MULTI-AGENT LOCAL VOTING PROTOCOL FOR ONLINE DAG SCHEDULING

Nickolay Zhitnukhin, Anastasia Zhadan, Ivan Kondratov,*
Alexander Allahverdyan, Ovanes Petrosian
Saint Petersburg State University
Saint Petersburg, 198504, Russia
kola1197mail@gmail.com, anastasiya_markel@mail.ru,
kondratov.ivan.vladimirovich@gmail.com, aleka_alexander@mail.ru,
petrosian.ovanes@yandex.ru

Aleksei Romanovskii & Vitaliy Kharin
Saint Petersburg Research Center
Huawei Russian Research Institute
Saint Petersburg, 191119, Russia
{aleksei.romanovskii, vitaliy.kharin}@huawei.com

ABSTRACT

Scheduling computational workflows represented by directed acyclic graphs (DAGs) is crucial in many areas of computer science, such as cloud/edge tasks and data mining. The complexity of online DAG scheduling is compounded by the large number of computational nodes, data transfer delays, and the non-uniform arrival of tasks. This paper introduces the Multi-Agent Local Voting Protocol (MLVP), a novel approach focused on dynamic load balancing for DAG scheduling in heterogeneous computing environments, where executors represented as an agents. The MLVP employs a local voting protocol to achieve effective load distribution by formulating the problem as a differentiated consensus achievement. The algorithm calculates an aggregated DAG metric for each executor-node pair based on node dependencies, node availability, and executor performance. These metrics are optimized using a genetic algorithm to assign tasks probabilistically, achieving efficient workload distribution across the system and thus improving makespan. The effectiveness of the MLVP is demonstrated through comparisons with state-of-the-art DAG scheduling algorithm and popular heuristics such as DONF, FIFO, Min-Min, and Max-Min. Simulations show that MLVP achieves makepsan improvements of up to 70% on specific graph topologies and an average makespan reduction of 23.99% over DONF across various random DAGs. Notably, the algorithm's scalability is evidenced by enhanced performance with increasing numbers of executors and graph nodes.

1 INTRODUCTION

In heterogeneous computing systems, task scheduling and resource allocation present significant challenges. These systems integrate various types of processors, such as CPUs, GPUs, and FPGAs, to execute tasks efficiently and optimize resource utilization. Workflows in these systems are often modeled as directed acyclic graphs (DAGs), which clearly represent task dependencies, where nodes denotes individual tasks and edges reflect precedence and data transfer costs between tasks.

Numerous studies examine online and dynamic scheduling, often using the terms interchangeably. Typically, papers specify the scheduling model and context, which depend on executor parameters, especially in fog computing environments (Alizadeh et al. (2020)). Task arrival attributes, whether random-based for non-DAG models (Alizadeh et al. (2020)) or involving random arrival times for

*Corresponding author.

known DAGs (Quintin & Wagner (2012)), also define this. When DAGs aren't fully visible, schedulers have limited visibility and cannot use heuristics like the Critical-Path method like CPOP and its successors like HEFT (Ghose & Dey (2022); Grandl et al. (2014)).

Significant research on online and dynamic scheduling for DAG-modeled applications has been contributed by major companies. Microsoft's research team has focused on resource management (Mao et al. (2016)) and cluster scheduling (Grandl et al. (2014)). Cloud providers, including Microsoft (Jalaparti et al. (2015); Yan et al. (2016)), Amazon (Durillo & Prodan (2013)), Google (Soualhia et al. (2015)), and IBM (Center & Feitelson (1994)), have published findings on process optimization and scheduling. Distributed application frameworks like Spark (Duan et al. (2020)) and Flink (Li et al. (2020)) utilize DAGs to depict applications and employ various online scheduling methodologies.

This paper introduces the Multi-Agent Local Voting Protocol (MLVP), a novel approach to DAG scheduling in heterogeneous computing environments. The MLVP leverages a multi-agent framework where executors, acting as agents, use a local voting protocol to dynamically balance loads across the system and in turn reducing makespan. The protocol calculates an aggregated DAG metric for each task-executor pair, optimized via a genetic algorithm, to ensure efficient task assignment. This method addresses the limitations of traditional scheduling algorithms by accommodating the dynamic and online nature of real-world computing tasks.

The remainder of this paper is organized as follows. Section 2 provides an overview of the online scheduling problem and dynamic DAG scheduling approaches. Section 3 describes distributed heterogeneous computing system, DAG model and performance criteria. Section 5 details the proposed MLVP-based scheduling algorithm and its architecture. Section 6 evaluates the MLVP-based scheduling algorithm performance under different workloads.

2 Related works

In online scheduling in general, review by Alizadeh et al. (2020) identifies popular scheduling algorithms: MCT, Min–Min, and Max–Min used in fog computing for task allocation based on task size and completion times. Review by Khallouli & Huang (2022) assess scheduling methods in YARN, Borg, and Kubernetes, and advocate for "smarter" approaches, like reinforcement learning and decision trees to refine multi-objective optimization in cloud resource allocation, addressing the complexities of heterogeneous jobs and resources.

Online DAG scheduling often deals with incomplete knowledge of tasks and communications. To address this, Quintin & Wagner (2012) introduces WSCOM, a variation of the work-stealing algorithm, for scheduling file execution tasks. Grandl et al. (2014) tackles online task scheduling in data-parallel clusters that require diverse resources such as CPU, memory, disk, and network. They propose the Tetris scheduling method, which strives to optimally assigns tasks based on comprehensive resource needs.

