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

Project for Algorithm and Complexity

Taoran Han, Longxuan Wei, Zilong Li

May 30, 2021

Part I

Algorithm Design

Introduction

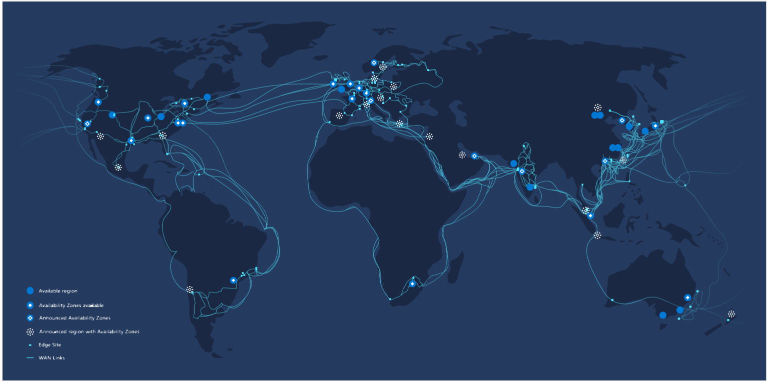

Available region
Availability Zones available
Announced Availability Zones
Announced region with Availability Zones
Edge Site
WAN Links

Figure: Data centers and its connection in Azure global network[1]

[1] Microsoft. *Azure Global Network*. 2021. URL: https://azure.microsoft.com/en-us/global-infrastructure/global-network/.

Notation

Table: 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
$d_{k,i}^{j}$	The amount of data that task i of job k requires from center j
$T_{i,j}^{k}$	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_i^k	The number of tasks of stage i of job k
$x_{k,i}^{j}$	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

Framework

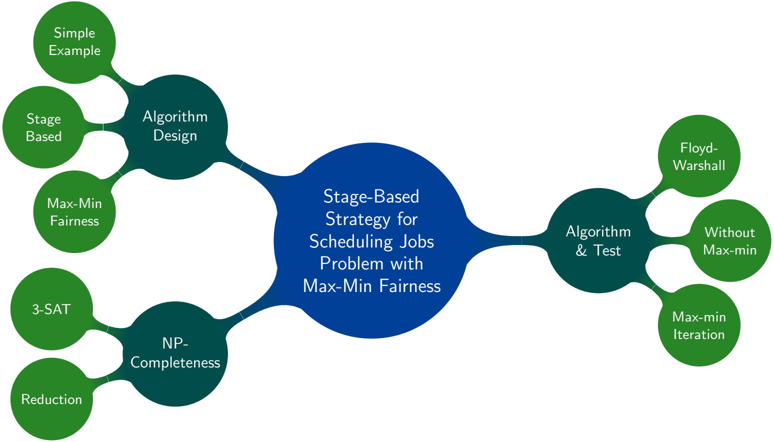


Figure: Framework of this project

Initial Constraints

We need to restrict that each task can only be allocated to exactly one center. And for every center, the number of its allocated tasks can not exceed the number of its slots. With these two constraints, we will then calculate the completion time.

$$\sum_{j=1}^{m} x_{k,i}^{j} = 1, \forall i \leq t_i, \forall k \leq K \tag{1}$$

$$\sum_{k=1}^{K} \sum_{i=1}^{t_k} x_{k,i}^{j} \leq s_j, \forall j \leq m \tag{2}$$

$$c_{k,i} = \max_{1 \leq j_2 \leq m} \sum_{j_1=1}^{m} \frac{x_{k,i}^{j_1} \times d_{k,i}^{j_2}}{b_{j_2,j_1}} + e_{k,i} \tag{3}$$

