Greedy

To solve that problem, we first want to put forward a naive algorithm: a simple greedy algorithm to give a scheduling plan.

Our idea is really simple. To ensure that the completion time of all tasks is minimized, we should guarantee that all the available slots in every data center remain busy. So for each slot, we set its two states: busy state and idle state. When a slot is in the idle state, we let it recursively find tasks that can be performed.

This involves a scheduling problem. When a slot has multiple optional tasks to choose from, and when multiple slots select the same task at the same time, we should determine the order of scheduling in some way. We greedily use the SJF(Short Job First) algorithm for scheduling. When an idle slot has multiple tasks to choose from, it should calculate the sum of the transmission time and execution time of all these tasks to be selected, and choose the task with the shortest total time. In addition, if there are multiple slots competing for the same task at the same time, the task will be assigned to the slot with the shortest transmission time.

Here we can use multithreading to implement this algorithm to improve efficiency. Specifically, we need to assign a thread to each available slot to execute the above algorithm. Whenever a task is executed, the thread selects the thread with the shortest time from the current ready queue to execute next. This involves synchronization between threads, and mutual exclusion needs to be achieved by adding locks.

Although the greedy algorithm can indeed give us a solution, does this solution give the optimal solution? The answer is negative. Such an algorithm only pays attention to the operation of each slot, and does not perform scheduling from an overall perspective. It is easy for us to think of the following situation: A certain task is suitable for execution in a certain slot, but this task is still running at some time. But at this time, this slot can find a troublesome other task.