The consensus approach is increasingly popular for addressing various practical issues, such as cooperative control in multi-vehicle networks (Granichin et al. (2012); Ren et al. (2007)), distributed control of robotic networks (Bullo et al. (2009)), the flocking problem (Yu et al. (2010); Virágh et al. (2013)), and optimizing sensor networks (Kar & Moura (2010)). It holds promise for load balancing in computer, production, transport, logistics, and service networks by framing the problem as a consensus issue among network nodes (Amelina et al. (2015)). The study in (Vergados et al. (2018)) examines optimal task redistribution in stochastic networks with variable priorities by determining an ideal step-size for a consensus-type protocol.

3 Online DAG Scheduling Problem

This paper addresses the challenge of scheduling a single DAG within a distributed heterogeneous system, specifically when executors have visibility limited to only the initial layers of the working front. The scheduling model is composed of four distinct parts.

Directed acyclic graph $G = (C, E)$, where $C = 1, 2, 3...c$ is a set of nodes and E is the set of edges, where:

- Edge $(v, s) \in E$, $v, s \in C$ denotes the precedence constraint such that node j must wait until task v finishes its execution.
- Cost of communication $b_{v,s}$ between nodes v and s, $(v, s) \in E$ should be taken into account if node i and s assigned to different executors, otherwise there is no cost of communication. If node s has several parents $(v, ..., k), k \in O$ that were performed on executors other than the executor assigned to node v, then the cost of communication are taken into account from each parent $b_{v,s}$ +..+ $b_{k,s}$.
- Each node v has a type that denotes on what type of executor should this node be executed.
- Set of immediate predecessors of node i in a DAG is expressed as $pred(v)$. A node without any predecessor is called an entry node. Multiple entry nodes may exist in a DAG.
- Set of immediate successors of node v is expressed as $succ(v)$. A node without any successor is called an exit node v_{exit}. There may be multiple exit nodes in a DAG.

Distributed heterogeneous computing system is represented by the graph $D = (N, L)$, where:

- Graph has $N = 1, 2, 3...n$ executors that work in parallel, L bidirectional links exist between any two executors.
- In every moment of time t each executor performs his task or chooses a new one from the DAG ready to perform nodes $H^t = 1, 2, 3...h^t$, $H^t \subset O$.
- Each node $i, i \in H^t$ of the DAG has $R^i = 1, 2, 3...r^i$ child nodes and each node $j^m, j^m \in R^i$ could have not only node i as a mother nodes.
- Executor $m, m \in N$ chooses the new node to perform in the state $x^m(t)$. The state of executor m is the evaluation of the probability of choosing a node P_i^m for each DAG node i, at each moment of time.
- τ_i^m is the execution time for executor m to process the node i.
- All executors are divided in equal shares according to the types of DAG nodes.
- Each executor m has a type that denotes what type of tasks can be defined for this executor. In order to assign a node i to a executor m, it is necessary that the type of node i and the type of executor m match. Executor m receives a penalty $w_{v,m}$ for each parent node of v, that is not perform by node m.

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 that can be utilized for scheduling and makes problem online, i.e. scheduler cannot utilize information about nature of task arrival. So, on the one hand, in comparison with the static scheduling there are significantly less information, it makes impossible to utilize classic DAG scheduling algorithms like CPOP or HEFT, that requires full information about DAG. On the other hand, limited information about DAG topology and parameters of not yet observed tasks combines limitations described in various online DAG scheduling papers (see introduction) and makes problem more difficult. If compared with purely online scheduling, there are more information available such as dependencies and parameters of tasks that inside sliding informational window.

Performance criterion for scheduling: before presenting the final scheduling objective function, we first define the Makespan, Earliest Start Time(EST), Earliest Finish Time(EFT) attributes.

- Makespan is the finish time of the last node in the scheduled DAG. It is defined by $makespan = max\{AFT(v_{exit})\}$ where $AFT(v_{exit})$ is the actual finish time (AFT) of exit node v_{exit}. In the case where there are multiple exit nodes, the makespan is the maximum AFT of all exit nodes.
- $EST(v, m)$ denotes the earliest start time of node v on executor m and it is defined as $EST(v, n) = \max\{T_{Ava}(m), \max_{\{v \in pred(v)\}}\{AFT(v) + b_{v,s}\}\}$. $T_{Ava}(m)$ is the earliest ready time of executor m.
- $EFT(v, m)$ denotes the earliest finish time of node v on executor m and is defined as $EFT(v, m) = EST(v, m) + w_{v,m}$.

The objective function of the DAG scheduling is to determine the assignment policies of an application's node to heterogeneous executors so that the makespan is minimized.

4 GRAPH GENERATION

We utilize the open-source project DAGGEN to generate random DAGs (22). It is a popular tool for generating DAGs to evaluate the performance of scheduling heuristics and has been used in studies on DONF (Lin et al. (2019)), CPOP (Sih & Lee (1993)), HEFT (Topcuoglu et al. (1999)), and PETS (E & Thambidurai (2007)). DAGGEN creates random, synthetic task graphs for simulation purposes, with the process varying according to the configured parameters:

1. Generate the tasks according to -n, -fat, -regularity
2. Generate the dependencies according to -density, -jump, -ccr
3. Add transfer costs, these costs derive from the size of the data handled by the initiator of the transfer

Table with all parameters used for DAG generation can be found in appendix A.

