Resource Scheduling Problem in Hadoop

Project for Algorithm Design and Analysis

Xun Gong, 021033910012, gongxun@sjtu.edu.cn
Ning Yang, 021033910045, yn937391832@sjtu.edu.cn
Ziyu Wan, 021033910080, alex_wan@sjtu.edu.cn

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

Overview

The problem of multi-core(job) resource scheduling has been studied for a long time. Researchers need to find an optimal schedule that minimize the *makespan* or maximize the average utilization. Though deterministic solutions for such a problem have been proved to be NP-hard, a variety of approximation methods with different properties and approximation ratios have been proposed.

The problem that we try to solve here is an variant in which we need make multi-level decisions on how to combine and allocate data blocks of each job onto different CPU cores of different hosts. In this problem, we cannot naively compute the speedup ratio as the number of cores used when executing a job but bear in our mind that the more cores we used, the more speed decay happens for a single core. Moreover, if we consider scheduling the jobs under multiple hosts settings, we should also take the time of data transmission into consideration.

During designing an efficient and effective algorithm, we've find some properties or limitations of this task. And by leveraging such properties or limitations, we've come up with some simple but very effective ideas which help us design good and practical algorithms finally.

For more convenient and accurate implementation, We refactored the code to the Python, and we use Python as our programming language for this project.

Properties and Limitations

What we want to mention here at first is the design of speed decay for multi-core execution. Given n cores running one job together, the speed decay on each core is defined as

$$g(n) = 1 - \alpha(n-1). \tag{1}$$

We have two key observations about this setting:

(i.) **More cores does not means better speedup ratio.** If we consider of the total speedup ratio of using n cores which is $n(1-\alpha(n-1))$, this is a quadratic function that have its maximum at $\frac{\alpha+1}{2\alpha}$. Take $\alpha = 0.08$ for example, the above statement means that if we can distributed all data blocks of a job perfectly uniformly, we can get our theoretical optimal speedup ratio by using $n = 7$ cores.

(ii.) **Very large number of cores even make negative speedups which is invalid and undefined.** From the definition of speed decay, we can find that if we the core number is too large, $n > \frac{\alpha+1}{\alpha}$, we will get invalid speed since the decay factor becomes negative. This means we only need to consider at most a certain number of cores no matter how many cores there are when trying to run a job in parallel.

Considering the above properties, we can easily find ways to accelerate and improve our algorithms.

Resource Scheduling for Single Host

Problem formulation

Variables: If index is not specified, we use i is for job, j is for core, k is for data block. Except those defined in the documentation, we define some new symbols as below:

$\omega \geq 0$ is the time when all jobs are done which is a positive real number.
$x_{i,j,k} \in [0,1]$is binary indicator of whether block k of job i allocated to core j
$y_j^i \in [0,1]$ is binary indicator of whether some block of job i allocated to core j.
$sp_i > 0$ is the speed of job i.

Constraints: To satisfy the requirements, we should add the following constraints:

a.) The time when all jobs are done must be greater than or equal to all jobs' finish time.

$$\omega \geq tf(job_i), \forall job_i \tag{2}$$

b.) The finish time of a job must be greater than or equal to the finish time of all cores for that job.

$$t_i + tp_j^i = tf_j^i \leq tf(job_i), \forall job_i, c_j \tag{3}$$

c.) If two jobs share CPU cores, they must be executed in a strict order, namely one must be executed before/after the other one on that cpu core.

$$y_j^{i_1} y_j^{i_2} (t_{i_1} - t_{i_2})(tf_j^{i_1} - t_{i_2}) \geq 0, \quad \forall job_{i_1} \neq job_{i_2}, c_j, \tag{4}$$

d.) y should be harmonic with x.

$$y_j^i \geq x_{i,j,k} \quad \forall job_i, c_j, b_k^i \tag{5}$$

e.) all data blocks must be allocated to exactly one CPU core.

$$\sum_{c_j} x_{i,j,k} = 1. \quad \forall job_i, b_k^i \tag{6}$$

f.) The processing time of a job i on core j is must be the sum of data blocks allocated to it divides its speed.

$$\sum_{b_k^i} (x_{i,j,k} \cdot size(b_k^i)) = tp_j^i \cdot sp_i \quad \forall c_j, job_i \tag{7}$$

g.) The speed of each job must satisfy the project's requirements.

$$s_i \cdot (1 - \alpha \cdot (\sum_{c_j} y_j^i - 1)) \leq sp_i \tag{8}$$

Objective we use the objective in the documentation to find a schedule that minimize the finish time of the last job, i.e.

$$\min \omega \tag{9}$$

Solution

Here we discuss our solution.