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

A Simple Example

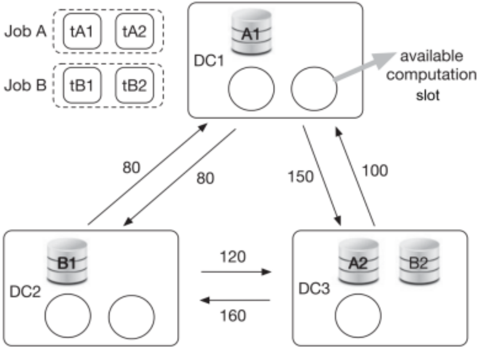
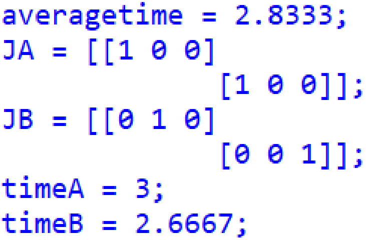


Figure: The simple example

A Simple Example



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

Figure: Result of the simple example

More Constraints for Stages

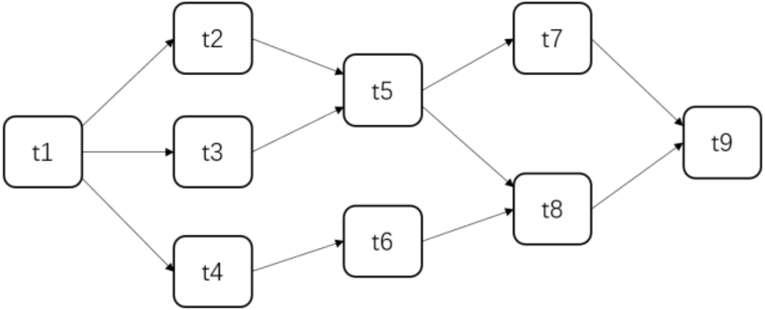


Figure: A sample of a job with different stages

More Constraints for Stages

The completion time of a job is the summation of all stages. Moreover, we need to consider the transmission between tasks of the same job.

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

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

Floyd-Warshall on Bandwidth I

We can calculate the corresponding bandwidth between every two data centers. Considering that the data transferring path need to be optimal (the data could be transferred indirectly), we will use the Floyd-Warshall's Algorithm.
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.

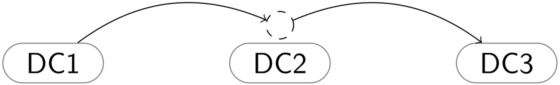


Figure: Indirect transfer through a *transfer* task

Floyd-Warshall on Bandwidth II

The reciprocal of the bandwidth is the transfer time through this line. And the bandwidth could be calculated reversely after the process of Floyd-Warshall.

$$\frac{1}{b_{i,j}} \rightarrow \boxed{\textbf{Floyd-Warshall}} \rightarrow \frac{1}{b'_{i,j}}$$

Max-Min Fairness

Initial request: Minimize average completion time.

View overflowing centers as smaller-size glasses. View centers with empty slots as larger-size glasses. Pour overflowing beer into glasses with capacity.

Loop the procedure again and again until there are no overflowing glasses.

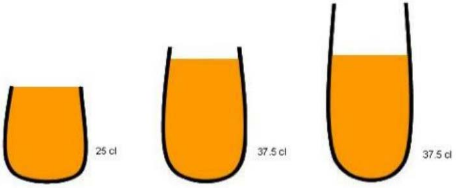


Figure: Max-min Example on Beer

Part II

NP-Completeness

3-SAT

The problem is to schedule all the tasks in the DAG to a number of worker nodes — while minimizing the average time of the job.
To prove this, we need to reduce into 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.

Definition (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 $\overline{x}_i$.

Construction 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 $1 \leq j \leq n$;
Given a set S of all n tasks which can be put into different DAG's, a limit k, a relation $<$ on S which can be determined from the DAG's directed edges. Our goal is to find a function f from S to $\{0, 1, \ldots, t-1\}$ to make sure that the average cost will be minimum.

Construction II

The relation ¡ is given by:

1. $x_{ij} < \overline{x}_{i,j+1}$ and $\overline{x}_{ij} < \overline{x}_{i,j+1}$, for $1 \leq i \leq m$ and $0 \leq j < m$;
2. $x_{i,i-1} < y_i$ and $\overline{x}_{i,i-1} < \overline{y}_i$ for $1 \leq i \leq m$;
3. 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.