5 DAG SCHEDULING ALGORITHMS

The Multi-Agent Local Voting Protocol (MLVP) algorithm is designed to optimize task scheduling for directed acyclic graphs (DAGs) in heterogeneous computing environments. The algorithm achieves efficient task distribution by calculating an aggregated metric for each executor-DAG node pair and assigning probability of given node execution on this executor. Aggregated metric is derived from three key factors:

1. Incoming Connections: The ratio of the total incoming connections from unfinished DAG nodes to their child nodes
2. Available Nodes: The number of DAG nodes currently available for execution compared to nodes of the same type
3. Executor Performance: The performance capability of the executor in processing task

A genetic algorithm is employed to optimize the coefficients of these factors in a linear combination, forming the aggregated DAG metric. This metric serves as a basis for determining the probability of assigning a DAG node to a specific executor. By optimizing these coefficients, the genetic algorithm ensures that the metric accurately reflects the importance of each factor, leading to better task prioritization and scheduling efficiency.

During the Local Voting Procedure, the genetic algorithm further refines the assignment probabilities by dynamically adjusting them to balance the workload across executors. This iterative optimization process helps the system adapt to varying task and executor conditions, thereby reducing makespan and enhancing overall performance.

Consensus plays a crucial role in the MLVP algorithm by coordinating task distribution among executors. The local voting protocol (LVP) facilitates consensus by enabling executors to share their current task assignment probabilities and states with neighboring executors. The protocol iteratively adjusts these probabilities to ensure a balanced workload distribution, which is achieved when the difference in task assignment states between any executor and its neighbors falls below a specified threshold (ϵ), which in this case was 0.05. Algorithms pseudocode can be found in Appendix B.

5.1 EVALUATION OF PROBABILITY FOR NODES

At each moment of time t the executors choose $|H^t|/0.8n$ nodes for estimation in time moment t randomly. To synchronize the data of all nodes the local voting protocol (LVP) is used. The LVP is necessary to reach a consensus on the state between the executors.

For each node i at time moment t the following values are calculated by the algorithm:

1. Ratio of the sum of all incoming connections from not finished DAG node into child's nodes of node i to the number of child nodes of node i:

$$W_t = \sum_{j^i \in R^i} inp_t(j^i)/R^i,$$

where R^i is the number of child nodes of i node, $inp_t(j^i)$ is the number of **not finished** mother nodes of j_i node.

2. Sum of the ratio of DAG nodes number currently available for execution to the number of the same as j^i types:

$$Q_i = \sum_{j^i \in R^i} (|H^t|/size_{j^i}(|H^t|)),$$

where $size_{j^i}(H^t)$ is the number of DAG nodes ready to execute with the same type as node j^i.

3. Performance of executor m is taken into account. For executor m the rate of perform the node i is compared with executor $e \in N$ which is fastest executor for task i:

$$Z_i = \min_{e \in N}(exTime_e(i)) - exTime_m(i),$$

where $exTime$ is the time when the executor needs to finish the node i.

4. Sum of previous elements with coefficients:

$$result_i = wW_i + qQ_i + zZ_i.$$

Coefficients w, u , z were defined using the genetic algorithm (Carson (2017)) for 2 types of graph topologies. Random population of 100 elements is created, after testing the top 10 go to the next stage, we also take 40 crossed from the top 10, 39 crossed with additional mutations, 10 copies of the top 10 with mutations and 1 random DNA, with the target $makespan \to \min$ and stopping condition if makespan improvement with previous is less than 0.1%.

5. Probability of choosing the executor m to perform the node i is

$$P_i^m = \frac{result_i}{\left(\sum_{j \in nodesToExecute} result_j\right)}.$$

5.2 Local Voting Protocol

The state of executor m is the evaluation of the probability of choosing a node P_i^m for each DAG node i, at each moment of time. The dynamics of the state of the node i has the form:

$$x_{t+1}^m = x_t^m + f_t^m + u_t^m$$

where f_t^m is the number of new nodes of DAG were opened for execution.

At time instant t executor m sends to the $n/4$ nodes its own load x_m^t. We assume that to form the control strategy each executor m has observations about its other agents state with noise and delay:

$$y_t^{m,c} = x_{t-d_t^{m,c}}^j + w_t^{m,c},\ c \in N,$$

where $d_t^{m,c}$ is delay of transfer data, and $w_t^{m,c}$ is a noise.

The message exchange is undergone only once per system cycle. Applying the local voting algorithm, we obtain a parameter characterizing the state of executor m relative to its neighbors:

$$u_t^m = \gamma \sum_{c \in N} (y_t^{m,c} - x_t^{m,m})$$

where $\gamma > 0$ is a step-size, which represents the sensitivity of the algorithm to the difference between agents states. The consensus is achieved when the state of the last node (the node to which the message was intended) differs from its neighbors by no more than a specified one ϵ - consensus.

5.3 ALGORITHMS FOR COMPARISON

In this section consider a detailed description of the graph scheduling algorithm that we use to compare with the proposed algorithm. Degree-of-Node-First algorithm selected in accordance with the review article for graph scheduling in heterogeneous environments (Suman & Kumar (2018); Murad et al. (2022); Li et al. (2010); Kanani et al. (2015); Hayatunnufus et al. (2020)). Selection was based on following considerations: select state-of-the-art DAG scheduling (DONF) and online scheduling heuristics (Optimized Min-Min and Max-Min), as well as widely used FIFO.