Method We propose to use greedy algorithms to schedule resource allocation. And we've tried some alternatives.

1. Use only a single core for each job, then the problem becomes a Job Scheduling problem and we can use a sequential algorithm with approx. ratio about $\frac{4}{3}$.
2. Sort all jobs to make their time of using single core in non-increasing order. Then each time pick a job and try to allocate as much cores as possible.
3. Sort all jobs to make their time of using single core in non-increasing order. Set the maximal number of cores a job can use, M. Each time pick a job i in the sorted list and then compare the bubble time of allocating it to M cores and the bubble time of allocating i and $(i+1)$ to M cores(which is M possible cases), if the former is smaller, choose the one with smaller bubble time and allocate next jobs.
4. Enumerate the maximum number of jobs in parallel, m_j. Sort all jobs to make their time of using single core in non-increasing order in a queue. Each time pop the first m_j jobs and allocate these jobs on almost equivalent number of cores. Then pick the optimal m_j.

Algorithm 1: Greedy Schedule on Single Host

Input: $\{c_j\}_j$, $\{jobs_i\}_i$, $\{b_k^i\}_k^i$, $\{s_i\}_i$
Output: A schedule of resource allocation

```
1  SC_OPT=None, T_OPT=∞;
2  for job ∈ J with index i do
3      Sort the blocks of job i decreasingly according to their data size;
4      for j = 1 to |C| do
5          compute EndTime[i][j] as the shortest time using j cores for job i with sequential algorithm;
6      end
7  end
8  Sort J according to their total data block size divides their speed in non-increasing order;
9  for np = 1 to |C| do
10     for job ∈ J with index i do
11         compute nc as used for this job as #core = min[arg min_j EndTime[i][j] , floor((|C|+i mod np)/np)];
12         pick nc cores with minimal finish time as U;
13         for b_k^i ∈ B^i in a non-increasing order do
14             find the core c ∈ U with minimal finish time;
15             put b_k^i on c;
16         end
17     end
18     if Current schedule's finish time is better then T_OPT then
19         SC_OPT= current schedule;
20         T_OPT = current schedule's finish time;
21 end
   Output: SC_OPT, T_OPT
22 ;
```

Table 1. Time Complexity for single-host algorithm

code lines	Time Complexity
line 3	$O(\|J\|\|B\|\log\|B\|)$
line 5	$O(\|J\|\|C\|\|B\|\log\|B\|)$
line 8	$O(\|J\|\|B\|+\|J\|\log\|J\|)$
line 12	$O(\|J\|\|C\|^2\log\|C\|)$
line 14	$O(\|J\|\|C\|^2\|B\|)$

Pseudocode In our experiments generated under multiple hyper-parameters, we found that the last algorithm has best performance in average. So we only show its pseudocode 1 and analyze it.

Complexity Analysis The time complexity is analyzed in Table.1

Case Study We present the visualization results of some test cases here and analyze our algorithm's behavior. Fig.1 shows the execution diagram of solution for given test case for single-host scheduling problem. In our optimal solution, almost all jobs are executed on a single core and such schedule brings a tiny amount of bubble. With up to 88.72% utilization, we can come to the conclusion that our algorithm has good enough performance.

Moreover, we also present the visualization result for a larger test case generated randomly. As is shown in Fig.2, all data blocks have been tightly arranged as a rectangle resulting in few amount of bubbles which means our algorithm find an efficient schedule for this test case. And the 98.7% utilization means not only our algorithm is powerful but also the metrics of utilization is not very reasonable as it should take all the bubbles into consideration.