Reduction I

We will show that the above instance has a solution if and only if the given instance of $\mathrm{3\text{-}SAT}$ does.

$$\mathrm{3\text{-}SAT} \leq_P \mathrm{DAG}\ \textsc{Job Scheduling}$$

where $\mathrm{3\text{-}SAT}$ is a NP-Complete problem, so is $\mathrm{DAG}\ \textsc{Job Scheduling}$.

Reduction II

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.

Reduction III

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.

Reduction IV

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.

Reduction V

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.

Part III

Algorithm & Test

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

DAG Stage Seperation

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

The algorithm is efficient on sparse 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.

Improvement on `max` I

It is discovered that using built-in function `max` in CPLEX is *slow*. We turned the max function into inequalities.

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

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

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

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

Improvement on max II

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.

Iterative Max-Min Fairness

Algorithm 2: Iterative Max-Min Fairness Algorithm

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

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

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

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

Toy Data (without max-min)

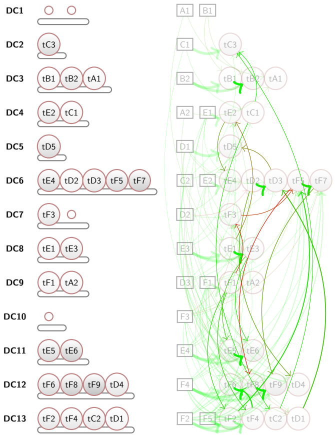


Figure: Toy Data Allocation without max-min

Toy Data (max-min iteration)

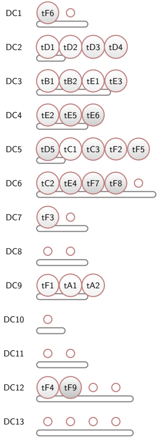
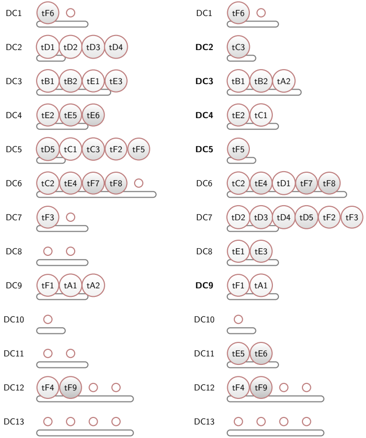


Figure: Iteration on Max-min Fairness

Toy Data (max-min iteration)

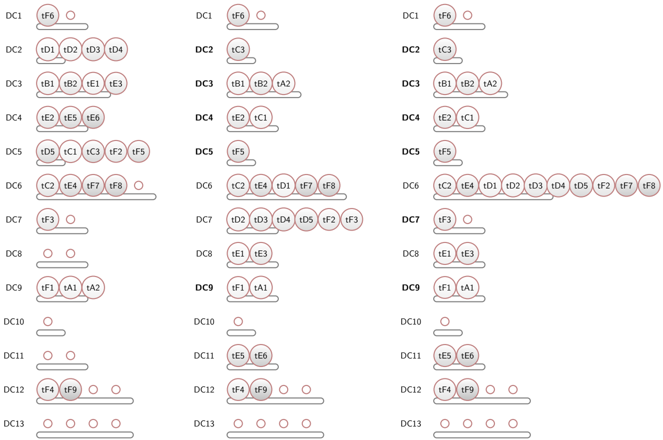


DC1 tF6
DC2 tD1 tD2 tD3 tD4
DC3 tB1 tB2 tE1 tE3
DC4 tE2 tE5 tE6
DC5 tD5 tC1 tC3 tF2 tF5
DC6 tC2 tE4 tF7 tF8
DC7 tF3
DC8
DC9 tF1 tA1 tA2
DC10
DC11
DC12 tF4 tF9
DC13