- **Degree of Node First (DONF) algorithm** (Lin et al. (2019)). It is feasible to maintain higher parallelism during the scheduling process in order to make full use of heterogeneous system resources. Thus the chosen node should have the property of enlarging parallelism as much as possible. Degree-of-node scheduling procedure can be shortly described as nodes with larger out-degree should be scheduled earlier. The weighted out-degree (WOD) of node v_i is defined by:

$$WOD(v_i) = \sum_{v \in succ(s)} \frac{1}{ID(s)}, \tag{1}$$

  where $ID(s)$ is the in-degree of node s. Each executor on it's turn selects best node by DONF algorithm, it's so it will be until all nodes will be executed

- **Optimized Min-Min task scheduling algorithm** (Murad et al. (2022)). Min-Min does the opposite by choosing and assigning small tasks in resources that can perform the task with minimum execution time. The objective of the Min-Min algorithm is to ensure, firstly, that all tasks with a minimum completion time are completed. The Min-Min has the following steps:
  1. For all nodes of DAG and all executors the time of perform is calculated.
  2. Each executor on it's turn selects and executes node with the lowest execution time.
- **Max-Min Task scheduling Algorithm** (Kanani et al. (2015)). The Max-Min is quite similar with a previous algorithm, the main difference is in node selection criterion - in this case executor should select the biggest node by weight. The Max-Min has the following steps:
  1. For all nodes of DAG perform weight is calculated.
  2. Each executor on it's turn selects and executes node with the biggest perform weight.
- **FIFO** (Hayatunnufus et al. (2020)). This is the default Scheduler. The tasks are placed in a queue and the tasks are performed in their submission order. In this method, once the job is scheduled, no intervention is allowed. The FIFO algorithm has the following steps:
  1. Each node receives its place in the queue when ready for execution.
  2. Each executor on it's turn selects and executes first node in the queue.

6 NUMERICAL SIMULATION

In this section, the design of an experiment is explained and the performance of the proposed MLVP-based algorithm is evaluated. The hardware used for computations is

- Server 1: CPU AMD 9 5950x 32 threads; GPU RTX 3080 ti; RAM 128 GB.
- Server 2: CPU AMD Ryzen Threadripper 3990X 128 threads; GPU RTX 3080 ti; RAM 192 GB.

6.1 SIMULATION ENVIRONMENT

The proposed Local Voting based algorithm is tested to measure its generalization and robustness depending on different system configurations. The working environment is determined by different heterogeneous configurations from three to twenty four executors are used, 4 configurations in total. The types of executors are evenly distributed (equally for executors of types 1, 2, and 3), and their powers were set randomly from a range of 10 to 300 GFlops. Each workspace is intended for testing

on a fixed DAG dimension. The details of the configurations test cases are shown in Table 2 in Appendix G.

In the simulation, the executors inside a computer node are "fully connected": links between different executors do not obstruct each other. Links between computer nodes also have similar properties, however later data transmission starts only when all the former ones finish.

For testing and training algorithms, a real-time graph traversal C++ simulation environment was developed (example can be seen at Appendix C). The logic of the simulation is as follows: after starting the calculation, executors, working in parallel and exchanging service data, choose a node for execution, and then work on it for the time period, depending on node weight and executor power. After execution of a node the operation is repeated, and so on until the moment when all nodes in the DAG are executed.

For training and evaluation of the proposed MLVP-based algorithm batch of random DAG are used (described in Section 4). At fixed dimensions exists 48 topologies. For training, 3 graphs are generated for each topology for training (totally 144 for each dimension 3000, 6000, 12000, 24000) and 10 graphs for evaluation (totally 480 for each dimension 3000, 6000, 12000, 24000). Total of 1920 DAGs were tested. Example of DAG are shown in Fig.3. Comparative analysis is presented in the Section 7

7 RESULTS

The comparison results between MLVP-based algorithm and state-of-the-art DAG scheduling algorithms DONF, FIFO, Min-Min, Min-Max for different workspaces.

The comparison was carried out in accordance with the following metric for each tested graph:

$$p = \frac{(makespan_{\text{MLVP}} - makespan_{\text{other}}) \cdot 100\%}{makespan_{\text{other}}}, \tag{2}$$

where $makespan_{\text{other}}$ is the finish time of the last node in the scheduled DAG using one of the algorithms from the list above, $makespan_{\text{MLVP}}$ is the finish time of the last node in the scheduled DAG using the MLVP-based algorithm. Obtained values are summarized and averaged for each workspace and DAG dimension DAG. The proposed MLVP-based algorithm is tested on a large-scale DAGs and is compared to the state-of-the-art algorithms, to determine its generalization and robustness depending on dimensions DAGs and different system configurations.

Fig. 1 shows a comparison of the MLVP-based approach compared to DONF, FIFO, Min-Min, Min-Max algorithms for various test cases, where X-Axis is test cases and Y-Axis is the percentage of how much the MLVP-based exceeds/inferior compared with state-of-the-art algorithms. On average, the proposed approach outperforms the DONF by 23.99%, FIFO by 26.21%, Min-Min 26.61% Min-Max 30.81% on all test cases.

Fig. 4 in Appendix F presents box plots illustrating the percentage improvements in makespans of the MLVP algorithm relative to various scheduling algorithms—DONF, FIFO, Min-Min, and Min-Max—across different workspaces and DAG dimensions. Each workspace, differentiated by executor counts of 3, 6, 12, and 24, is analyzed separately to delineate the algorithm's performance under varying computational resources.

