Learning to Optimize DAG Scheduling in Heterogeneous Environment

Yunfan Zhou[1], **Xijun Li**[2], Jinhong Luo, Mingxuan Yuan, Jia Zeng and Jianguo Yao

Huawei Noah's Ark Lab

January 8, 2023

[1] Equal contribution

[2] Corresponding author and equal contribution

Table of Contents

Table of Contents

Problem Description

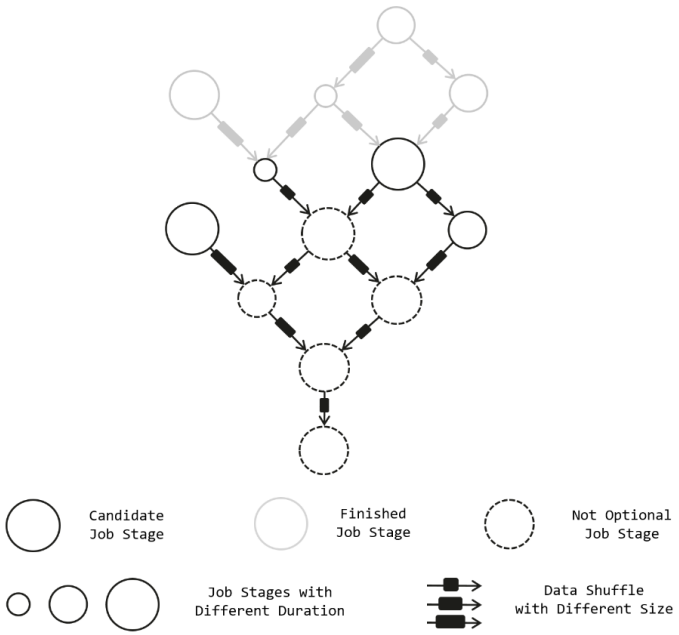
Objective

Assign job stages of **DAG jobs** in **heterogeneous environment** to minimize the makespan.

a. DAG jobs

- Vertex: A Job stage consisting of multiple parallel tasks in the DAG.
- Edge: Data shuffle between different job stages with a certain direction.

A job stage can be selected only when its predecessors are all finished or under executing!



Problem Description

Objective

Assign job stages of **DAG jobs** in **heterogeneous environment** to minimize the makespan.

b. Heterogeneous Environment

Executors that are different in:

- computing speed
- memory size
- transition bandwidth

Challenges

a. **Information representation**
   - Complex dependencies. All the job stages with their dependencies must be taken into consideration.
   - Dynamic DAG structure. The job stages arrive continuously, making the dag structure change over time.

b. **scheduling Decision**
   - A huge decision space. The scheduler assigns all job stages (usually $> 10^3$) to available executors (usually $> 10^2$).

Table of Contents

Overview

A two phase framework:

