Liquid: Intelligent Resource Estimation and Network-Efficient Scheduling for Deep Learning Jobs on Distributed GPU Clusters

Edge System Reading Group @ SEU

Haodong Tian

Dec. 18, 2023

Accurate DL job **resource requirement estimation** and **intelligent resource scheduling** are two main challenges for building efficient DL job running platforms.

- **Resource requirement estimation**
  Job resource requirement specified by users often inaccurate → **over-provisioning** or **performance degradation**.

- **Intelligent resource scheduling**
  Avoiding parameter communication overhead → **network-efficient** scheduling.

- **Law of diminishing marginal utility**
  Exists an appropriate value of the computing resources amount for running a given DL job.
  Beyond that amount, the job execution time is similar, but the extra allocated resource is wasted.

Motivation - Intelligent Resource Scheduling

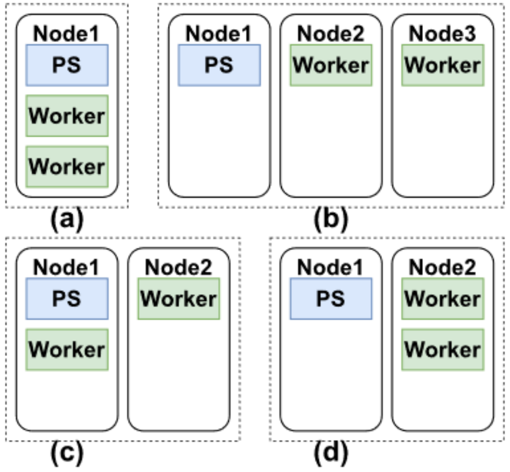
- **Distributed training by Parameter Server (PS)**

  Exchanging large scale parameter gradients with other nodes per iteration.

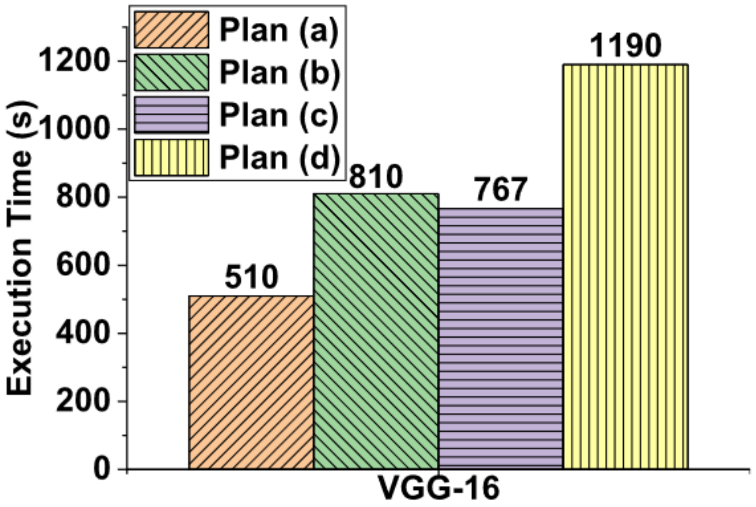
- **Network bandwidth would become bottleneck**

  Workers need to communicate and synchronize with the PS in each epoch.

  Resource scheduling plans have a significant impact on job execution performance.