In the first workspace with three executors, the MLVP algorithm exhibits substantial variation in performance relative to the reference algorithms, with a wider interquartile range. Conversely, in workspaces with six, twelve, and twenty-four executors, the MLVP algorithm demonstrates more consistent improvements, as indicated by the narrower interquartile ranges.

Observing the median values, it is evident that the MLVP algorithm tends to offer improvements in makespan across all workspaces, albeit with variations in the magnitude of improvement. Particularly, the algorithm shows consistent positive enhancements compared to the FIFO, Min-Min, and Min-Max algorithms across all executor counts. In the three-executor workspace, MLVP presents around a 10% improvement over DONF and FIFO and approximately a 5% improvement over Min-Min and Min-Max. In the six-executor workspace, MLVP shows nearly 20% improvement across all compared algorithms. In larger workspaces with twelve and twenty-four executors, MLVP consistently exhibits around a 60% improvement over the reference algorithms.

The MLVP algorithm demonstrates a promising capacity to improve makespan in various workspaces across different topologies and against multiple scheduling algorithms. It exhibits consistency and adaptability across different number of executions, underlining its potential to scale effectively in heterogeneous computing environments Fig 5. Future work might focus on exploring the robustness and scalability of MLVP in more diverse and complex scheduling scenarios to further validate its applicability and performance.

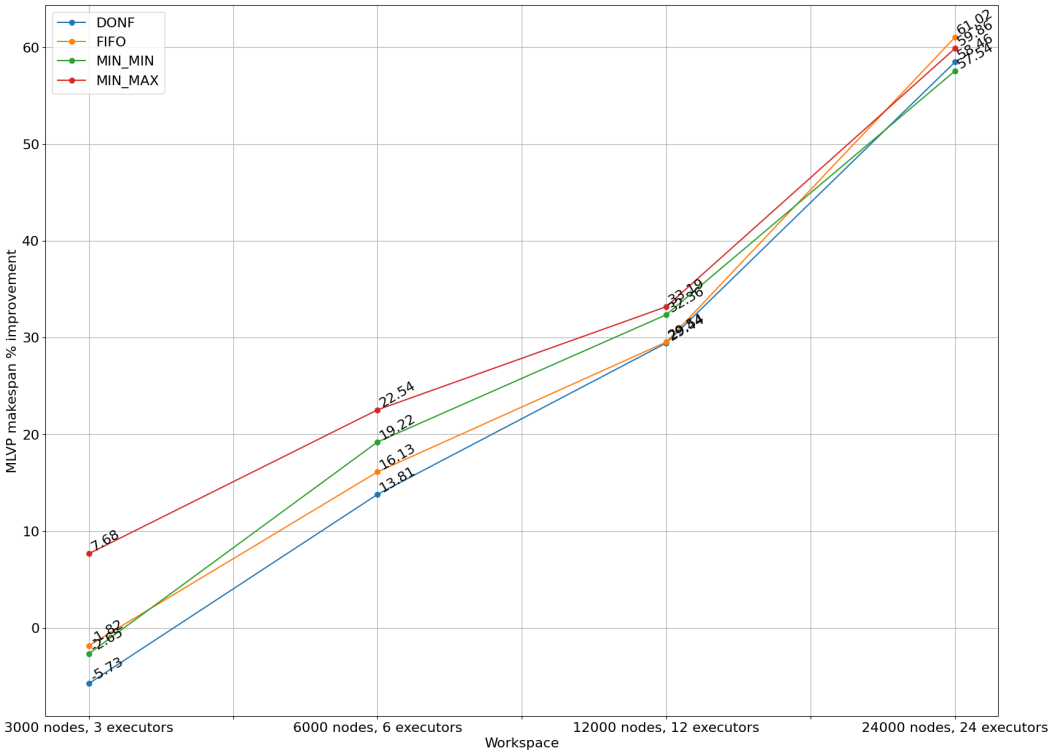


Figure 1: Percentage improvements MLVP-based approach compared to other algorithms.

8 Conclusion

This paper presents the Multi-Agent Local Voting Protocol (MLVP), a novel approach to scheduling tasks in directed acyclic graphs (DAGs) within heterogeneous computing environments. The MLVP effectively addresses the challenges of online DAG scheduling by dynamically balancing the load across multiple executors. This is achieved through a local voting protocol that calculates an aggregated metric for each executor-DAG node pair, leveraging factors such as incoming connections, available nodes, and executor performance. The optimization of combination of these metrics is performed using a genetic algorithm, allowing for probabilistic task assignment that improves makespan.

The MLVP's innovative approach demonstrates significant improvements over traditional scheduling algorithms, with simulations showing up to a 70% reduction in makespan on specific graph topologies and an average performance gain of 23.99% over the Degree of Node First (DONF) algorithm. These results highlight the MLVP's scalability and effectiveness in various system configurations, making it a promising solution for real-world applications in cloud computing, data mining, and beyond.

Looking forward, future research could explore extending the MLVP to support multi-dimensional executor requirements and handling dynamic conditions such as fluctuating numbers of executors. Additionally, further exploration into optimizing resource reservation based on anticipated critical system tasks could enhance the algorithm's adaptability and efficiency.

REFERENCES

GitHub - frs69wq/daggen: DAGGEN: A synthethic task graph generator — github.com. https://github.com/frs69wq/daggen. [Accessed 03-08-2024].

Mohammad Reza Alizadeh, Vahid Khajehvand, Amir Masoud Rahmani, and Ebrahim Akbari. Task scheduling approaches in fog computing: A systematic review. *International Journal of Communication Systems*, 33(16):e4583, 2020. doi: https://doi.org/10.1002/dac.4583. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.4583. e4583 IJCS-19-0949.R2.

