Multi-Agent Local Voting Protocol For Online DAG Scheduling

Nickolay Zhitnukhin[1], Anastasia Zhadan[1], Ivan Kondratov[1], Alexander Allahverdyan[1],
Ovanes Petrosian[1], Aleksei Romanovskii[2], Vitaliy Kharin[2]

Saint Petersburg State University[1]
Huawei Russian Research Institute[2]

Центр искуственного интеллекта и науки о данных СПбГУ

ICOMP
INTERNATIONAL CONFERENCE ON COMPUTATIONAL OPTIMIZATION

Graph Scheduling / Distributed Heterogeneous Computing

Directed acyclic graph (DAG) $G = (J, E)$; J- set of nodes, E – set of edges:

- Node $v \in J$ represent a computational task (job) with a given type of executor that it should be executed upon
- Edge $(v_i, v_j) \in E$ denotes precedence constraint, i.e. node v_j cannot be executed until v_i and all its other predecessors are not complete
- Node without predecessors – entry node, without successors – exit node; there can be multiple of both types in a given DAG
- Cost of communication is $b_{i,j}$ defined for all edges $(v_i, v_j) \in E$ and should accounted for if v_i and v_j are executed by different executors

Distributed heterogeneous computing consists of a set P where $p_i \in P$ is heterogeneous executors with fully connected topology, for which defined:

- Computational cost matrix D, where $D^{i,j} = w_{i,j}$ is the execution time required for executor $p_i \in P$ to process node $v_j \in J$
- Each executor has a defined type; to execute task v_j on p_i it is required that their types match

Goal is to construct mapping of nodes from DAG to executors, such that

$$makespan \rightarrow min$$

where makespan is $\max_{v \in V}(actual\ finish\ time\ of\ v)$.

Online vs offline scheduling

Online sliding informational window: at each given moment the **scheduler observes** only **ready-to-execute** nodes and their **immediate successors** within the **informational window with length 2**. This limits amount of information and makes problem online: **graph is not known in advance**, opposite to offline case

MLVP

At each moment of time t, executors calculate a probability of assignment of a given task (node). Probability is calculated as a ratio of a given node's **aggregated metric** to sum of all other observable nodes **aggregated metrics.** It is constructed from following characteristics of a DAG and executors:

- **Incoming connections –** ratio that represent how many unfinished parent nodes has each of a given node's child nodes W_t
- **Available nodes –** ratio of tasks of a matching to the given executor's type that are ready-to-be-executed Q_i
- **Relative performance difference –** difference between minimum execution time of a given node by all executors to a given executor Z_i

Coefficients for sub-metrics are calculated using genetic algorithm with single-point crossover and random mutation, and selective fixed-size elitism.

Assignment probabilities are synchronized using **Local Voting Protocol (LVP):** Each executor shares its task assignment probabilities with neighboring executors that iteratively update their task probabilities based on the state of their neighbors and **consensus** is reached when the difference in task assignment probabilities across all neighboring executors falls below a predefined threshold (ε = 0.05)

Offline methods [1, 2]

MARL [1] first level – make ordered list of tasks to be executed by each type of resource via Multi-Agent Proximal Policy Optimization with agent for each type

- **State**: matrix of 13 DAG metrics [1] for each buffer of ready-to-execute node with pre-defined length (hyperparameter)
- **Action**: select metric by which order the list of ready-to-executed nodes buffer
- **Reward:** normalized difference between MILP and MARL makespan after each DAG scheduled (training on a batch of DAGs at once)

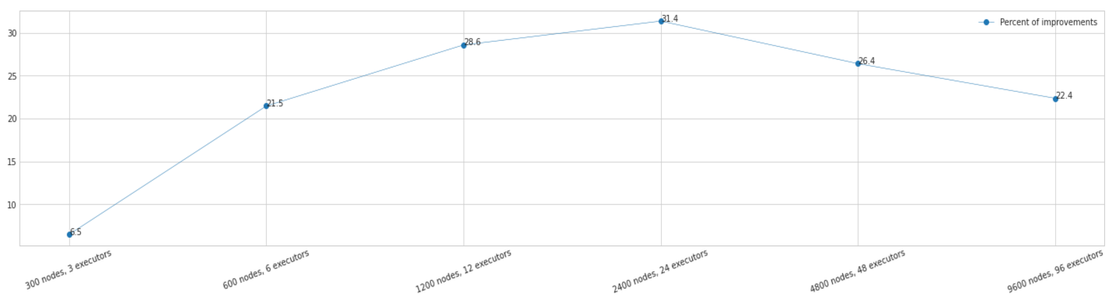
Second level – allocate tasks from ordered list using Earliest Finish Time (EFT)