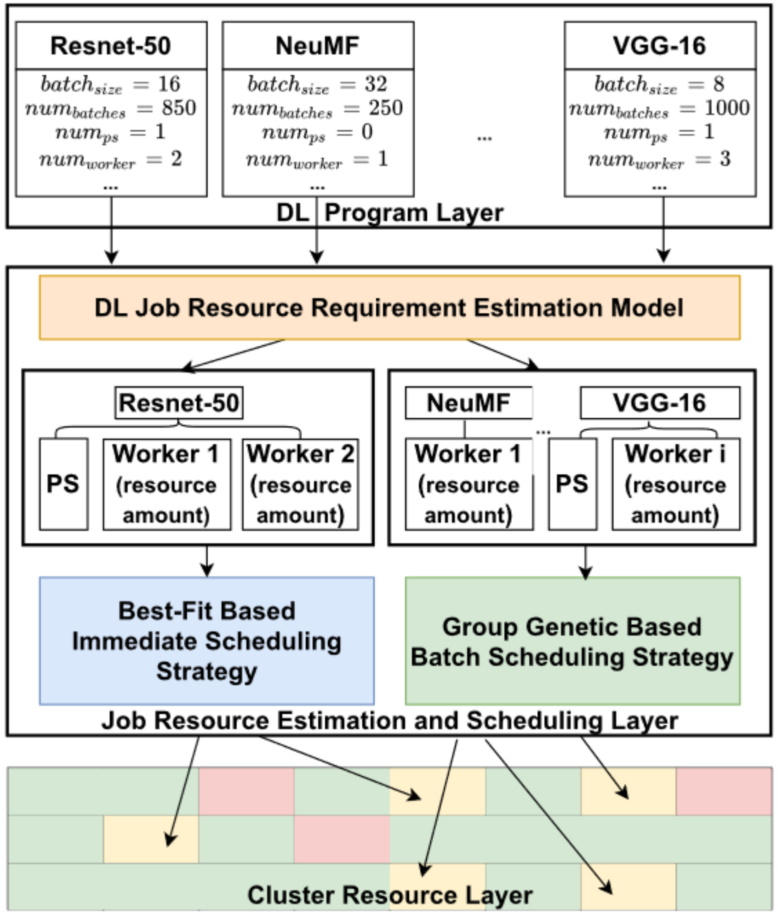
(a) Scheduling plans



(b) Job execution time

System Framework Overview

- DL Program Layer
- **Job Resource Estimation and Scheduling Layer**
  - Job resource requirement estimation model
  - network-efficient scheduling algorithm
- Cluster Resource Layer



DL Job Resource Requirement Estimation

- Feature: job types and key parameters, e.g., $batch\_size$, $num\_batches$, num_ps, $num\_workers$, etc.
- Label: Job Resource Requirement Vector

$$< N \times GPU, GMem, Bandwidth >$$

- Resource Requirement Vector Estimation Model
  - Offline training (Random Forest) $\leftarrow$ DL jobs usually executed multiple times in real world
  - Training data: Randomly pick job parameters and resource allocation, utilizing **Law of diminishing marginal utility** to find the appropriate resource requirement vector

Network-Efficient DL Job Scheduling

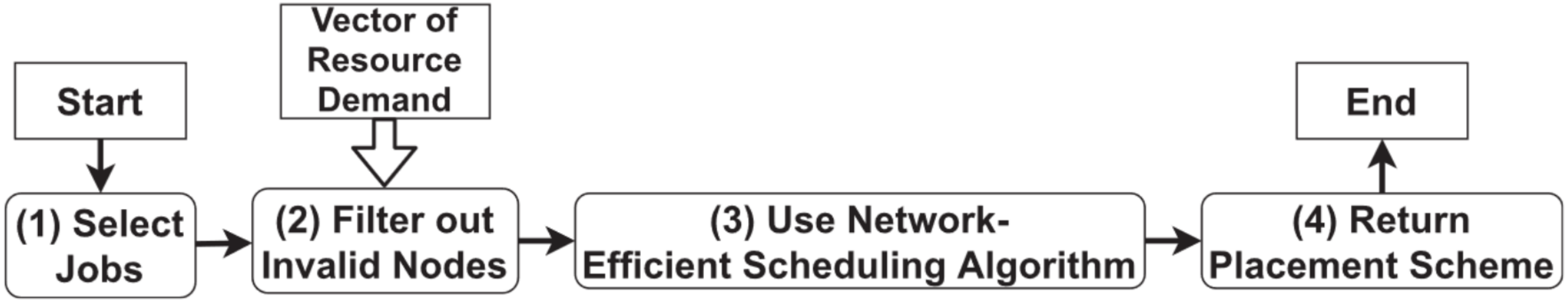
- Find a mapping from containers to idle computing nodes $\leftarrow$ resource/environment isolation by containerization (?)
- Execution time consists of N iterations. Each iteration ccan be divided into training time T_t and synchronization time T_{sy}. Besides, there is a launching time T_l and model saving time T_{sa}.

$$T_e = T_l + N \times (T_t + T_{sy}) + T_{sa}$$

- Prior work finds that T_{sy} dominates the execution time $\rightarrow$ consider network communication bandwidth (among containers) where job instances are located

$$Cost(Node_1, Node_2) = \begin{cases} 0, & same\ node \\ number\ of\ nodes\ in\ the\ rack, & same\ rack \\ number\ of\ nodes\ in\ the\ domain, & cross\ rack \end{cases}$$

Network-Efficient DL Job Scheduling



- Workflow of the network-efficient DL job scheduling
- **Immediate Scheduling Strategy** → DL jobs submitted to cluster one by one with long time intervals
- **Group Genetic Algorithm Based Batch Scheduling Strategy** → DL jobs come quickly and can be grouped in batch for scheduling

- Purpose of scheduling of job i:

$$\min_{\kappa \in \alpha_i} Score_\kappa$$

$$Score_\kappa = \lambda \times Cost_\kappa + (1 - \lambda) \times \sum_{j \in \beta_\kappa} Fitness_j$$

$$Cost_\kappa = \sum_{m=1}^{numps_\kappa} \sum_{n=1}^{numw_\kappa} Cost(PS_m, W_n)$$