DC1 tF6
DC2 tC3
DC3 tB1 tB2 tA2
DC4 tE2 tC1
DC5 tF5
DC6 tC2 tE4 tD1 tF7 tF8
DC7 tD2 tD3 tD4 tD5 tF2 tF3
DC8 tE1 tE3
DC9 tF1 tA1
DC10
DC11 tE5 tE6
DC12 tF4 tF9
DC13

Figure: Iteration on Max-min Fairness

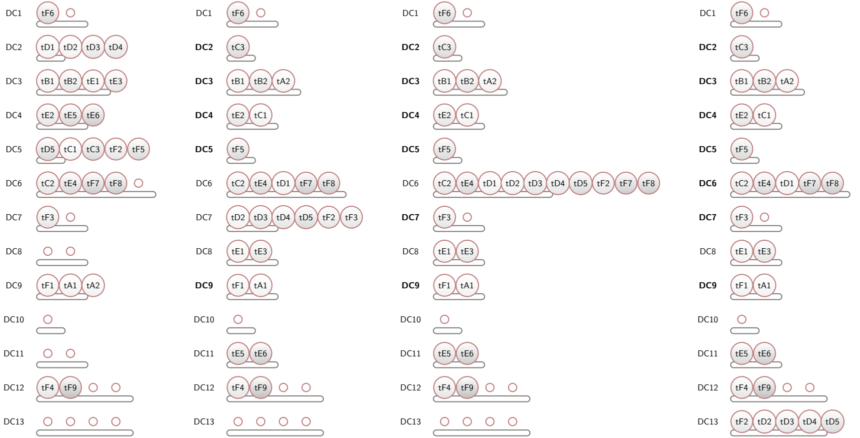
Toy Data (max-min iteration)



DC1	tF6 ○	DC1	tF6 ○	DC1	tF6 ○
DC2	tD1 tD2 tD3 tD4	**DC2**	tC3	**DC2**	tC3
DC3	tB1 tB2 tE1 tE3	**DC3**	tB1 tB2 tA2	**DC3**	tB1 tB2 tA2
DC4	tE2 tE5 tE6	**DC4**	tE2 tC1	**DC4**	tE2 tC1
DC5	tD5 tC1 tC3 tF2 tF5	**DC5**	tF5	**DC5**	tF5
DC6	tC2 tE4 tF7 tF8 ○	DC6	tC2 tE4 tD1 tF7 tF8	DC6	tC2 tE4 tD1 tD2 tD3 tD4 tD5 tF2 tF7 tF8
DC7	tF3 ○	DC7	tD2 tD3 tD4 tD5 tF2 tF3	**DC7**	tF3 ○
DC8	○ ○	DC8	tE1 tE3	DC8	tE1 tE3
DC9	tF1 tA1 tA2	**DC9**	tF1 tA1	**DC9**	tF1 tA1
DC10	○	DC10	○	DC10	○
DC11	○ ○	DC11	tE5 tE6	DC11	tE5 tE6
DC12	tF4 tF9 ○ ○	DC12	tF4 tF9 ○ ○	DC12	tF4 tF9 ○ ○
DC13	○ ○ ○ ○	DC13	○ ○ ○ ○	DC13	○ ○ ○ ○

Figure: Iteration on Max-min Fairness

Toy Data (max-min iteration)