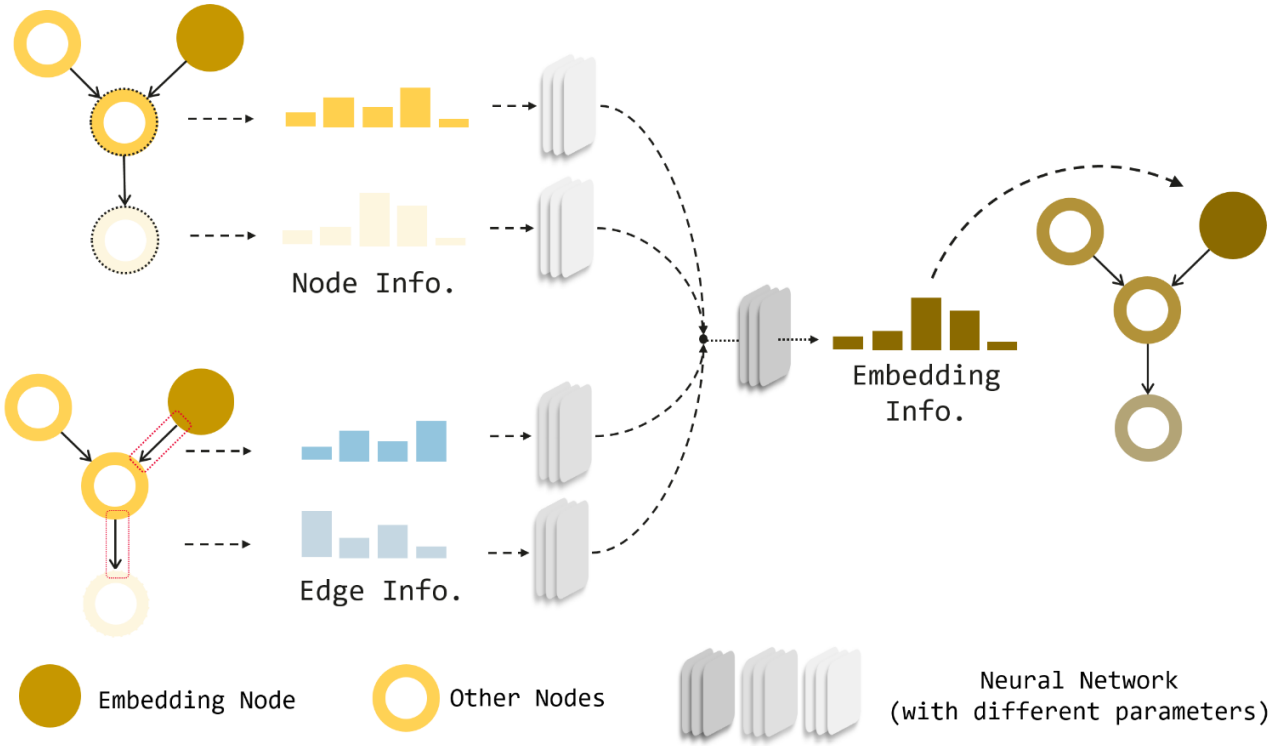
a. Job stage selection. Select a job stage by a GNN from all executable job stages.
b. Executor allocation. Allocate the selected job stage to an executor according to a heuristic method.

Why two phase?

a. A much smaller decision space for each phase.
b. Heuristic method performs good enough for executor allocation in our experiment.

Design of the GNN

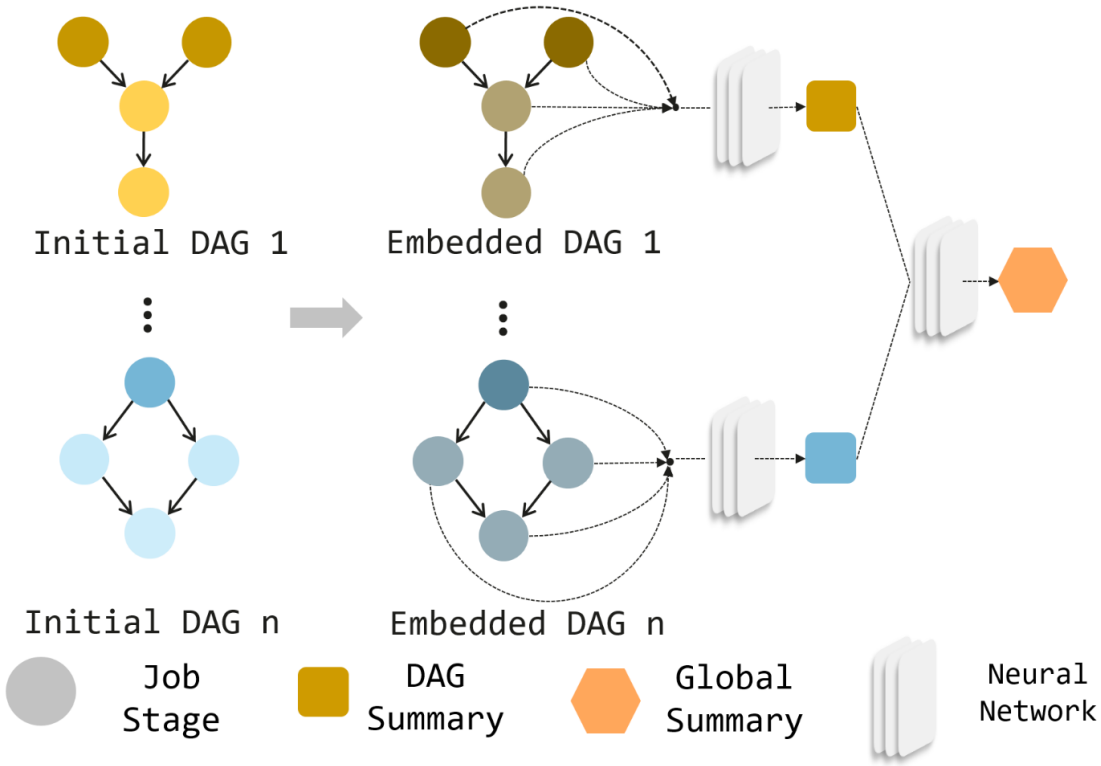
a. Information embedding.