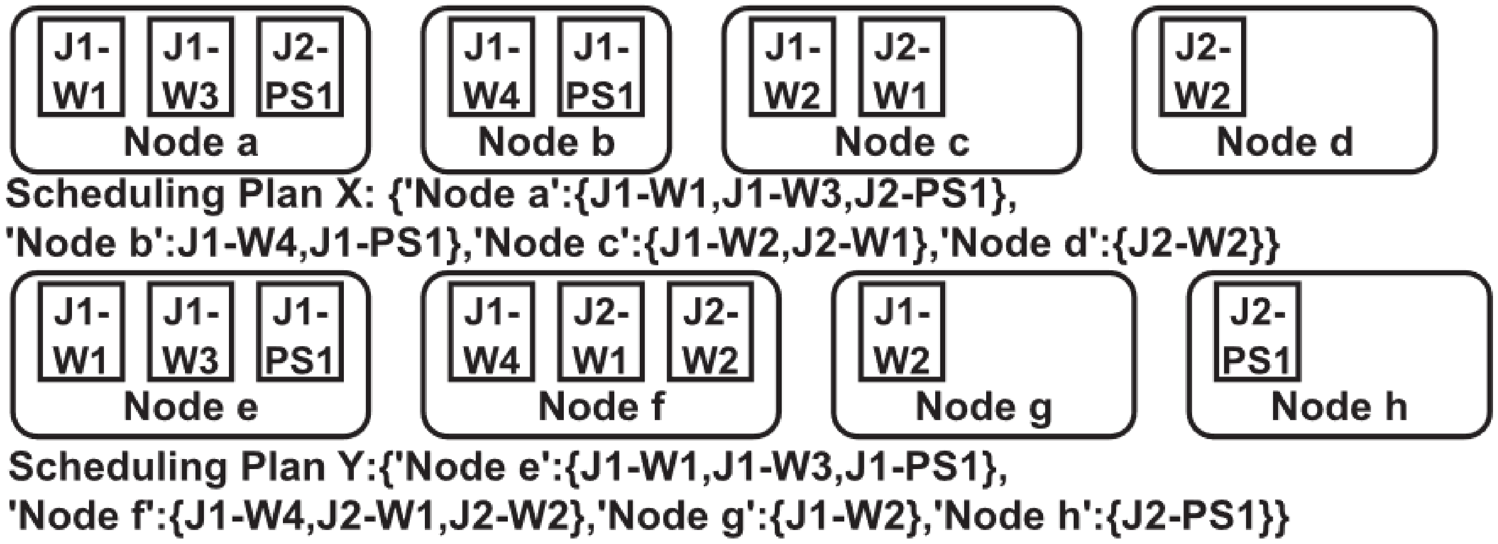
$$Fitness_j = -\frac{ReqGPU_j + UsedGPU_j}{TotalGPU_j}$$

- Tends to distribute instances in a **centralized manner** (?)

Best-fit Algorithm

1. For each instance of the job, select candidate computing nodes that meet the resource requirement
2. Use the evaluation function to calculate the score after placing the instance on the candidate computing node
3. Select the candidate node with the smallest score, and place the instance on the computing node (*remove the node from the candidate node set*) (?)
4. Iterate until all instances are placed