DC1	tF6 ○	DC1	tF6 ○	DC1	tF6 ○	DC1	tF6 ○
DC2	tD1 tD2 tD3 tD4	**DC2**	tC3	**DC2**	tC3	**DC2**	tC3
DC3	tB1 tB2 tE1 tE3	**DC3**	tB1 tB2 tA2	**DC3**	tB1 tB2 tA2	**DC3**	tB1 tB2 tA2
DC4	tE2 tE5 tE6	**DC4**	tE2 tC1	**DC4**	tE2 tC1	**DC4**	tE2 tC1
DC5	tD5 tC1 tC3 tF2 tF5	**DC5**	tF5	**DC5**	tF5	**DC5**	tF5
DC6	tC2 tE4 tF7 tF8 ○	DC6	tC2 tE4 tD1 tF7 tF8	DC6	tC2 tE4 tD1 tD2 tD3 tD4 tD5 tF2 tF7 tF8	**DC6**	tC2 tE4 tD1 tF7 tF8
DC7	tF3 ○	DC7	tD2 tD3 tD4 tD5 tF2 tF3	**DC7**	tF3 ○	**DC7**	tF3 ○
DC8	○ ○	DC8	tE1 tE3	DC8	tE1 tE3	DC8	tE1 tE3
DC9	tF1 tA1 tA2	**DC9**	tF1 tA1	**DC9**	tF1 tA1	**DC9**	tF1 tA1
DC10	○	DC10	○	DC10	○	DC10	○
DC11	○ ○	DC11	tE5 tE6	DC11	tE5 tE6	DC11	tE5 tE6
DC12	tF4 tF9 ○ ○	DC12	tF4 tF9 ○ ○	DC12	tF4 tF9 ○ ○	DC12	tF4 tF9 ○ ○
DC13	○ ○ ○ ○	DC13	○ ○ ○ ○	DC13	○ ○ ○ ○	DC13	tF2 tD2 tD3 tD4 tD5

Figure: Iteration on Max-min Fairness

Toy Data (max-min iteration)

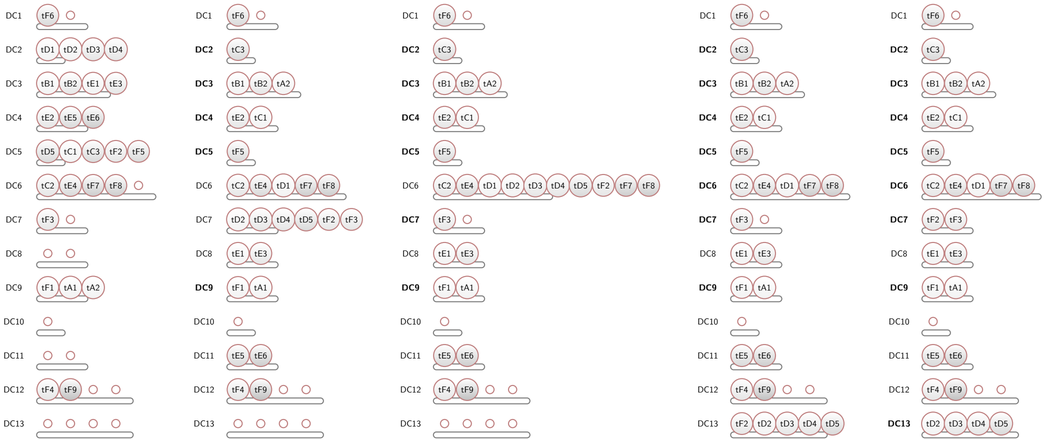


Figure: Iteration on Max-min Fairness

Toy Data Result

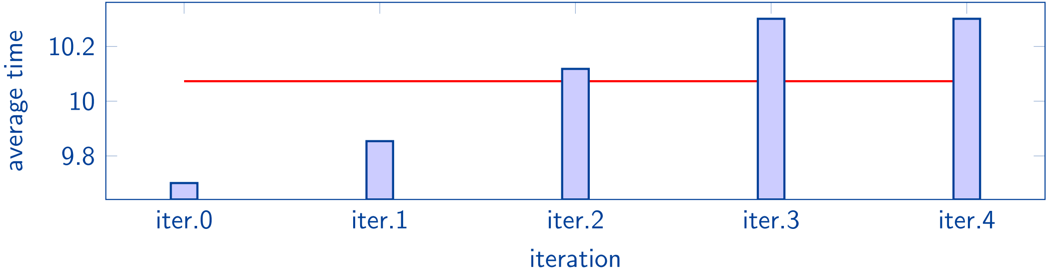


Figure: Average Time

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.

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.