Natalia Amelina, Alexander Fradkov, Yuming Jiang, and Dimitrios J. Vergados. Approximate consensus in stochastic networks with application to load balancing. *IEEE Transactions on Information Theory*, 61(4):1739–1752, 2015. doi: 10.1109/TIT.2015.2406323.

F. Bullo, J. Cortés, and S. Martínez. *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. Princeton Series in Applied Mathematics. Princeton University Press, 2009. ISBN 9781400831470.

J. Carson. *Genetic Algorithms: Advances in Research and Applications*. Computer science, technology and applications. Nova Science Publishers, Incorporated, 2017. ISBN 9781536118926.

Thomas J. Watson IBM Research Center and D.G. Feitelson. *A Survey of Scheduling in Multiprogrammed Parallel Systems*. Research report. IBM T.J. Watson Research Center, 1994. URL https://books.google.ru/books?id=4lRKHAAACAAJ.

Yubin Duan, Ning Wang, and Jie W. Reducing makespans of dag scheduling through interleaving overlapping resource utilization. In *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 392–400, 2020. doi: 10.1109/MASS50613.2020.00055.

Juan Durillo and Radu Prodan. Multi-objective workflow scheduling in amazon ec2. *Cluster Computing*, 17:169–189, 06 2013. doi: 10.1007/s10586-013-0325-0.

Ilavarasan E and Perumal Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Science*, 3, 02 2007. doi: 10.3844/jcssp.2007.94.103.

Anirban Ghose and Soumyajit Dey. Fgfs: Feature guided frontier scheduling for simt dags. *J. Supercomput.*, 78(9):11702–11743, jun 2022. ISSN 0920-8542. doi: 10.1007/s11227-022-04323-8. URL https://doi.org/10.1007/s11227-022-04323-8.

Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. *SIGCOMM Comput. Commun. Rev.*, 44(4): 455–466, aug 2014. ISSN 0146-4833. doi: 10.1145/2740070.2626334. URL https://doi.org/10.1145/2740070.2626334.

O. Granichin, Petr Skobelev, Aleksandr Lada, Igor Mayorov, and Alexander Tsarev. Comparing adaptive and non-adaptive models of cargo transportation in multi-agent system for real time truck scheduling. pp. 282–285, 01 2012.

Hayatunnufus Hayatunnufus, Mardhani Riasetiawan, and Ahmad Ashari. Performance analysis of fifo and round robin scheduling process algorithm in iot operating system for collecting landslide data. pp. 63–68, 07 2020. doi: 10.1109/DATABIA50434.2020.9190608.

Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. Network-aware scheduling for data-parallel jobs: Plan when you can. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pp. 407–420, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450335423. doi: 10.1145/2785956.2787488. URL https://doi.org/10.1145/2785956.2787488.

Bhavisha Kanani, Bhumi Maniyar, and Sameer Mohammad. Review on max-min task scheduling algorithm for cloud computing. *SSRN Electronic Journal*, 2:781–784, 03 2015.

Soummya Kar and José M. F. Moura. Distributed consensus algorithms in sensor networks: Quantized data and random link failures. *IEEE Transactions on Signal Processing*, 58(3):1383–1400, 2010. doi: 10.1109/TSP.2009.2036046.

Wael Khallouli and Jingwei Huang. Cluster resource scheduling in cloud computing: literature review and research challenges. *The Journal of Supercomputing*, 78, 04 2022. doi: 10.1007/s11227-021-04138-z.

Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: Comparing public cloud providers. pp. 1–14, 11 2010. doi: 10.1145/1879141.1879143.

Ziyang Li, Jiong Yu, Chen Bian, Yonglin Pu, Yuefei Wang, Yitian Zhang, and Binglei Guo. Flinker: An elastic resource-scheduling strategy for processing fluctuating mobile stream data on flink. *Mobile Information Systems*, 2020:1–17, 05 2020. doi: 10.1155/2020/5351824.

Han Lin, Ming-Fan Li, Cheng-Fan Jia, Jun-Nan Liu, and Hong An. Degree-of-node task scheduling of fine-grained parallel programs on heterogeneous systems. *Journal of Computer Science and Technology*, 34:1096–1108, 09 2019. doi: 10.1007/s11390-019-1962-4.

Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, pp. 50–56, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450346610. doi: 10.1145/3005745.3005750. URL https://doi.org/10.1145/3005745.3005750.

Sallar Murad, Rozin Abdal, Nashat Salih, Abdulkarim Alsandi, Rafi Faraj, Abdullah Muhammed, Mohd Derahman, and Reham Ahmed. Optimized min-min task scheduling algorithm for scientific workflows in a cloud environment. *Journal of Theoretical and Applied Information Technology*, 100:480–506, 01 2022.

Jean-Noël Quintin and Frédéric Wagner. Wscom: Online task scheduling with data transfers. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pp. 344–351, 2012. doi: 10.1109/CCGrid.2012.21.

Wei Ren, Randal W. Beard, and Ella M. Atkins. Information consensus in multivehicle cooperative control. *IEEE Control Systems Magazine*, 27(2):71–82, 2007. doi: 10.1109/MCS.2007.338264.

G.C. Sih and E.A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2): 175–187, 1993. doi: 10.1109/71.207593.