- Gene representation of a scheduling plan

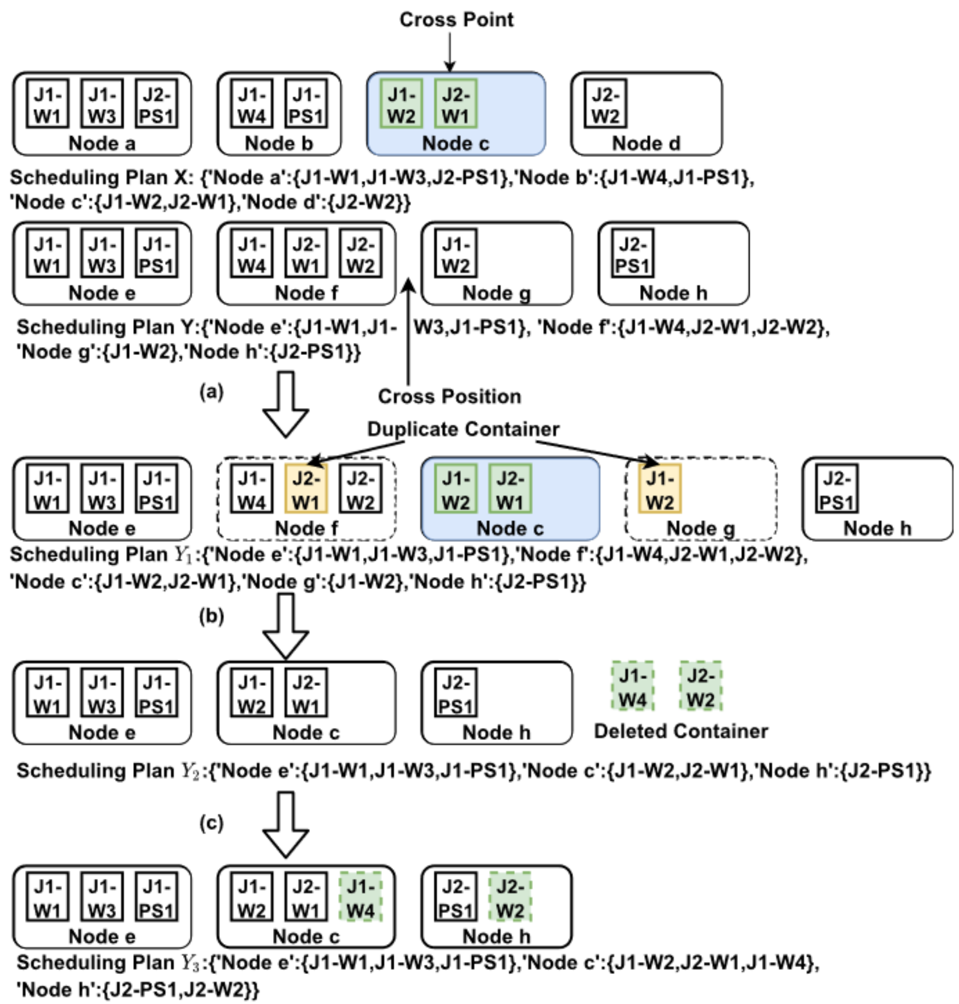


$$GeneticFitness = -\sum_{i \in \gamma} Score_{\kappa_i} - NumNU$$

东南大学 SOUTHEAST UNIVERSITY

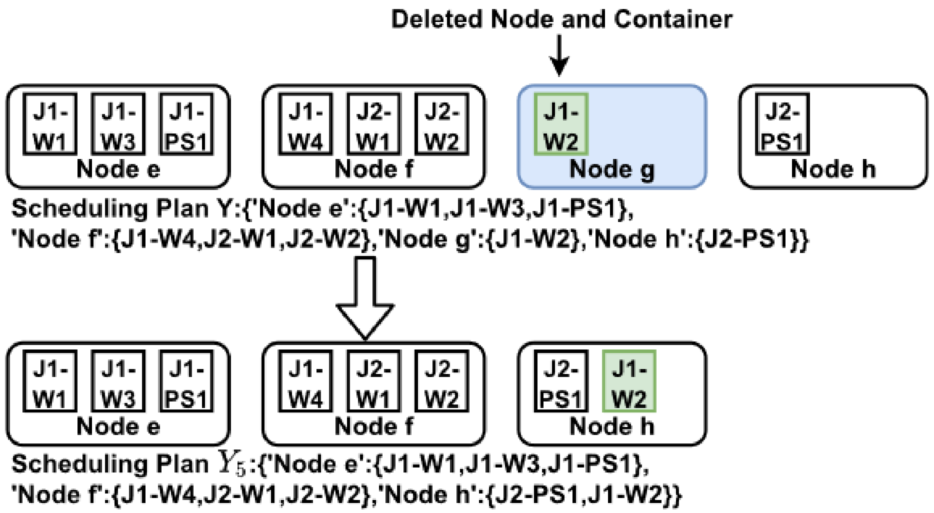
Crossover operation

- Select intersection point (computing node) and intersection location
- Delete computing nodes with duplicate containers
- Add the deleted containers to the remaining computing nodes using first-fit algorithm



Mutation operation

- Select a computing node randomly, deleting the node and the containers on it
- Replace the deleted container on the remaining nodes using first-fit algorithm



SOUTHEAST UNIVERSITY

Pre-Scheduling Data Transmission

- DL job running process can be divided into launching phase, training phase and saving phase ← Following job can be scheduled to the GPU ahead for preparation when a job comes to the end of the training phase