For each executable job stage, the information contained in its successors and the vertices leading to its successors is aggregated and transferred by a two level neural network.

Design of the GNN

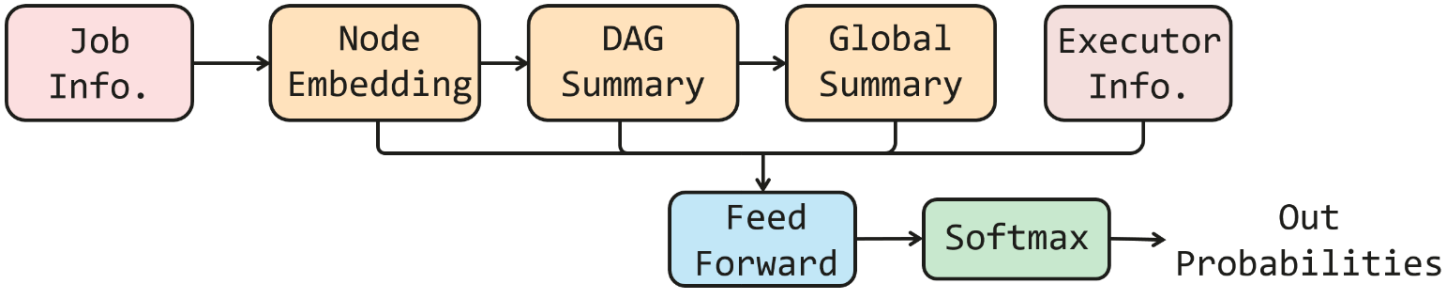
b. Information aggregation.



The information of vertices is pooled to generate DAG summaries, which is a representation for a single DAG job. Moreover, all DAG summaries are passed through a neural network to generate a global summary.

Design of the GNN

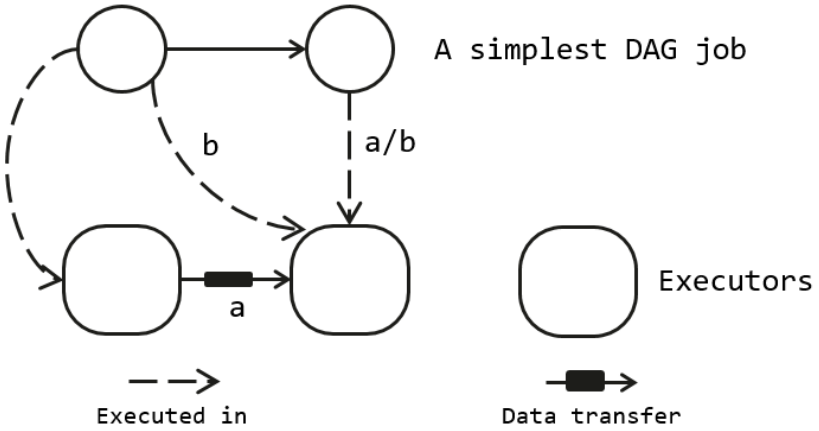
c. Framework of the GNN



- Job information and executor information are the raw input of the GNN.
- Afterwards, the job information is processed by the GNN and three representations are generated.
- Finally, the GNN outputs the probabilities of each executable job stage being selected.

