P4: Task Scheduling in Edge Server Systems with Limited Solar Energy and Infinite Batteries

Harsh Katara(210101045), Anand Keshav(210101014), Yash Raj Singh(210101113)

April 19, 2024

1 Introduction

Edge computing has become increasingly popular due to its ability to bring computation and data storage closer to the location where it is needed, reducing latency and bandwidth usage. However, edge servers often rely on renewable energy sources like solar power, which can be intermittent and unpredictable. In this report, we address the problem of task scheduling in edge server systems with limited solar energy and infinite batteries.

2 Problem Description

We are given a set of edge servers, each equipped with a solar panel and an infinite battery. The solar panels generate energy, which is stored in the batteries for later use. However, the energy generated by the solar panels is limited, and the batteries have a finite capacity. The goal is to maximize the number of tasks executed by the edge servers while ensuring that the total energy consumption does not exceed the available energy budget.

2.1 Constraints

1. **Solar Power Generation:** Each edge server is equipped with a solar panel with a limited capacity S. The power generated at each time slot must not exceed this capacity. This means at most $(T \cdot S)^{\frac{1}{3}}$ many tasks can be executed.
2. **Task Arrival:** Tasks arrive at each edge server at different time slots throughout the day, denoted by $D[i][j]$. These tasks must be executed within the same time slot they arrive.
3. **Battery Capacity:** Edge servers have an infinite-capacity battery for storing excess solar power. However, the stored power can only be utilized within the same edge server and cannot be transferred between servers.
4. **Power Consumption:** The amount of power consumed at an edge server in a time slot is proportional to the cube of the number of tasks executed in that time slot. This relationship must be taken into account when determining task execution strategies.
5. **Task Migration:** Tasks can be migrated between edge servers without incurring any additional cost. However, battery power cannot be transferred between edge servers.

3 Solution Approach

We have employed a Greedy approach to maximize the number of tasks executed within the given constraints. Our solution involves iteratively assigning tasks to servers and timestamps in order to distribute tasks as uniformly as possible across servers and timestamps. We maintain two queues, 'node1' and 'node2', to keep track of servers for which task assignment is possible in the current and next iterations, respectively (we are defining node as combination of server and timestamp).

3.1 Helper Functions

We utilize the following helper functions in our program:

- `binpow`: Implements binary exponentiation to efficiently calculate exponents.
- `is_possible`: Determines whether it's feasible to assign a task to a server based on its current assignment and available energy.
- write_schedule_to_csv: This function writes the task assignments from a 2D vector to a CSV file named "schedule.csv". It iterates over the vector, formatting the entries and separating them with commas. And this file is used in visualization of the task schedule.
- `find_value`: This function plays a crucial role in the task allocation process by determining the sequence in which tasks are assigned to servers at a particular timestamp during each iteration. It calculates a parameter that influences the decision-making process for task assignment, ensuring efficient utilization of resources.

3.2 Algorithm

Initialisation

- Initialize PostUsed1 to identify bottlenecks in the future and PostUsed2 to store the sum of current energy waste and the energy required to schedule the maximum energy-consuming job in the future timestamp at the current server.
- Initialize Nodes1 as a queue for storing currently assignable servers and timestamps and Nodes2 as a queue for storing future assignable servers and timestamps.

In each iteration

- Iterate through all timestamps and machines.
- Assign at most one job in each timestamp to each machine according to the algorithm's criteria if possible.
- Sort the order of machines for each timestamp based on the value calculated by (PostUsed2 - energy required to assign one more task). This sorting operation has a time complexity of O(MLogM), where M is the number of machines.
- Update PostUsed1, PostUsed2, Nodes1, and Nodes2 for all machines where job assignment has taken place.
- If no jobs were assigned in this iteration, stop executing.

3.3 Optimizations

We have implemented several optimizations to reduce the time complexity of the program:

- We identify the bottleneck server in each timestamp, i.e., the server with the minimum available energy, and use that energy for task assignment.
- We use a set ('tub_ordered') to store servers sorted by their available energy, facilitating efficient determination of the order in which tasks are assigned.

4 Time Complexity Analysis

At first, we iterate through levels, which can be at maximum $(S \times T)^{\frac{1}{3}}$ times. For each iteration, we traverse through the potential nodes where tasks can be assigned, which can be at most $M \times T$. Within each time slot in these $M \times T$ nodes, we perform the following operations(when we go satisfies if condition i.e. T times):

- Update the array `post_used2`: $O(M \times T)$ times
- Get the order of tasks to be assigned to potential nodes: $O(M \log M)$ (sorting operation)

So, in each iteration of level, the time complexity is $O(M \times T + T \times (M \times T + M \log M))$.