NN [2] First level – make ordered list of tasks to be executed by each type:

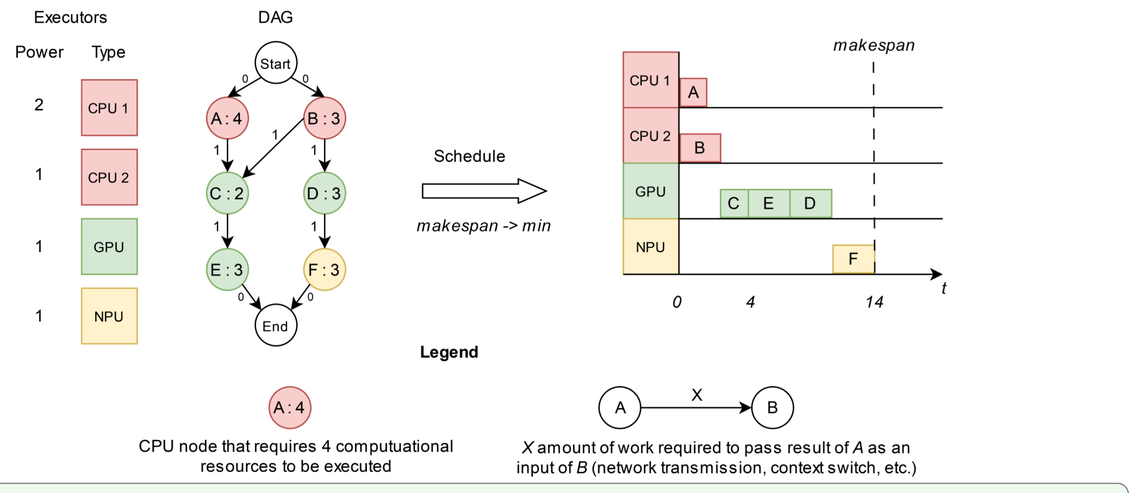
- **Single agent :** Neural Network trained using Genetic Programming approach
- **Genetic programming allows to skip construction of differentiable makespan improvement function**
- **Features**: 8 DAG metrics [3] + type
- **Architecture:** 3 fully connected layers + Genetic Programming algorithm

Second level – allocate tasks from ordered list using Earliest Finish Time (EFT)

	Workspace 3			Workspace 2			Workspace 1		
Dimension	DONF	NN	MARL	DONF	NN	MARL	DONF	NN	MARL
30	91.51	**95.45**	92.03	85.92	**92.49**	86.44	85.04	**95.79**	89.88
60	83.76	**93.13**	85.05	74.28	**86.13**	77.71	75.77	**95.35**	82.73
90	73.44	**91.31**	82.05	70.15	**89.19**	77.03	74.44	**95.20**	80.98



