Heterogeneous GPU Scheduling

I. Formal Problem Description: Simplified Heterogeneous GPU Scheduling

A. Sets and Indices

- $\mathcal{T} = \{1, 2, \ldots, n\}$: Set of tasks .
- $\mathcal{G} = \{1, 2, \ldots, m\}$: Set of available GPUs in the cluster.
- $i, k \in \mathcal{T}$: Indices for tasks.
- $j \in \mathcal{G}$: Index for GPUs.

B. Parameters

Task Model (from Dataset):

- L_i: Workload of task i (Floating point operations or relative units).
- m_i: Memory demand of task i (GB).
- d_i: Deadline of task i (Time unit).
- w_i: Weight/Priority of task i.
- a_i: Arrival time of task i.

Resource Model (GPU Cluster):

- v_j: Computing speed of GPU j (Workload per time unit).
- M_j: Memory capacity of GPU j (GB).

Derived Parameter:

- $p_{ij} = \frac{L_i}{v_j}$: Execution time of task i if assigned to GPU j.

C. Decision Variables

- $x_{ij} \in \{0, 1\}$: Binary variable. equals 1 if task i is assigned to GPU j, 0 otherwise.
- $s_i \geq 0$: Start time of task i.
- $c_i \geq 0$: Completion time of task i.
- $y_{ik} \in \{0, 1\}$: Binary sequence variable (for tasks on the same GPU). Equals 1 if task i precedes task k, 0 otherwise.

D. Mathematical Formulation

*a) **Objective Function**:* Minimize the Total Weighted Tardiness. This balances the priority (w_i) and the urgency (meeting d_i).

$$\text{Minimize } Z = \sum_{i \in \mathcal{T}} w_i \cdot \max(0, c_i - d_i) \tag{1}$$

b) Constraints: 1. Assignment Constraint: Each task must be assigned to exactly one GPU.

$$\sum_{j \in \mathcal{G}} x_{ij} = 1, \quad \forall i \in \mathcal{T} \tag{2}$$

2. Memory Constraint: A task can only be assigned to a GPU if the GPU's memory capacity is sufficient.

$$x_{ij} \cdot m_i \leq M_j, \quad \forall i \in \mathcal{T}, \forall j \in \mathcal{G} \tag{3}$$

3. Timing Constraints: The completion time is the start time plus the execution time on the assigned GPU.

$$c_i = s_i + \sum_{j \in \mathcal{G}} x_{ij} p_{ij}, \quad \forall i \in \mathcal{T} \tag{4}$$

A task cannot start before its arrival time.

$$s_i \geq a_i, \quad \forall i \in \mathcal{T} \tag{5}$$

4. Non-overlapping Constraint (Disjunctive): If two tasks i and k are assigned to the same GPU, they cannot overlap in time. (H is a sufficiently large positive number).

$$\begin{aligned} s_i + \sum_{j \in \mathcal{G}} x_{ij} p_{ij} &\leq s_k + H(3 - x_{ij} - x_{kj} - y_{ik}) \\ s_k + \sum_{j \in \mathcal{G}} x_{kj} p_{kj} &\leq s_i + H(3 - x_{ij} - x_{kj} - (1 - y_{ik})) \end{aligned} \tag{6}$$

This ensures that if both are on GPU j, either i finishes before k starts or k finishes before i starts.

E. Performance Metrics

Based on the decision variables and parameters defined above, the following metrics are used to evaluate the scheduling performance:

1) Total Weighted Completion Time (TWCT): This metric represents the overall value-weighted responsiveness of the system. Lower is better.

$$\text{TWCT} = \sum_{i \in \mathcal{T}} w_i \cdot (c_i - a_i) \tag{7}$$

2) Average Completion Time (ACT): The average time at which tasks finish execution. Lower is better.

$$\text{ACT} = \frac{1}{n} \sum_{i \in \mathcal{T}} (c_i - a_i) \tag{8}$$

3) Deadline Miss Count (DMC): The total number of tasks that failed to complete before their deadline. Lower is better. Let $\mathbb{I}(\cdot)$ be an indicator function that equals 1 if the condition is true, and 0 otherwise.