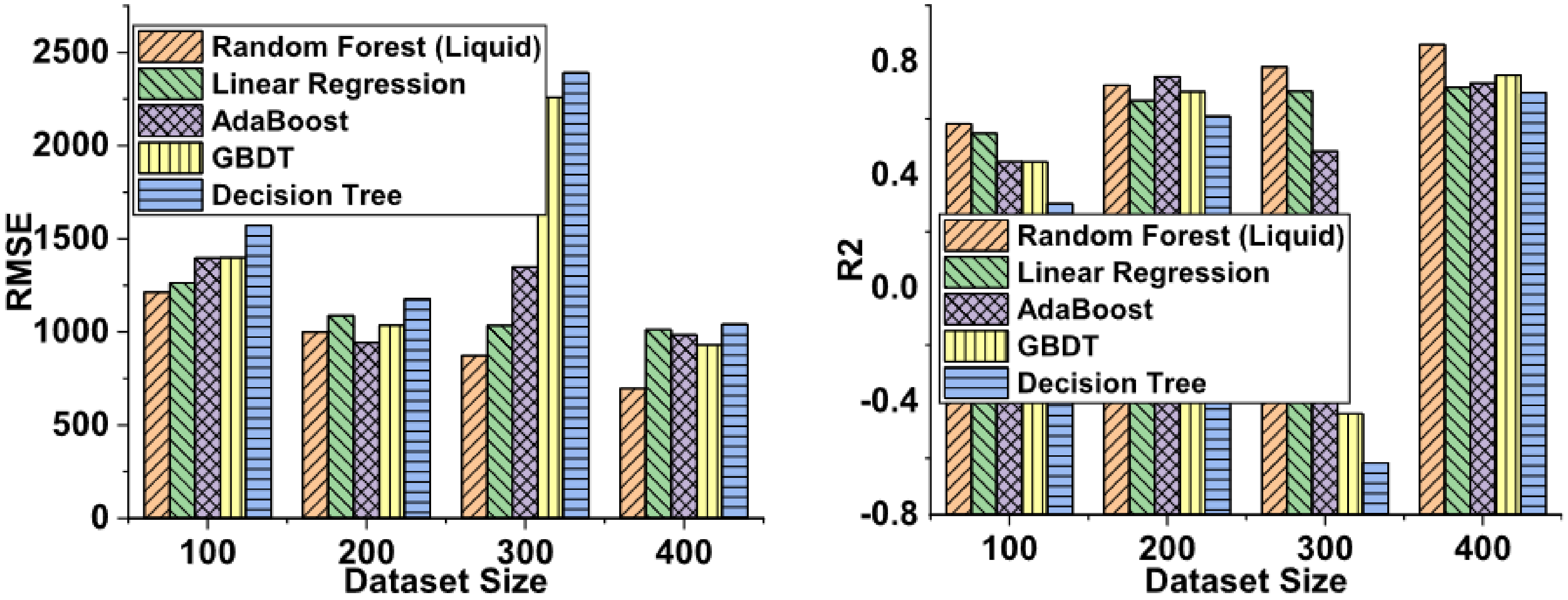
Fine-Grained GPU Sharing

- Exclusive use of GPU → Sharing GPU when executing tasks for development and testing purposes (low requirement for GPU resources)

Event-Driven Scheduler

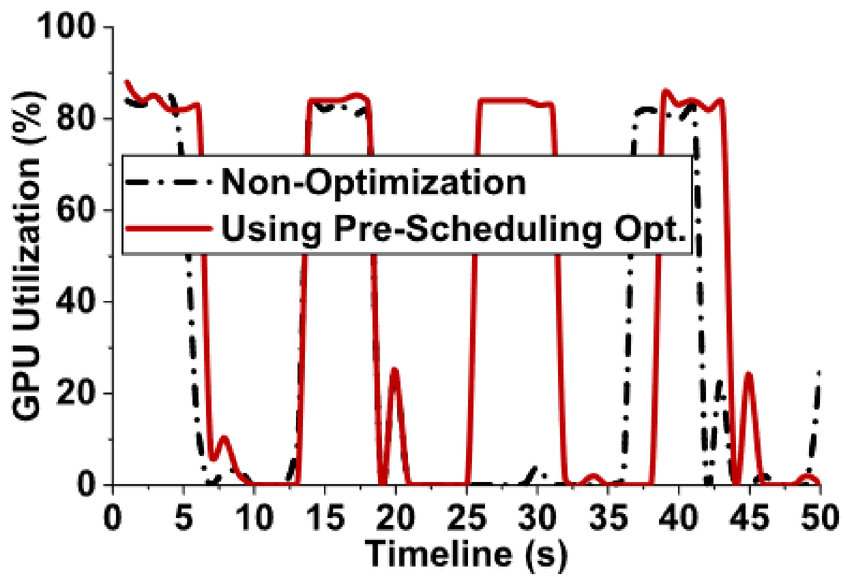
- Polling → Event-driven

Evaluation - Resource Requirement Estimation Model

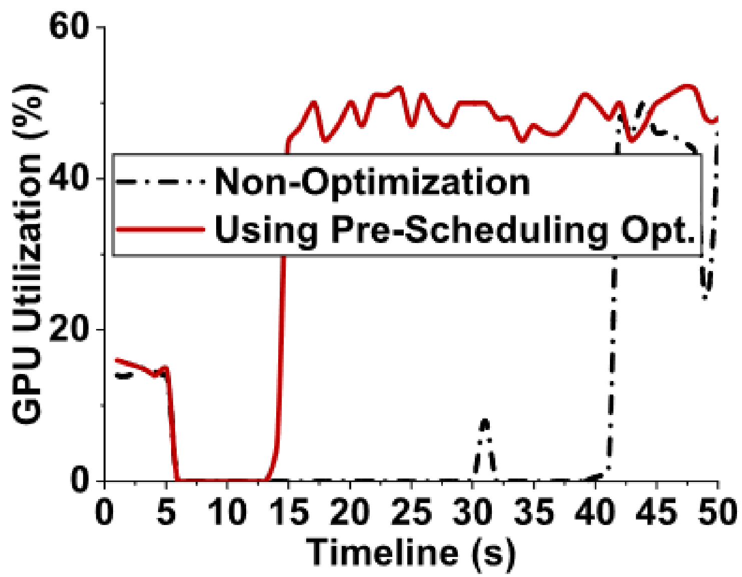


(a) RMSE (smaller is better) (b) R2 (larger is better)

