with its greatest need nonincreasingly;
for i← 1to task_count do
    fetch_time[i]←0
    for j← 1to dataCenter_num do
        if DataCenter[j] can hold then
            time← the time of fetching data Task[i] needs to DataCenter[j] end
            fetch_time[i]← min{fetch_time[i], time} end
        end
        return DataCenter[j];
```

3.3 Procedure simulator about data transfer and task execution

We set up a simulator to simulate the procedure of bandwidth allocation and task execution. To assign bandwidth fairly, we adopt max-min fairness algorithm by assigning the bandwidth to satisfiable demands and averages the bandwidth to the remaining demands if it is not sufficient. After one of the tasks finishes to execute or one of the demands is satisfied, we update all the demands and reassign the bandwidth until all the tasks are finished.

Every time before reallocating the bandwidth, we subtract the allocated bandwidth from the existing demands. Moreover, we calculate the current time by adding the first satisfied demand's transfer time. We divide the demand value by the allocated bandwidth as the transfer time.

Algorithm 2: Simulator

Input: Task assignment, bandwidth
Output: Shortest average time

```
while there exists unfinished job[] do
    Add_available_demand();
    Max_min_fairness();
    Update();
end
return Shortest_average_time;
```

Algorithm 3: Add_available_demand

Input: Task[], demand_list
Output: Void

```
for Task[i] is available and unexecuted do
    if Task[i] .job_j is unstarted then
        started the job_j;
        job_j .start_time ← current_time end
        for Task[i] .demand_k do
            demand_list ← demand_list + {demand_k};
        end
        Task[i]- executed ← true end
```

Algorithm 4: Max_min_fairness

Input: demand_list, bandwidth[][]
Output: Void