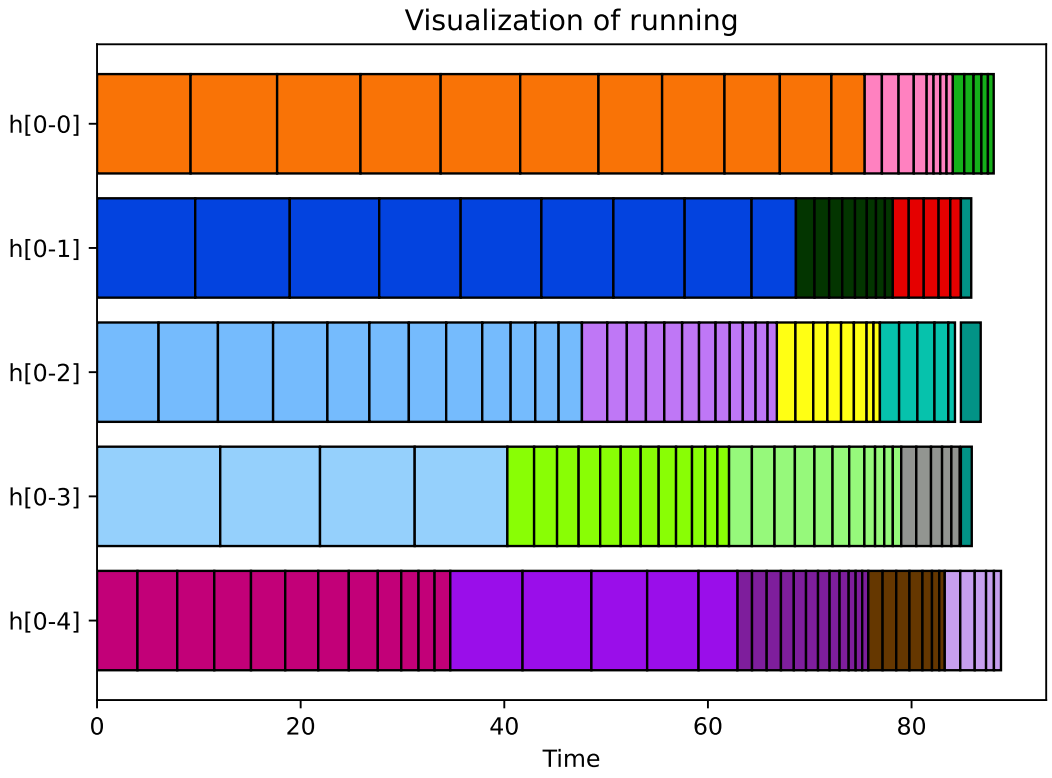

Fig. 1. Visualization of test case for single host scheduling problem

Resource Scheduling for Multiple Hosts

Problem formulation

Variables: If index is not specified, we use i is for job, j is for core, k is for data block. Except those defined in the documentation, we define some new symbols as below:
$\omega \geq 0$ is the time when all jobs are done which is a positive real number.
$z_i \geq 0$ is the finish time of job i, which is a positive real number.
$x_{i,j,k,l} \in [0,1]$is a binary indicator of whether block k of job i allocated to core j on host l
$y^i_{j,l} \in [0,1]$ is a binary indicator of whether some block of job i allocated to core j on host l.
$sp_i > 0$ is the speed of job i.
$ts_{i,j,l} \geq 0$ is the time of transmission of job i to core j on host l.
$loc^i_{k,l} \in [0,1]$ is a binary indicator of whether the initial location of block k of job i is host l.

Constraints: To satisfy the requirements, we should add the following constraints:

a.) The time when all jobs are done must be greater than or equal to all jobs' finish time.

$$\omega \geq tf(job_i), \quad \forall job_i \tag{10}$$

b.) The finish time of a job must be greater than or equal to the finish time of all cores on all hosts for that job.

$$t_i + tp^i_{l,j} = tf^k_{l,j} \leq tf(job_i) \tag{11}$$

c.) If two jobs share CPU cores, they must be executed in a strict order, namely one must be executed before/after the other one on that CPU core.

$$y^{i_1}_{j,l} y^{i_2}_{j,l} (t_{i_1} - t_{i_2})(tf^{i_1}_{j,l} - t_{i_2}) \geq 0, \quad \forall job_{i_1} \neq job_{i_2}, c_j, h_l \tag{12}$$