Executor Allocation

Briefly, the executor allocation phase is to find the executor that can finish the job stage the earliest.



To start a job stage on a chosen executor, we can:

a. wait for the predecessor job stage to be finished and the needed data transferred to the executor.

b. duplicate the predecessor to the executor and execute it.

Executor Allocation

To assign the i th job stage to an executor:

a. calculate the earliest start time for each available executor.
b. finish time = start time + time to execute the ith job stage.
c. choose the executor with the earliest finish time.

Algorithm Training

We adopt a reinforcement learning paradigm to train the algorithm.

Markov Decision Process[3]

In the k th transition, we have:

State s_k: Information of DAG jobs and executors.

Action a_k: select one job stage.

Reward r_k: $r_k = -(t_k - t_{k-1})$, where t_k is the finish time of the k th job stage.

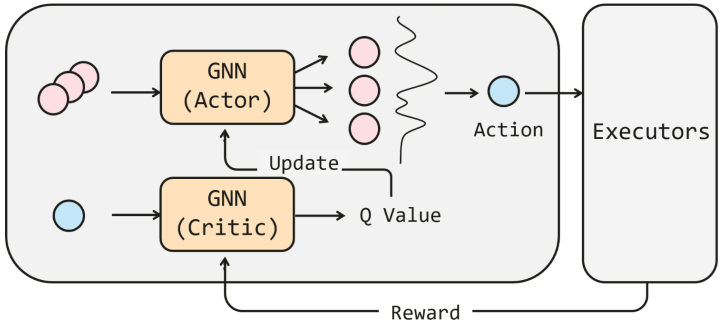
[3] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Algorithm Training

We leverage the actor-critic[4] framework to train our algorithm.

Algorithm 1 Training process

Generate N job stages randomly
Initialize ω
Initialize θ
for k $\in [0, N]$ **do**
  Receive environment information s_k
  Select action (job stage) $a_k \sim \pi_\theta$
  Select executor
  Execute action a_k and observe reward r_k, state s_{k+1}
  Estimator error:
    $\epsilon = (Q_\omega - (r_k + \mathbb{E}_{a_{k+1} \sim \pi_\theta} \gamma Q_w(s_{k+1}, a_{k+1}))^2$
  Update the critic
    $\omega_{k+1} \leftarrow \omega_k - \alpha_Q \nabla_\omega \epsilon$
  Update the actor policy:
    $\theta_{k+1} \leftarrow \theta_k - \alpha_\pi \nabla_\theta \log \pi_\theta (s_k, a_k) Q_\omega(s_k, a_k)$
end for



[4] Vijay Konda and John Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems* 12 (1999).

Table of Contents

Experiment Results

Metrics

a. Makespan: time to finish a DAG job.
b. Normalized Makespan (to compare performance on different DAG jobs):
   - Speedup: a higher speedup is better
   - Schedule length ratio (SLR): a lower SLR is better