- **Pre-Scheduling Data Transmission**

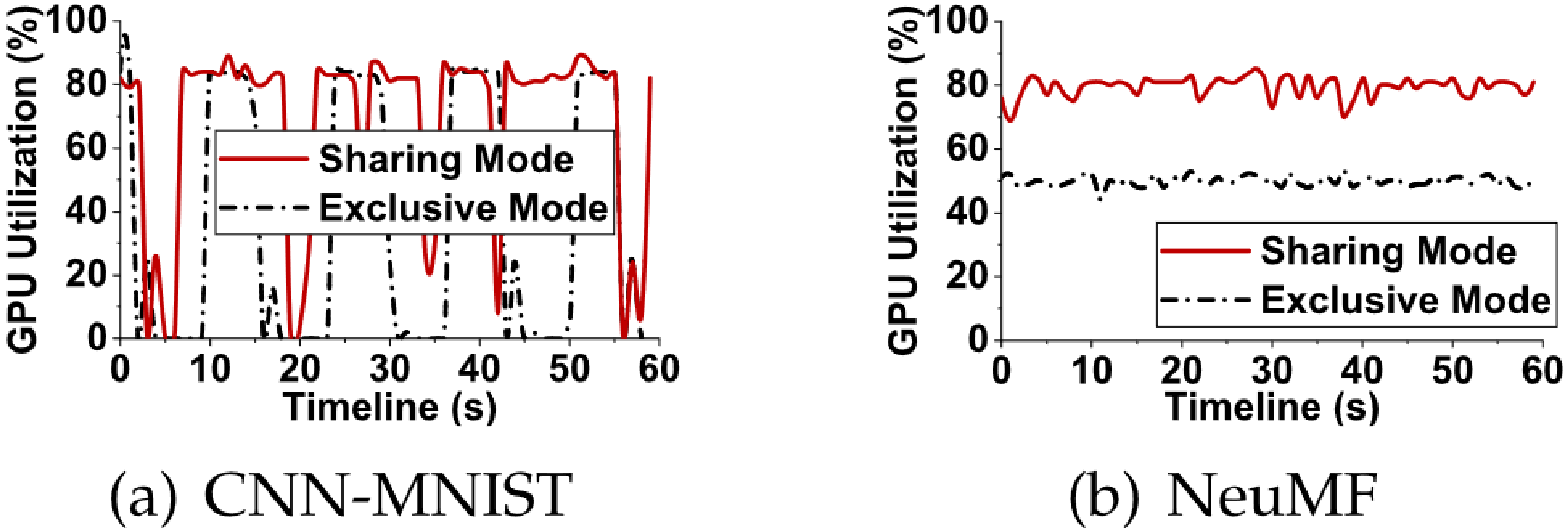


(a) CNN-MNIST

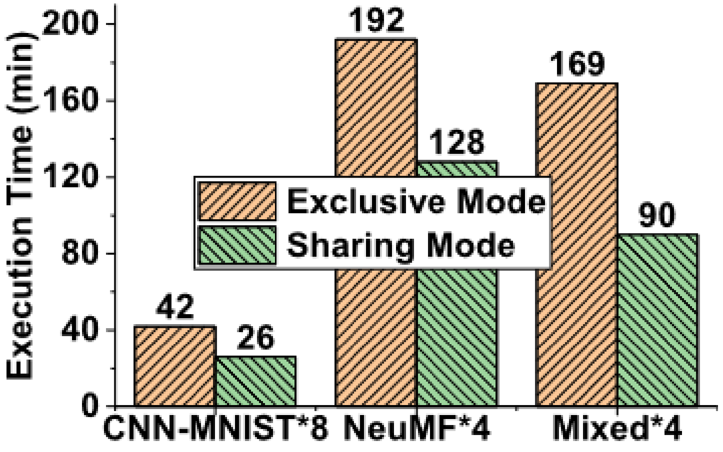


(b) NeuMF

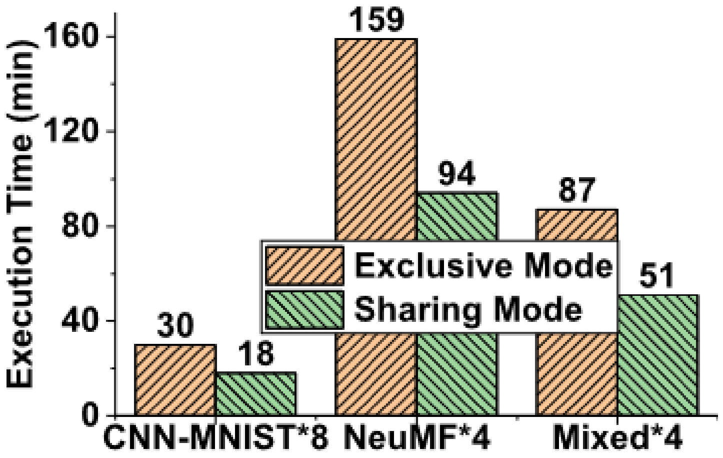
- **Fine-Grained GPU Sharing**



- **Fine-Grained GPU Sharing**



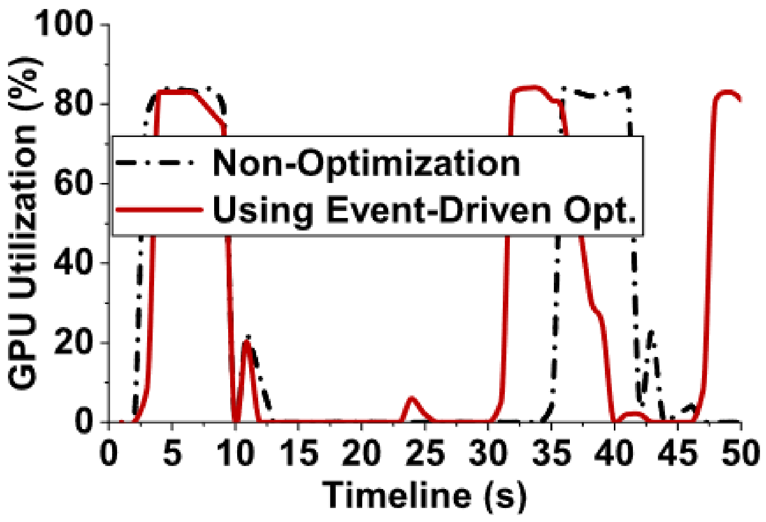
(a) NVIDIA Tesla K80 GPU



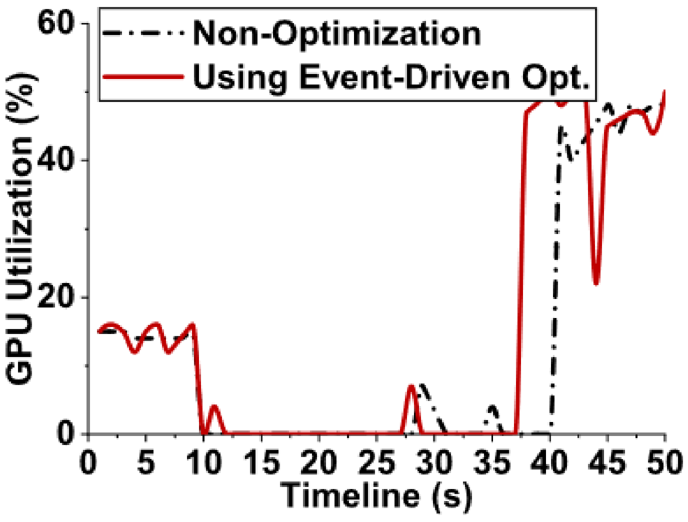
(b) NVIDIA Teala T4 GPU

东南大学 SOUTHEAST UNIVERSITY