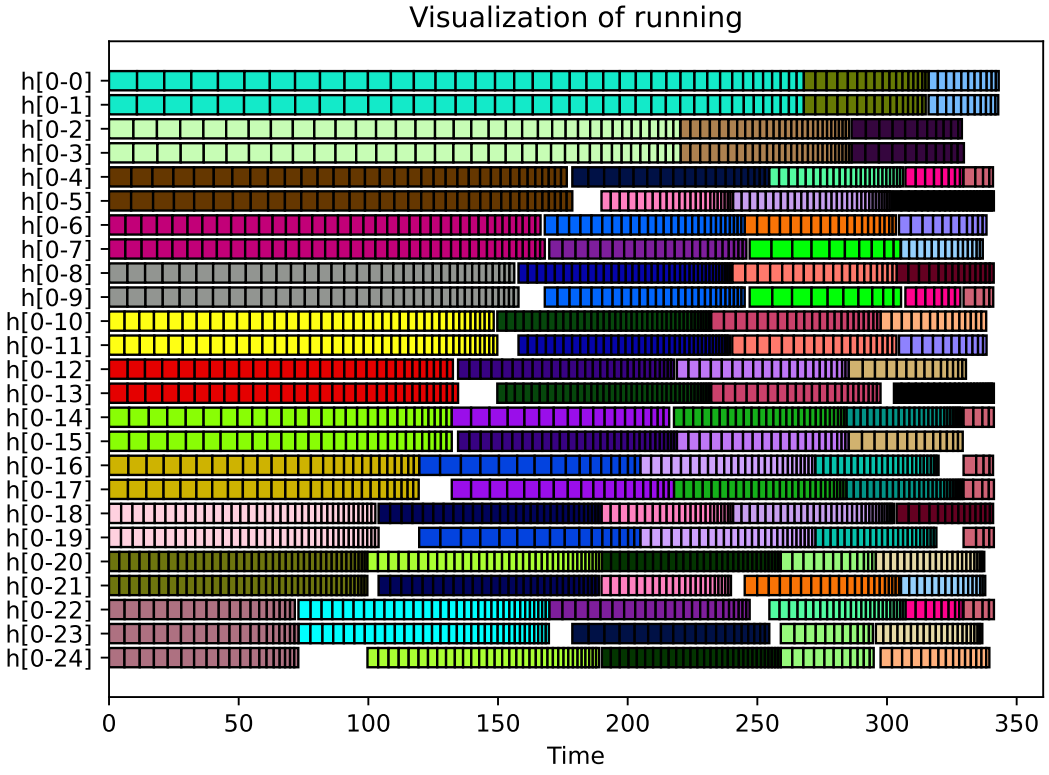

Fig. 2. Visualization of randomly generated test case for single host scheduling problem

d.) y should be harmonic with x.

$$y^i_{j,l} \geq x_{i,j,k,l}, \quad \forall job_i, c^l_j, b^i_k, h_l \tag{13}$$

e.) all data blocks must be allocated to exactly one CPU core

$$\sum_{c_j, h_l} x_{i,j,k,l} = 1, \quad \forall job_i, b^i_k \tag{14}$$

f.) The processing time of a job i on core j must be the sum of data blocks allocated to it divides its speed plus the time of transmission.

$$\sum_{b^i_k} (x_{i,j,k,l} \cdot size(b^i_k))/sp_i + ts_{i,j,l} = tp^i_{j,l}, \quad job_i, c^l_j, h_l \tag{15}$$

g.) The speed of each job must satisfy the project's requirements.

$$0 < s_i \cdot (1 - \alpha \cdot (\sum_{c_j, h_l} y^i_{j,l} - 1)) \leq sp_i \tag{16}$$

h.) The time of transmission must satisfy the project's requirements.

$$\sum_{b^i_k} \left[x_{i,j,k,l} \cdot (x_{i,j,k,l} - loc^i_{k,l}) \cdot size(b^i_k) \right] = ts_{i,j,l} \cdot s_t, \quad \forall job_i, h_l, c^l_j \tag{17}$$

Objective we use the objective in the documentation to find a schedule that minimize the finish time of the last job, i.e.

$$\min \omega \tag{18}$$

Solution

Here we discuss our solution.