$$\text{DMC} = \sum_{i \in \mathcal{T}} \mathbb{I}(C_i > d_i) \tag{9}$$

4) Deadline Miss Rate (DMR): The proportion of tasks that failed to complete before their deadline. Lower is better.

$$\text{DMR} = \frac{1}{n} \sum_{i \in \mathcal{T}} \mathbb{I}(C_i > d_i) \tag{10}$$

5) Average GPU Utilization (η): The ratio of the total effective processing time to the total active time of the cluster. Higher is better. First, we define the **Makespan** ($C_{\max}$), which is the completion time of the last task in the system:

$$C_{\max} = \max_{i \in \mathcal{T}}(C_i) \tag{11}$$

The utilization is calculated as:

$$\eta = \frac{\sum_{i \in \mathcal{T}} \sum_{j \in \mathcal{G}} x_{ij} \cdot p_{ij}}{m \cdot (C_{\max} - \min_{k \in \mathcal{T}} a_k)} \tag{12}$$

Where m is the total number of GPUs, and the denominator represents the total GPU-time available during the active scheduling window.

II. Complexity Analysis

The Heterogeneous GPU Scheduling Problem (HGSP) with the objective of minimizing Total Weighted Tardiness is $\mathcal{NP}$-hard.

We prove the $\mathcal{NP}$-hardness of the HGSP by **restriction**. We show that a special case of our general problem reduces to the *Single Machine Total Weighted Tardiness Problem* ($1||\sum w_j T_j$), which is a known $\mathcal{NP}$-hard problem.

Consider a restricted instance of the HGSP with the following constraints:

1) **Single GPU:** The cluster consists of only one GPU ($m = 1$).
2) **Infinite Memory:** The memory capacity of the GPU is sufficiently large ($M_1 \to \infty$), or task memory demands are zero ($m_i = 0$), effectively removing the memory constraint.
3) **Simultaneous Arrival:** All tasks arrive at time zero ($a_i = 0, \forall i \in \mathcal{T}$).
4) **Homogeneous Processing:** Since there is only one GPU, the processing time p_{ij} simplifies to a fixed processing time p_i for each task.

Under these restrictions, the decision variables reduce to finding a permutation (sequence) of tasks. The objective function remains:

$$\text{Minimize} \sum_{i \in \mathcal{T}} w_i \cdot \max(0, C_i - d_i) \tag{13}$$

where C_i is determined solely by the sum of processing times of tasks preceding i in the sequence plus p_i.

This restricted problem is exactly the **Single Machine Total Weighted Tardiness Problem**, denoted as $1||\sum w_j T_j$ in Graham's notation. It has been proven by Lawler (1977) and Lenstra et al. (1977) that $1||\sum w_j T_j$ is $\mathcal{NP}$-hard in the strong sense.

Since a special case of the HGSP is $\mathcal{NP}$-hard, the general HGSP (which adds multiple heterogeneous machines, memory constraints, and release times) is at least as hard as this special case. Therefore, the HGSP is $\mathcal{NP}$-hard.

III. Algorithm Design

Given the online nature of the problem, where tasks arrive dynamically (a_i) and full knowledge of future workloads is unavailable, we adopt a dynamic scheduling approach. The scheduler processes tasks based on their arrival order and assigns them to the heterogeneous GPU cluster using different strategies.

We propose three algorithms to schedule tasks to the Heterogeneous GPU cluster: a baseline First-In-First-Out (FIFO) approach, an improved Greedy strategy, and a meta-heuristic Simulated Annealing with Greedy assignment (SAGreedy) algorithm.

A. Baseline Method: FIFO

The First-In-First-Out (FIFO) scheduler serves as a simple baseline that processes tasks strictly in their arrival order. This approach is straightforward and provides a reference for comparing more sophisticated algorithms.

1) Algorithm Logic: At any scheduling decision point:

1) **Sort**: Sort all tasks in ascending order of their arrival times a_i.
2) **Assign**: For each task i in order:
   - Identify the set of valid GPUs $\mathcal{G}_{valid} = \{j \in \mathcal{G} \mid M_j \geq m_i\}$.
   - If $\mathcal{G}_{valid} = \emptyset$, skip the task (resource unavailable).
   - Otherwise, assign task i to the GPU $j^* \in \mathcal{G}_{valid}$ that minimizes the start time.