- **Event-Driven Scheduler**

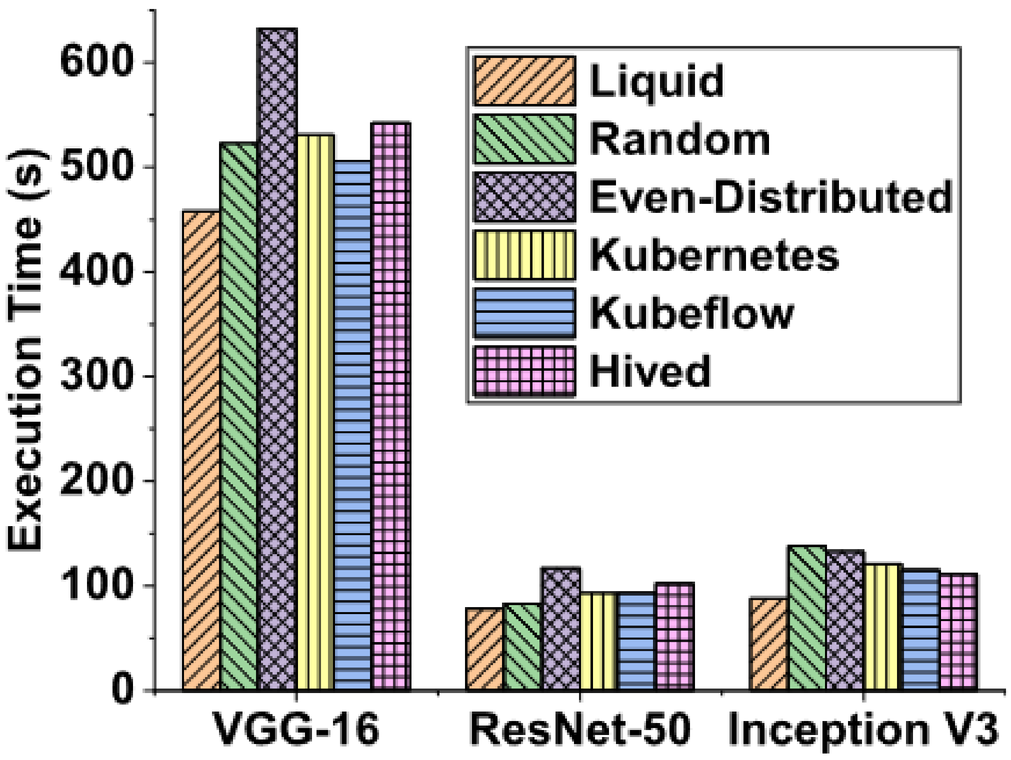


(a) CNN-MNIST

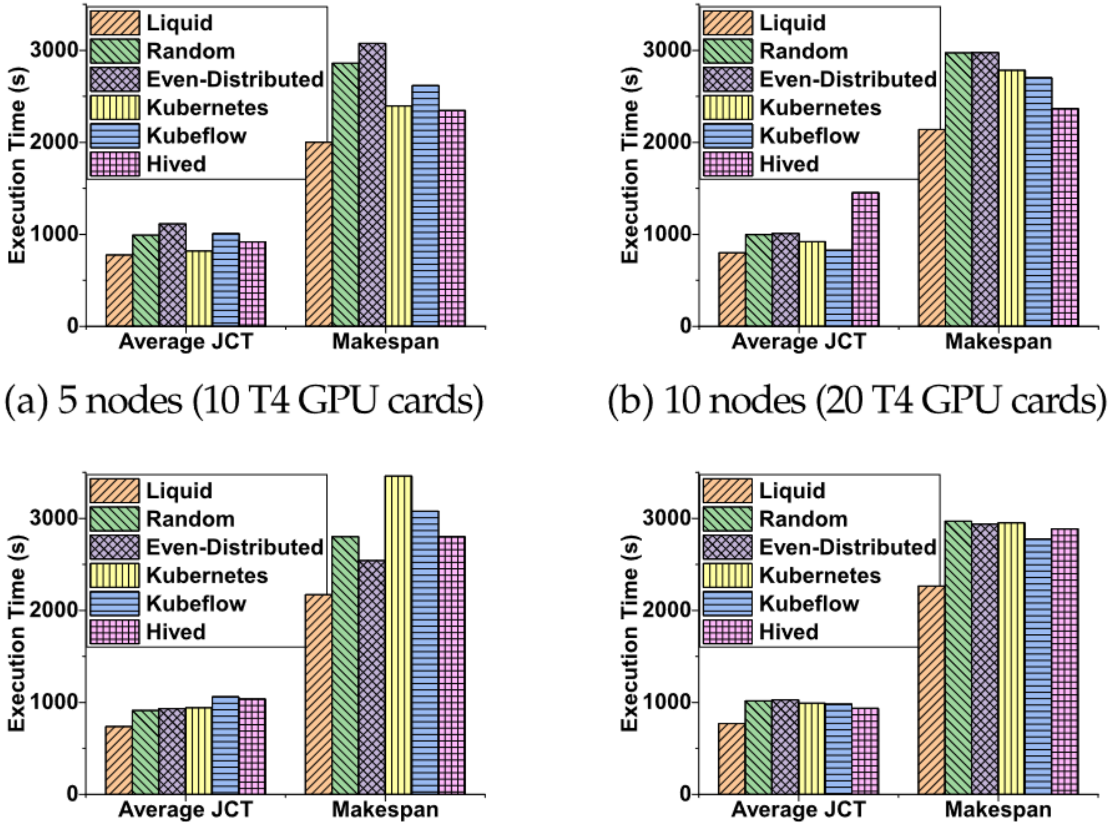


(b) NeuMF

- **Immediate Scheduling Strategy**



- **Group Genetic Algorithm Based Batch Scheduling Strategy**



(a) 5 nodes (10 T4 GPU cards)
(b) 10 nodes (20 T4 GPU cards)
(c) 15 nodes (30 T4 GPU cards)
(d) 20 nodes (40 T4 GPU cards)

Discussion

- Instance Overcommitment?
  - Resource Allocation ↓
  - Latency ↑ (Meet Quality-of-Service)
- Enable online training for estimation model?
  - Require large amount of training data
  - *Mondrian Forest* (online random forest algorithm)
- Take hardware heterogeneity into consideration?
  - Different hardware configurations
  - Aligned batch size?
- Inter-container resource contention?
  - Take inter-container metrics (e.g. CPU Cache Miss Rate) into consideration