Real World (without max-min)

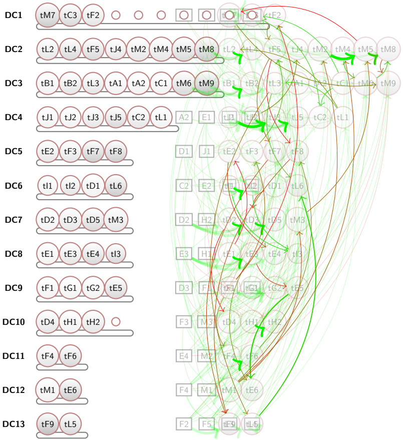


Figure: Real World Allocation without max-min

Real World (max-min iteration)

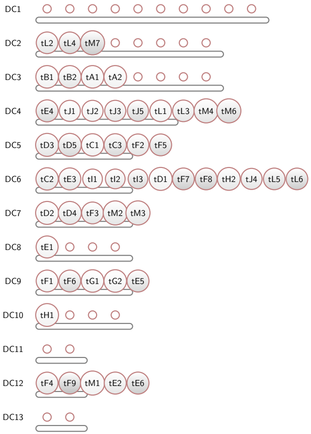


Figure: Iteration on Max-min Fairness

Real World (max-min iteration)

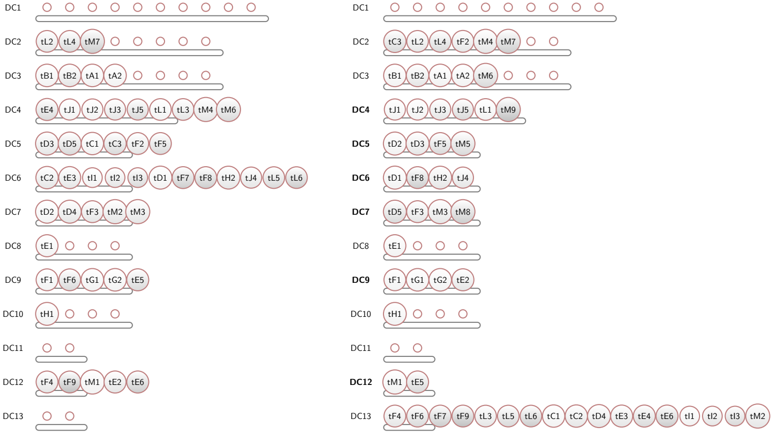


Figure: Iteration on Max-min Fairness

Real World (max-min iteration)

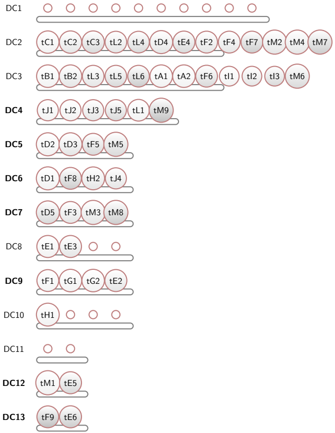
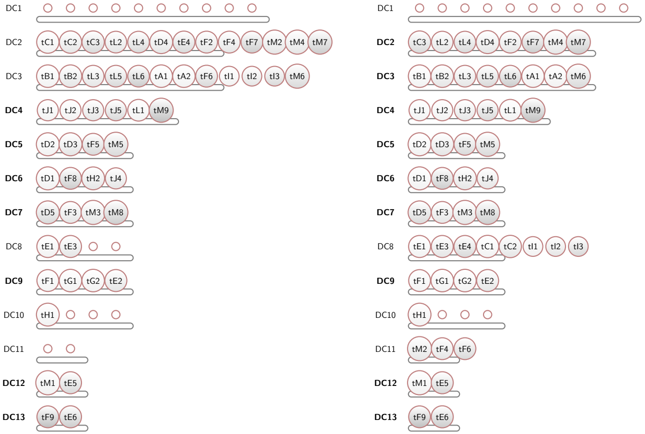