The start time on GPU j is calculated as $s_{ij} = \max(a_i, \text{avail}_j)$, where avail_j is the time GPU j finishes its currently assigned tasks. The completion time is $c_{ij} = s_{ij} + \frac{L_i}{v_j}$.

2) Complexity Analysis: Let N be the number of tasks and M be the number of GPUs. The complexity is $O(N \log N + N \cdot M)$ for sorting and GPU assignment.

B. Improved Heuristic: Greedy (EFT)

The Greedy scheduler improves upon FIFO by considering both arrival time and GPU heterogeneity when making scheduling decisions. It uses an Earliest Finish Time (EFT) strategy that accounts for GPU computing speeds.

1) Algorithm Logic: At any scheduling decision point:

1) **Sort**: Sort all tasks in ascending order of their arrival times a_i.
2) **Assign**: For each task i in order:
   - Identify the set of valid GPUs $\mathcal{G}_{valid} = \{j \in \mathcal{G} \mid M_j \geq m_i\}$.
   - If $\mathcal{G}_{valid} = \emptyset$, skip the task.
   - Otherwise, assign task i to the GPU $j^* \in \mathcal{G}_{valid}$ that minimizes the completion time $c_{ij} = \max(a_i, \text{avail}_j) + \frac{L_i}{v_j}$.

Key Difference from FIFO: While FIFO selects the GPU with the earliest start time, Greedy considers the execution time on different GPUs (varying v_j) and selects the GPU that minimizes the actual completion time.

2) Complexity Analysis: The complexity remains $O(N \log N + N \cdot M)$, but the EFT strategy better exploits GPU heterogeneity for improved performance.

C. Meta-Heuristic: SAGreedy

To address the NP-Hardness of minimizing Total Weighted Tardiness (as proven in Section II), we propose a Simulated Annealing with Greedy GPU assignment (SAGreedy) algorithm. This meta-heuristic explores the global solution space of task permutations while using efficient greedy GPU assignment for evaluation.

1) Algorithm Overview:

1) **Initialization**: Start with the Greedy solution as the initial state.
2) **Temperature Schedule**: Use exponential cooling $T_{k+1} = \alpha \cdot T_k$ where $\alpha \in (0, 1)$.
3) **Neighbor Generation**: Generate new task ordering by prioritizing tardy tasks based on current temperature.
4) **Acceptance Criterion**: Accept worse solutions with probability $e^{-\Delta/T}$ to escape local optima.
5) **GPU Assignment**: For each task ordering, use greedy GPU assignment (EFT) for evaluation.

2) Fitness Function: The fitness function corresponds directly to the minimization objective defined in Eq. (1). For a given task ordering with greedy GPU assignment, the fitness value (to be minimized) is:

$$Fitness = \sum_{i \in \mathcal{T}} w_i \cdot \max(0, c_i - d_i) \tag{14}$$

3) Priority-Based Neighbor Generation: A key innovation in SAGreedy is the temperature-dependent neighbor generation strategy:

- **High Temperature**: Prioritize a larger fraction of tardy tasks, enabling more exploration.
- **Low Temperature**: Prioritize fewer tardy tasks, focusing on exploitation of good solutions.

Let τ be the set of tardy tasks (where $c_i > d_i$). The priority ratio $\rho \in [0.1, 0.8]$ is calculated as:

$$\rho = \min(0.8, \max(0.1, \frac{T - T_{min}}{T_{initial} - T_{min}})) \tag{15}$$

The top $\rho \cdot |\tau|$ tardy tasks are selected and prioritized in the new ordering, while remaining tasks follow arrival-time ordering.

4) Complexity Analysis: Let N be the number of tasks, M be the number of GPUs, K be the maximum iterations, and $T_{initial}$ be the initial temperature.

- **Fitness Evaluation**: For each candidate solution, greedy assignment involves $O(N \cdot M)$ operations.
- **Total Complexity**: $O(K \cdot N \cdot M)$ for the complete annealing process.

Although computationally more expensive than FIFO and Greedy ($O(N \log N + N \cdot M)$), SAGreedy allows for escaping local optima by explicitly considering weighted tardiness in the optimization objective.