Therefore, the overall time complexity for the entire algorithm is given by:

$$O\left((S \times T)^{\frac{1}{3}} \times (M \times T + T \times (M \times T + M \log M))\right)$$

5 Test Cases

We have tested our implementation on the following test cases:

1. **Test Case 1**:
   - Number of Servers: 2
   - Number of Timestamps: 2
   - Upper Bound of Energy (S): 30
   - Charging Capacities:
     $$\begin{matrix} 8 & 0 \\ 0 & 1 \end{matrix}$$
   - Tasks Arriving:
     $$\begin{matrix} 2 & 0 \\ 0 & 1 \end{matrix}$$

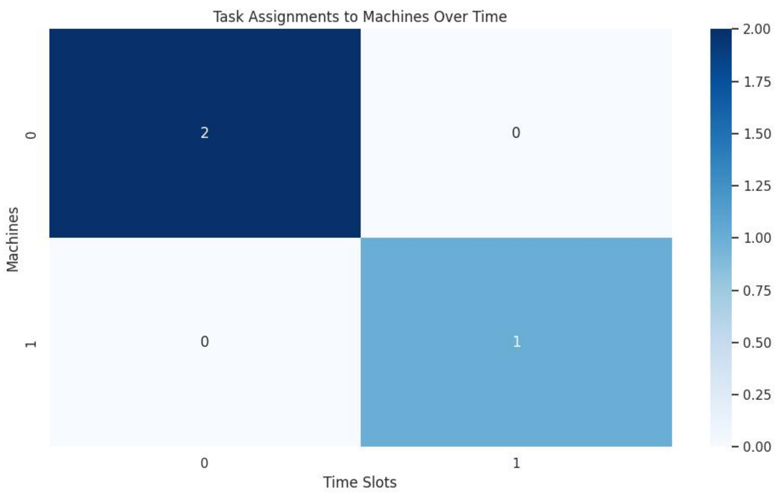

Figure 1: Output1

   - Jobs Scheduled: 3
   - Energy Used: 9

2. **Test Case 2**:
   - Number of Servers: 3
   - Number of Timestamps: 3
   - Upper Bound of Energy (S): 130
   - Charging Capacities:
     $$\begin{matrix} 16 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{matrix}$$
   - Tasks Arriving:
     $$\begin{matrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 1 \end{matrix}$$

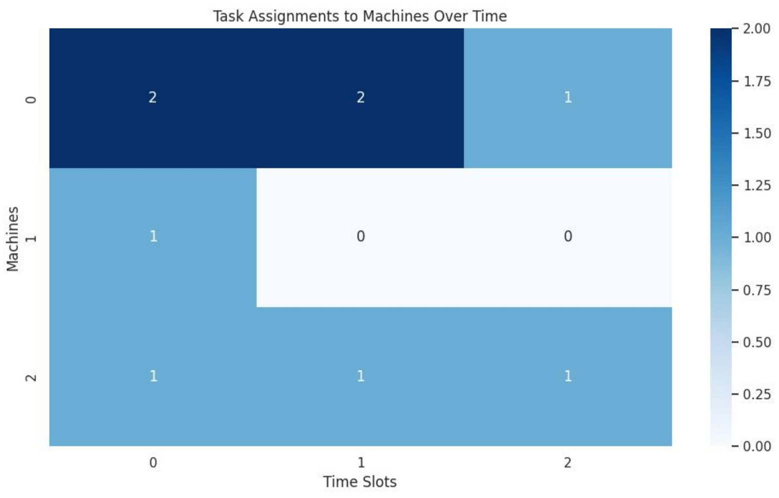

Figure 2: Output2

- Jobs Scheduled: 9
- Energy Used: 21

3. **Test Case 3**:
   - Number of Servers: 2
   - Number of Timestamps: 2
   - Upper Bound of Energy (S): 72
   - Charging Capacities:
$$\begin{matrix} 27 & 0 \\ 5 & 0 \end{matrix}$$
   - Tasks Arriving:
$$\begin{matrix} 2 & 3 \\ 3 & 2 \end{matrix}$$

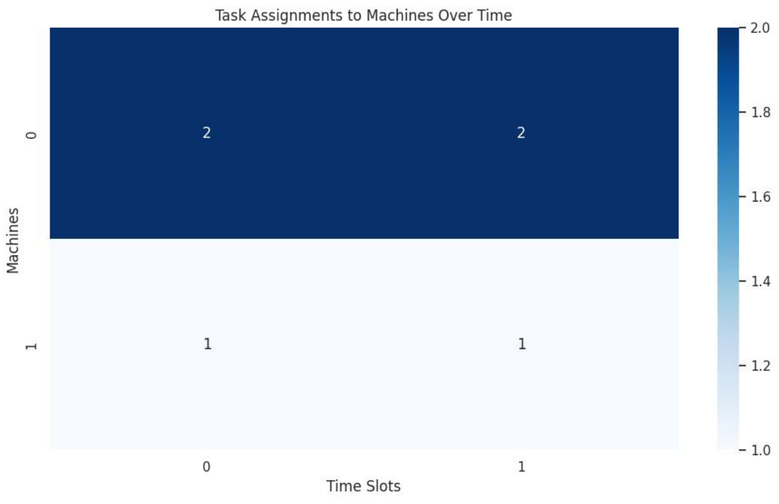

Figure 3: Output3

6 Conclusion

In conclusion, our Greedy algorithm efficiently addresses the task scheduling problem in edge server systems with limited solar energy and infinite batteries. By optimizing task distribution while adhering to energy constraints, we achieve effective utilization of resources. The provided test cases validate the effectiveness and efficiency of our solution.