Method We propose to use greedy algorithms to schedule resource allocation. And we've tried some alternatives.

It is worth noting that there is a feature where the transfer speed is much faster than the calculation speed of core for a given data range. So in most cases the benefits of transmission outweigh the costs of parallelism and transmission. To simplify the problem, the parallel benefits will be prioritized in the following algorithm.

1. Similar to Task 1, just consider use a single core fore each job, then the transmission time is almost fixed, which become an approximation of Job Scheduling of Single Host,
2. In this algorithm, we use two tricks: first, we want to consider all the core at one host, and add corresponding transmission time on each block as running time, then arrange them to core with the earliest finish time. Second, because of the existence of the computing decaying coefficient $g(\cdot)$ &, use core as many as possible may not get the optimal result. So we set a limitation for maximum number of parallel cores for a job, then explore this limitation for best result. A brief description as follow:
   Sort all the jobs by its sum of block size in non-increasing order, then consider all cores at one host, for each job, do a sorting to find which cores will be chosen to handle this job according to finish time and transmission time, then run a greedy algorithm to schedule jobs' blocks on these cores, at last update the parameters. This process will repeat for different maximum core limitation, and pick the optimal result.
   Intuitively speaking, this algorithm has considered the transmission time between host, whose result will be better than hadn't. But in the task2, we get better result without the consideration of transmission time. This phenomenon is counter-intuitive. A possible explanation is that transmission speed in this task is far more faster than calculation speed of core, so compare with saving transmission time, doing transmission to the core with earlier finish time will get better effect.

Pseudocode In our experiments generated under multiple hyper-parameters, we found that the last algorithm has best performance in average. So we only show its pseudocode 2 and analyze it.

Algorithm 2: Greedy Schedule on Multiple Host

Input: $\{c_j\}_j$, $\{jobs_i\}_i$, $\{b_k^i\}_k^i$, $\{s_i\}_i$
Output: A schedule of resource allocation

SC_OPT=None, T_OPT=∞;
Sort J according to their total data block size divides their speed in non-increasing order;
$C* = \bigcup_{host \in H}$ {cores in h} ;
for *job* $\in J$ *with index* i **do**
    Get D_h^i as datasize for all blocks initialized in host h ;
    Get maximize used #cores = $\min(|B^i|, |C*|)$;
    Pick #-earliest finished cores U to use, across all possible cores by $\arg\min_{top-j}(tf_j - B_h^i/s_t)$;
    $s_i = s_i * (1 - \alpha * (|U| - 1))$;
    Sort the blocks B^i of job i decreasingly according to their data size and transmission penalty $\arg\max(size(B_j^i)/s_i)$;
    for *Block* b_k^i *in* B^i **do**
        Find the core $c \in U$ win minimal finish time ;
        Put b_k^i on c ;
    end
    if *Current schedule's finish time is better then T_OPT* **then**
        SC_OPT= current schedule;
        T_OPT = current schedule's finish time;
end
Output: SC_OPT, T_OPT
;

Complexity Analysis The time complexity is analyzed in Table.2

Table 2. Time Complexity for multi-host algorithm

code lines	Time Complexity
line 2	$O(\|J\|\|B\|\log\|B\|)$
line 7	$O(\|C\|\|B\|\log\|B\|)$
line 9	$O(\|J\|\|B\|\log\|B\|)$
line 10	$O(\|B\|\|C\|^2)$