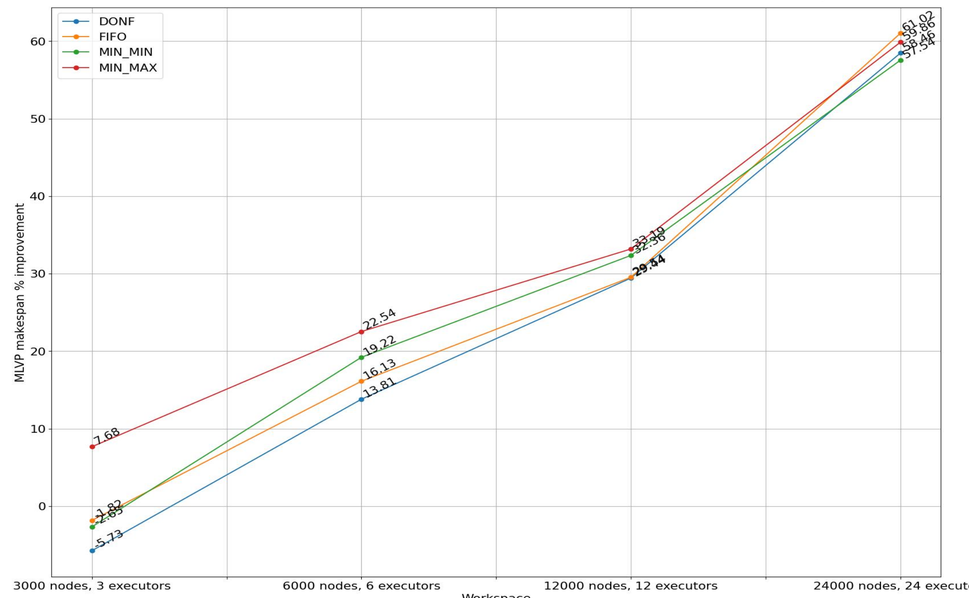
Example



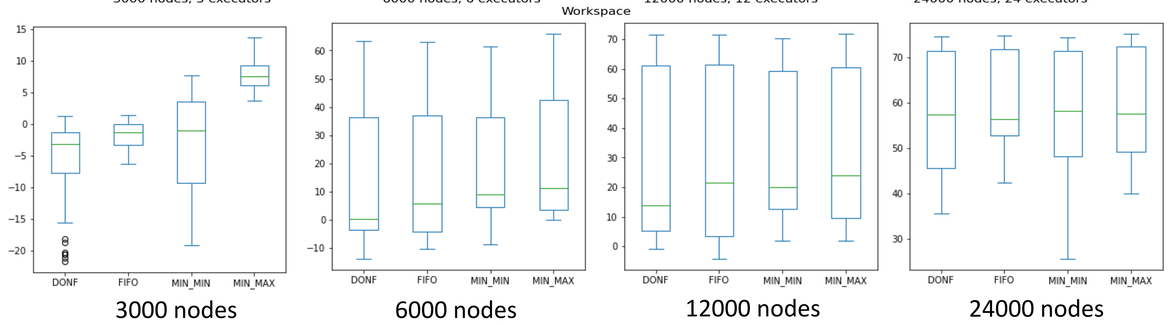
Evaluation

DAGs generated DAGGEN [2] – industry standard for evaluating DAG heuristics

- **Executors:** 3, 6, 12, 24 (with n executors of each type 1, 3, 4, 8)
- **Number of nodes:** 3000, 6000, 12000, 24000 respectively to executors



Average % of makespan improvement by MLVP



Box plot percentage improvements in makespans MLVP vs other algorithms

Predicting DAG [3]

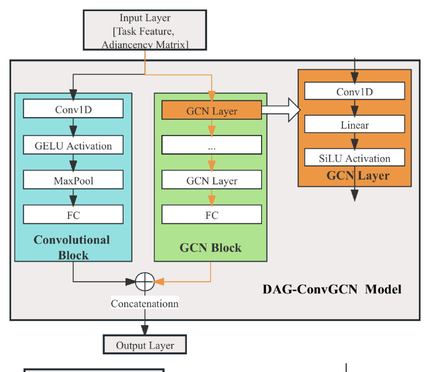
Use machine learning approaches [3] to **predict computational tasks and dependencies between them** for accurate prediction of the next task's properties and its dependencies to the existing **task to extend online method's DAG knowledge and capabilities**

Table 5 Similarity between Original and Predicted DAGs

Models	Propagation Kernel	GRBF Kernel
LR	0.7612	0.9257
SVM	0.7571	0.7509
XGBoost	0.7089	0.9646
LightGBM	0.7116	0.9450
MLP	0.9494	0.9521
RNN	0.9430	0.9089
CNN	0.9427	0.9777
GCN	0.9478	0.9671
DAG-ConvGCN	0.9751	0.9798
DAG-AttGCN	0.9743	0.9879

Table 4 Prediction Quality

Models	Avg_MSE	Avg_R^2	Avg_BCE	Avg_$Accuracy$
LR	0.0188	0.7577	0.7992	0.9769
SVM	0.0190	0.7554	0.9746	0.9718
XGBoost	0.0104	0.7670	0.2290	0.9924
LightGBM	0.0126	0.7780	0.0624	0.9727
MLP	0.0310	0.6425	0.3879	0.9978
RNN	0.0231	0.7449	1.3828	0.9906
CNN	0.0165	0.8150[1]	1.4649	0.9904
GCN	0.0247	0.7354	0.3686	0.9983[2]
DAG-ConvGCN	0.0148	0.8335	0.0628	**0.9997**[3]
DAG-AttGCN	0.0139	**0.8495**[4]	0.0578	0.9996



[1] Multi-agent Reinforcement Learning-based Adaptive Heterogeneous DAG Scheduling, ACM Transactions on Intelligent Systems and Technology, Volume 14, Issue 503, 2023, pp 1–26
[2] Heterogeneous Computational Scheduling using Adaptive Neural Hyper-heuristic, Doklady Mathematics (accepted, will be printed next month)
[3] DAGCN: hybrid model for efficiently handling joint node and link prediction in cloud workflows. Appl Intell 54, 12505–12530 (2024). https://doi.org/10.1007/s10489-024-05828-w