```
array edge[dataCenter_num][dataCenter_num][·]to hold demands
for edge[from][to] do
    while (edge[from][to].bandwidth-cost)/hold_num ¿ demand_k.total_need do
        demand_k.allocated_bandwidth ←demand_k.total_need;
        cost ← cost+demand_k.allocated_bandwidth;
        hold_num ← hold_num − 1
        pick up a new demand_k;
    end
    for remaining demand_k do
        demand_k.allocated_bandwidth ← (edge[from][to].bandwidth − cost)/hold_num end
    end
```

3.4 Algorithm Complexity

Denote the number of demands as n, the number of tasks as m and the number of data centers as k. What's more, the minimum of bandwidth is a and the total need of all the demands are x.

First, for task assignment part, we need to put tasks, sorted by the amount of their demands non-increasingly, into each data center. Thus the sorting work takes $O(n \log n)$ and the matching tasks with data centers by fetching-data-time takes $O(kn)$.

Then we come to the simulation part, the takes at most x/a loops, where one loop includes $Add\_available\_demand$, $Max\_min\_fairness$ and $Update$ each of time complexity $O(n)$, $O(n)$ and $O(n^2)$. Therefore it takes $O(xn^2/a)$.

In conclusion, the time complexity is $O(max\{kn, xn^2/a\})$.

3.5 Test Result

We test our greedy algorithm by toy data and get the task assignment and the final total execution time. As depicted in Figure 2, tasks are allocated to different data slots chosen by our greedy algorithm. We also get the completion time of our greedy algorithm is 8.6377 milliseconds. To test the efficiency of our algorithm, we enumerate other task allocation as the baseline and compare the outcome. We find the greedy algorithm is obviously better than other allocation chosen by plain enumeration, as shown in Figure 3. Although we do not have the optimal result, we also get the nice approximate solution.

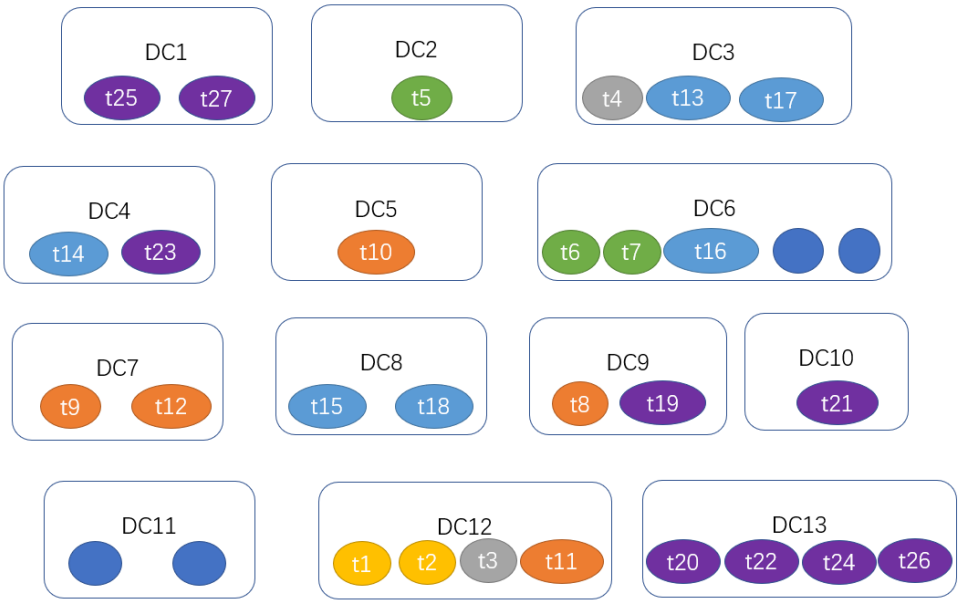

Fig. 2. Test result of task assignment

Fig. 3. Comparison between baseline and our greedy algorithm

4 Test by simulations

The above shows the process of how we test our algorithm.

4.1 Generate data

Firstly, we try to search the Internet to find available data for simulation. However, the available data on the internet is in different format of the toy data. It would consume us much time to pre-process the available data on the internet and cases are that this processed data can not match our algorithm. So, finally we decide to construct simulation data by ourselves in random algorithm.

Our constructed data is composed of five parts: bandwidth.txt contains the value of bandwidth from one data center to another; job.txt contains the number of tasks of each job; resource.txt denotes which data center the data set locates; slots.txt indicates the number of slots in each data center; task.txt contains the demand of all tasks and the execution time of them.

Our idea of constructing the simulation data is as follows: firstly, we fix the number of tasks of each job by generating a random number. Then, we determine the dependency of these tasks by constructing a DAG from a matrix. For example, if there is a job with n tasks, we construct a matrix H with the size of $n \times n$ and randomly pick m edges $e(i, j)$ which denotes task i needs data from task j. Here, we suppose

that $m = 8$ and there are 7 tasks. The matrix is as follows:

$$H = \begin{bmatrix} 0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0 \\ 1\,0\,0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,1\,1\,0 \end{bmatrix} \tag{18}$$

So, the DAG of this matrix is:

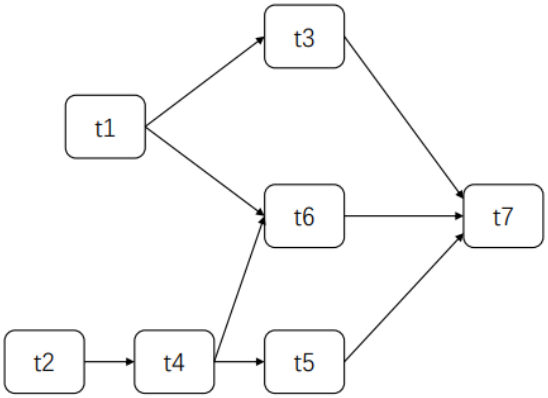

Fig. 4. The DAG of dependency matrix.

As long as the dependency between tasks is fixed, other data such as the bandwidth and demands, can be randomly generated since there is no limit in them.

4.2 Test result

We test the efficiency of our greedy algorithm by running random data we construct, getting the task assignment and the final total execution time. The average completion time is 6.87268. To test the efficiency of our algorithm, we enumerate other task allocation as the baseline and compare the outcome. We find the greedy algorithm is obviously better than other allocation chosen by plain enumeration. The comparison between baseline and our greedy algorithm is depicted in Figure **??** below.

Acknowledgements

Thanks for our teammates to work jointly to figure out this problem. We company each other in the late night, which encourages us to move on and never give up. Additionally, thanks for our teacher assistants Haolin Zhou for his instructed suggestions help us a lot to understand this question. Our classmates give us many help too. Communicating with them, we can change our wrong logic in time and solve bugs when algorithms go wrong. At last, thanks for Prof.Gao and Prof.Wang, who engage in whole semester and teach us a lot.

References

1. Author, Bo Li.: Scheduling Jobs across Geo-Distributed Data Centers. Journal **2**(5), 99–110 (2021)
2. Author, Chien-Chun Hung,Author, Leana Golubchik,Author, Minlan Yu.: Scheduling Jobs Across Geo-distributed Datacenters. (2016).
3. Author, Li Chen,Author, Shuhao Liu,Author, Bo Li,Author, Baochun Li.: Scheduling Jobs Across Geo-distributed Datacenters with Max-Min Fairness. (2017).