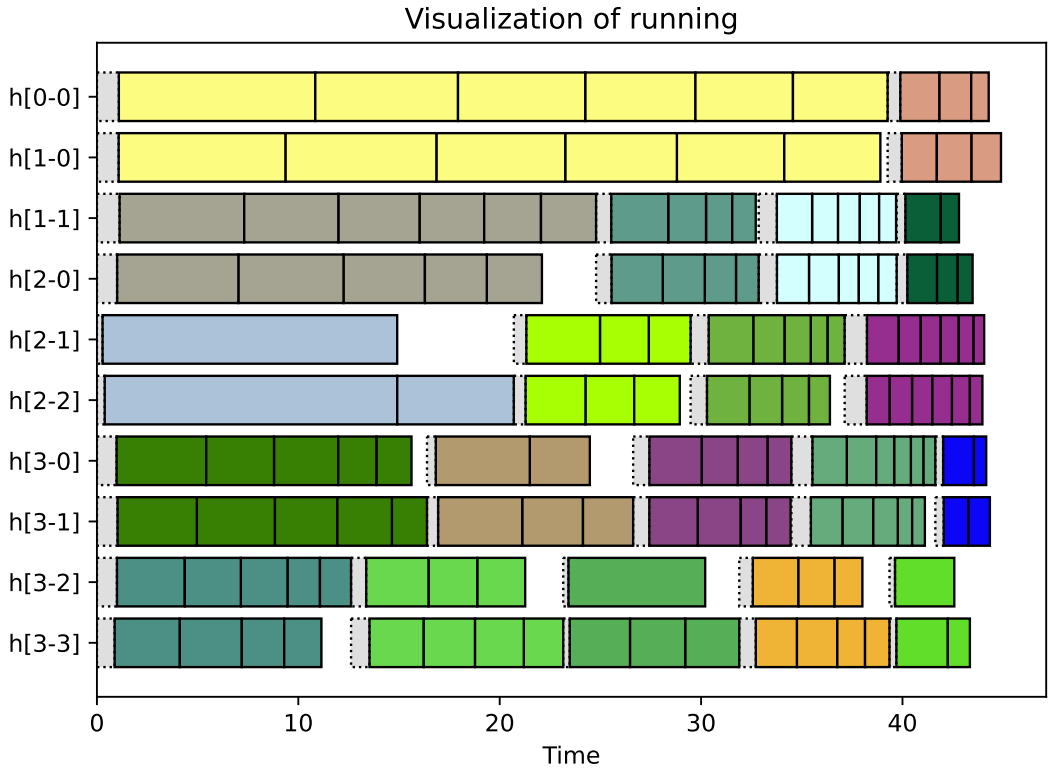

Fig. 3. Visualization of the given test case for multi-host scheduling problem. Grey box is the data transmission.

Case Study We present the visualization results of some test cases here and analyze our algorithm's behavior. Fig.3 shows the execution diagram of solution for given test case for multi-host scheduling problem. With given task2, our finish time is 44.89s, total response time is 670.21s, and our utilization rate can up to 98.1%, we can come to the conclusion that our algorithm has good enough performance.

Moreover, we generate some random case and based on our observation we have some conclusion: in our optimal solution, we enumerate the maximum core number to search better result, and we find that for a ideal arbitrarily divisible job, it uses 6 or 7 cores may get best result for itself; for the whole task, most job using 2-5 cores will get the global optimal result. Here we give a visualization result for a larger random-generated test case to prove this, as is shown in Fig.4. The core used for each block is usually at most 4, and the algorithm try to spilt equally with blocksize to get the theoretically optimal situation, so we can see that there are hardly any bubble time in cores, and the utilization rate is 97.46%, which shows that our algorithm has good expandability and generalizability. However, this algorithm did not fully consider the optimization of mixed quantity core, so it's still not optimal for sure, but it still a algorithm very close to real optimal solution.

Acknowledgements

We thank for the teachers, TAs and students who give great support in completing this project.