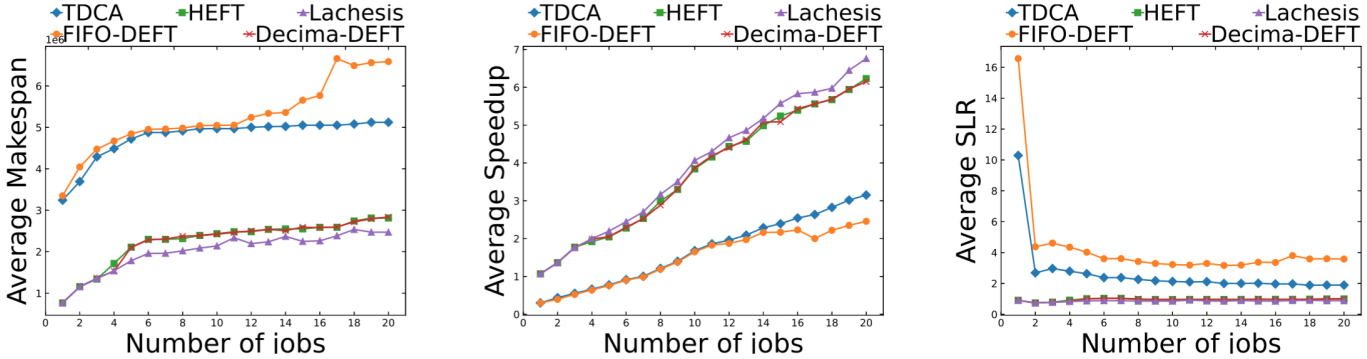
Settings

a. Our framework is named Lachesis.
b. Dataset: the experiment dataset is generated from TPC-H[5] workload.
c. Baselines: four Baselines including three heuristic method and Decima[6].

[5] TPC-H. *The TPC-H Benchmarks*. Jan. 25, 2021. URL: www.tpc.org/tpch/ (visited on 2021).

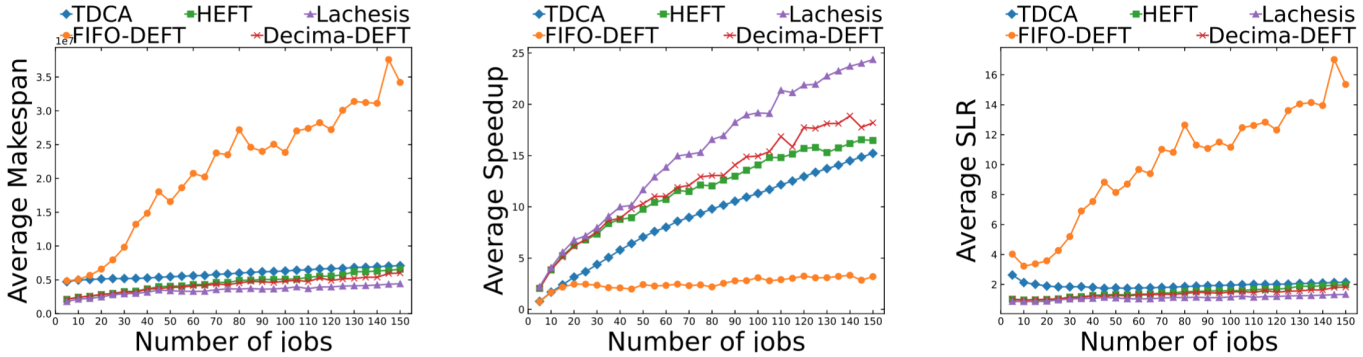
[6] Hongzi Mao et al. "Learning Scheduling Algorithms for Data Processing Clusters". In: *Proceedings of the ACM Special Interest Group on Data Communication*. SIGCOMM '19. Beijing, China: Association for Computing Machinery, 2019, 270–288.

a. Experiment results on small scale jobs.



(a) Average makespan (b) Average speedup (c) Average SLR

b. Experiment results on large scale jobs



(a) Average makespan (b) Average speedup (c) Average SLR

- The graphs above show that Lachesis out performs all baselines on both small scale jobs and large scale jobs.
- The advantage of Lachesis is significant on large scale jobs.

Table of Contents

Conclusion and Future work

Conclusion:

a. We introduce a two phase framework for DAG job scheduling.
b. The experiment results indicate that our framework effectively reduce the makespan comparing to baselines.

Future work

a. A lighter model to accelerate scheduling.
b. More effective algorithm for executor allocation.

THANKS