DC1
DC2 tC1 tC2 tC3 tL2 tL4 tD4 tE4 tF2 tF4 tF7 tM2 tM4 tM7
DC3 tB1 tB2 tL3 tL5 tL6 tA1 tA2 tF6 tI1 tI2 tI3 tM6
DC4 tJ1 tJ2 tJ3 tJ5 tL1 tM9
DC5 tD2 tD3 tF5 tM5
DC6 tD1 tF8 tH2 tJ4
DC7 tD5 tF3 tM3 tM8
DC8 tE1 tE3
DC9 tF1 tG1 tG2 tE2
DC10 tH1
DC11
DC12 tM1 tE5
DC13 tF9 tE6

Figure: Iteration on Max-min Fairness

Real World (max-min iteration)



DC1 ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
DC2 tC1 tC2 tC3 tL2 tL4 tD4 tE4 tF2 tF4 tF7 tM2 tM4 tM7
DC3 tB1 tB2 tL3 tL5 tL6 tA1 tA2 tF6 tI1 tI2 tI3 tM6
DC4 tJ1 tJ2 tJ3 tJ5 tL1 tM9
DC5 tD2 tD3 tF5 tM5
DC6 tD1 tF8 tH2 tJ4
DC7 tD5 tF3 tM3 tM8
DC8 tE1 tE3 ○ ○
DC9 tF1 tG1 tG2 tE2
DC10 tH1 ○ ○ ○
DC11 ○ ○
DC12 tM1 tE5
DC13 tF9 tE6

DC1 ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
DC2 tC3 tL2 tL4 tD4 tF2 tF7 tM4 tM7
DC3 tB1 tB2 tL3 tL5 tL6 tA1 tA2 tM6
DC4 tJ1 tJ2 tJ3 tJ5 tL1 tM9
DC5 tD2 tD3 tF5 tM5
DC6 tD1 tF8 tH2 tJ4
DC7 tD5 tF3 tM3 tM8
DC8 tE1 tE3 tE4 tC1 tC2 tI1 tI2 tI3
DC9 tF1 tG1 tG2 tE2
DC10 tH1 ○ ○ ○
DC11 tM2 tF4 tF6
DC12 tM1 tE5
DC13 tF9 tE6

Figure: Iteration on Max-min Fairness

Real World (max-min iteration)

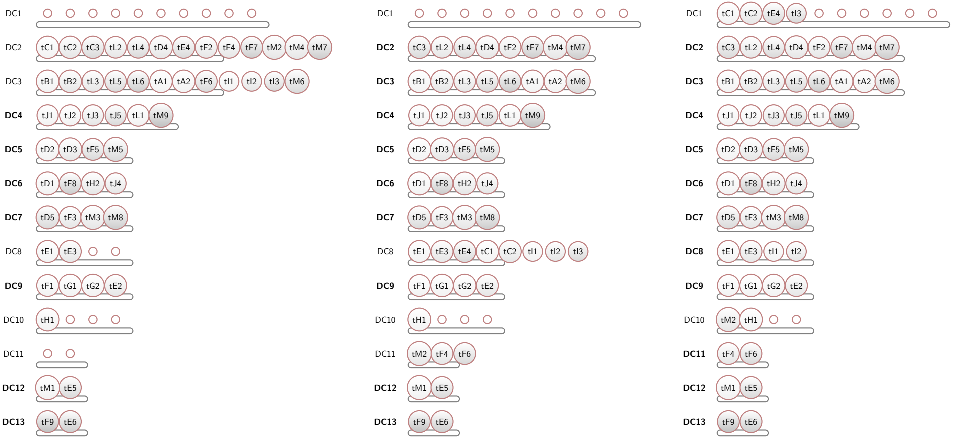


Figure: Iteration on Max-min Fairness

Running Time

The running time is 15s, which indicates that our algorithm is efficient enough. While the algorithm without max-min is 10min.

The less restrictions/variables will reduce the time complexity after each iteration, since the restrictions on slot will not exceed the naive version and the variable on task will get fixed if it is in the available data center.

Shanghai Jiao Tong University

Thank You

Taoran Han, Longxuan Wei, Zilong Li · Stage-Based Strategy for Scheduling Jobs Problem with Max-Min Fairness