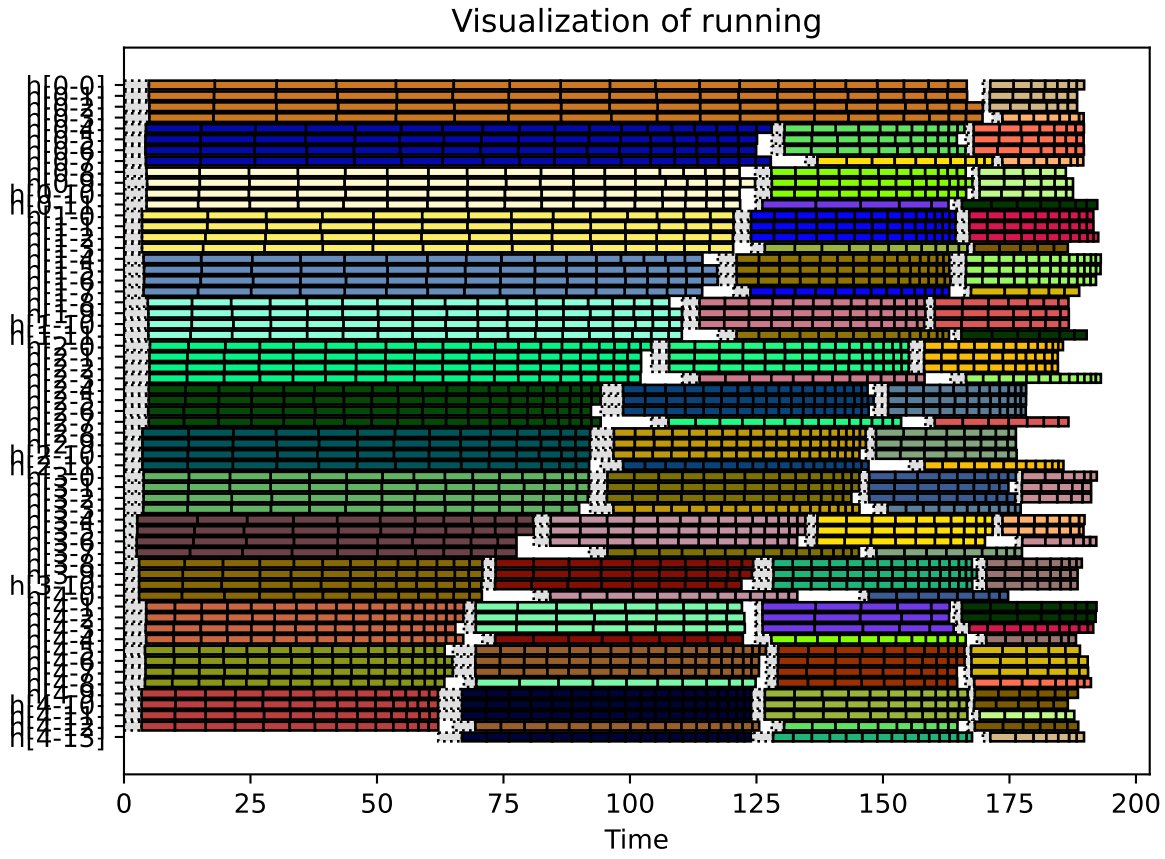

Fig. 4. Visualization of randomly generated test case for multi-host scheduling problem. Grey box is the data transmission.

Appendix

Table 3. Symbols and Definitions

Symbols	Definitions
n	The number of jobs
m	The number of cores
q	The number of hosts
job_i, J	job_i is the i-th job. The job set is $J = \{job_0, \cdots, job_{n-1}\}$.
h_l, H	h_l is the l-th host. The host set is $H = \{h_0, \cdots, h_{q-1}\}$.
m_l	The number of cores on host h_l
c_j^l, C_l	c_j^l is the j-th core on host h_l. C_l is the set of cores on host h_l.
C	The set of cores. $C = \{c_0, \cdots, c_{m-1}\}$ for single-host. $C = \cup_{l=0}^{q-1} C_l$ for multi-host.
b_k^i	The block of job_i whose id is k
B_j^i	The set of data blocks of job_i allocated to core c_j
B^i	The set of data blocks of job_i
B_{lj}^i	The set of data blocks of job_i allocated to core c_j^l
B	The set of blocks of all jobs, $B = \bigcap_i B^i$
$\widetilde{B}_{lj}^i$	The set of data blocks of job_i allocated to core c_j^l but not initially stored on h_l
$size(\cdot)$	The size function of data block
$g(\cdot)$	The computing decaying coefficient caused by multi-core effect
s_i	The computing speed of job_i by a single core
s_t	The transmission speed of data
e_i	The number of cores processing job_i
t_i	The time to start processing job_i
tp_j^i, tf_j^i	The processing time / finishing time of core c_j for job_i
tp_{lj}^i, tf_{lj}^i	The processing time / finishing time of core c_j^l for job_i
$tf(job_i)$	The finishing time of job_i
ω	The time when all jobs are done, which is a positive real number.
z_i	The finish time of job i, which is a positive real number.
$x_{i,j,k}$	The binary indicator of whether block k of job i allocated to core j
y_j^i	The binary indicator of whether some block of job i allocated to core j.