Mbarka Soualhia, Foutse Khomh, and Sofiène Tahar. Predicting scheduling failures in the cloud: A case study with google clusters and hadoop on amazon emr. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pp. 58–65, 2015. doi: 10.1109/HPCC-CSS-ICESS.2015.170.

Chandresh Suman and Gaurav Kumar. Analysis of process scheduling algorithm for multiprocessor system. In *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 564–569, 2018. doi: 10.1109/ICRITO.2018.8748657.

H. Topcuoglu, S. Hariri, and Min-You Wu. Task scheduling algorithms for heterogeneous processors. In *Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)*, pp. 3–14, 1999. doi: 10.1109/HCW.1999.765092.

Dimitrios J. Vergados, Natalia Amelina, Yuming Jiang, Katina Kralevska, and Oleg Granichin. Toward optimal distributed node scheduling in a multihop wireless network through local voting. *IEEE Transactions on Wireless Communications*, 17(1):400–414, 2018. doi: 10.1109/TWC.2017.2767045.

Csaba Virágh, Gábor Vásárhelyi, Norbert Tarcai, Tamás Szörényi, Gergő Somorjai, Tamás Nepusz, and Tamás Vicsek. Flocking algorithm for autonomous flying robots. *Bioinspiration Biomimetics*, 9, 10 2013. doi: 10.1088/1748-3182/9/2/025012.

Ying Yan, Yanjie Gao, Yang Chen, Zhongxin Guo, Bole Chen, and Thomas Moscibroda. Tr-spark: Transient computing for big data analytics. 10 2016. doi: 10.1145/2987550.2987576.

Wenwu Yu, Guanrong Chen, and Ming Cao. Distributed leader-follower flocking control for multi-agent dynamical systems with time-varying velocities. *Systems Control Letters*, 59:543–552, 01 2010.

A DAG GENERATION PARAMETERS

Table 1: DAG Generation Parameters

Parameter	Description
n	Number of computation nodes in the DAG (i.e., application "tasks"). **Values:** 3000, 6000, 12000, 24000.
fat	Width of the DAG, the maximum number of tasks that can be executed concurrently. A small value will lead to a thin DAG (e.g., chain) with low task parallelism, while a large value induces a fat DAG (e.g., fork-join) with high parallelism. **Values:** 0.2, 0.5.
density	Determines the number of dependencies between tasks of two consecutive DAG levels. **Values:** 0.1, 0.4, 0.8.
regularity	Regularity of the distribution of tasks between the different levels of the DAG. **Values:** 0.2, 0.8.
jump	Maximum number of levels spanned by inter-task communications. This allows generating DAGs with execution paths of different lengths. **Values:** 2, 4.
ccr	The ratio of the sum of edge weights to the sum of node weights. **Values:** 0.2, 0.8.

B MLVP PSEUDOCODE

Algorithm 1 Multi-Agent Local Voting Protocol (MLVP)

Input: Set of executors E, Set of DAG nodes N
Output: Assignment probabilities for each DAG node to each executor
Initialize $assignmentProbabilities$ to an empty data structure
for each executor $e \in E$ **do**
  for each node $n \in N$ **do**
    if n is not finished and n is ready for execution **then**
      $incomingConnections \leftarrow$ Sum of weights of all incoming edges to n
      $availableNodes \leftarrow$ Count of nodes ready for execution of the same type as n
      $executorPerformance \leftarrow$ Performance metric of executor e
      $aggregatedMetric \leftarrow$ LinearCombination($incomingConnections$, $availableNodes$, $executorPerformance$)
      $assignmentProbabilities[e][n] \leftarrow$ CalculateProbability($aggregatedMetric$)
    end if
  end for
end for
Find coefficients for LinearCombination function based on historical data
for each node $n \in N$ **do**
  if n is ready for execution **then**
    for each executor $e \in E$ **do**
      Tune $assignmentProbabilities[e][n]$ using genetic algorithm to balance workload
    end for
  end if
end for
return $assignmentProbabilities$

C SIMULATION EXAMPLE

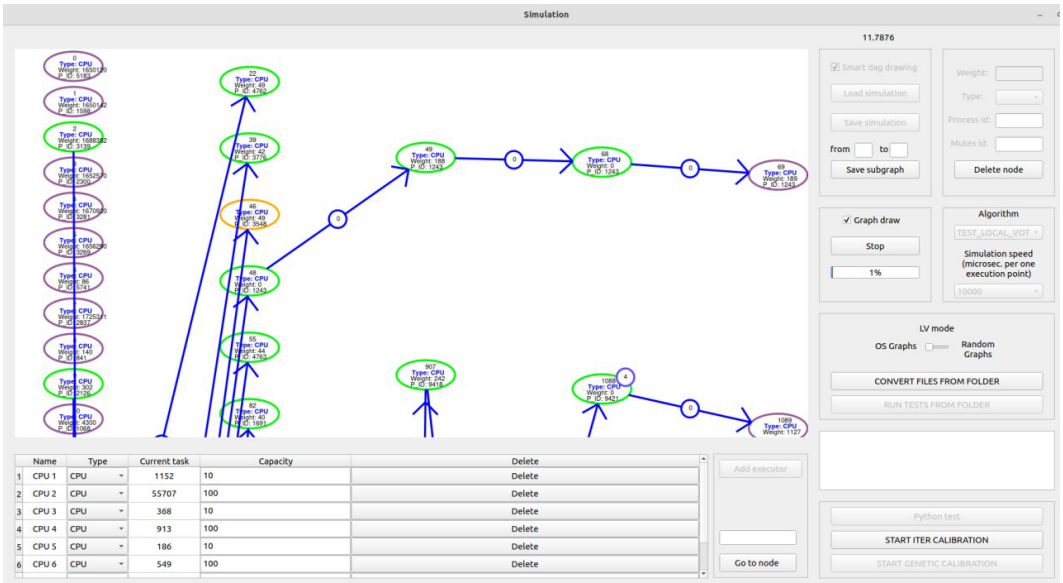


Figure 2: An example of a C++ simulation environment

D SIMULATION CONFIGURATIONS

A total of 4 test cases were identified for large-scale DAGs:

1. 3 executors, 1 executor of each type (1 for 1 type, 1 for 2 type, 1 for 3 type) which differ in computational capabilities: randomly generated from 10 to 300 GFlops. Testing is performed on DAGs of dimension 3000 nodes.

2. 6 executors, 2 executor of each type (2 for 1 type, 2 for 2 type, 2 for 3 type) which differ in computational capabilities: randomly generated from 10 to 300 GFlops. Testing is performed on DAGs of dimension 6000 nodes.

3. 12 executors, 4 executor of each type (4 for 1 type, 4 for 2 type, 4 for 3 type) which differ in computational capabilities: randomly generated from 10 to 300 GFlops. Testing is performed on DAGs of dimension 12000 nodes.

4. 24 executors, 8 executor of each type (8 for 1 type, 8 for 2 type, 8 for 3 type) which differ in computational capabilities: randomly generated from 10 to 300 GFlops. Testing is performed on DAGs of dimension 24000 nodes.

Table 2: Description of Test Cases for Large-Scale DAGs

Workspace ID	Total number of executors	Number of executors of each type	Dimension DAGs
1	3	1	3000
2	6	3	6000
3	12	4	12000
4	24	8	24000

E DAG EXAMPLE

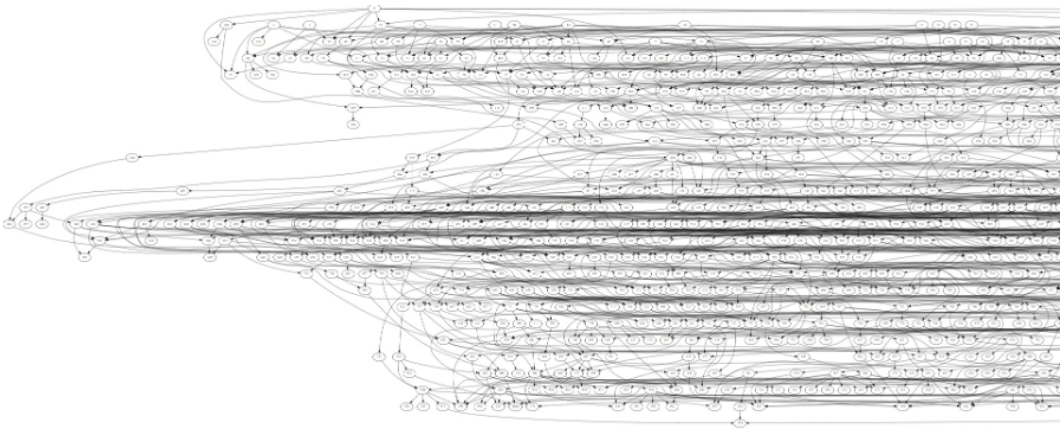
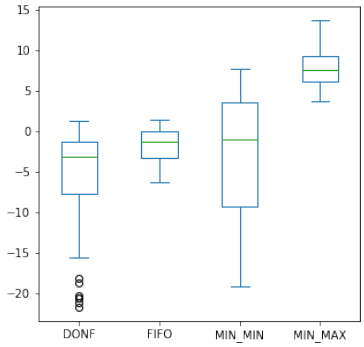
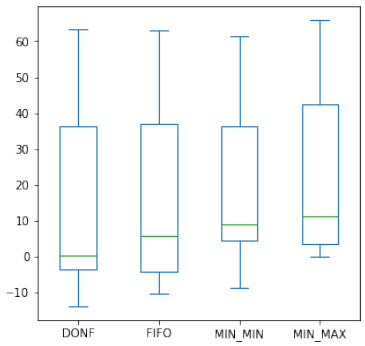


Figure 3: Examples of large-scale DAG

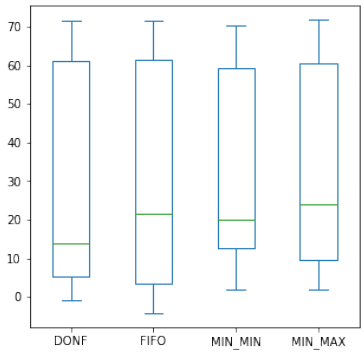
F BOX PLOTS



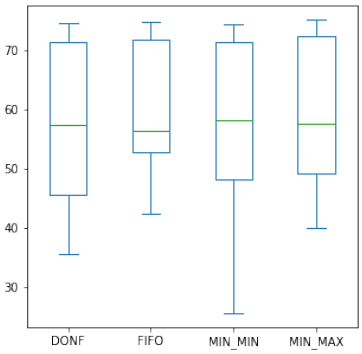
Workspace 1: 3 executor



Workspace 2: 6 executor



Workspace 3: 12 executor



Workspace 4: 24 executors

Figure 4: Box plot percentage improvements in makespans MLVP compared to DONF, FIFO, Min-Min,Min-Max for each workspace and each DAG dimension.

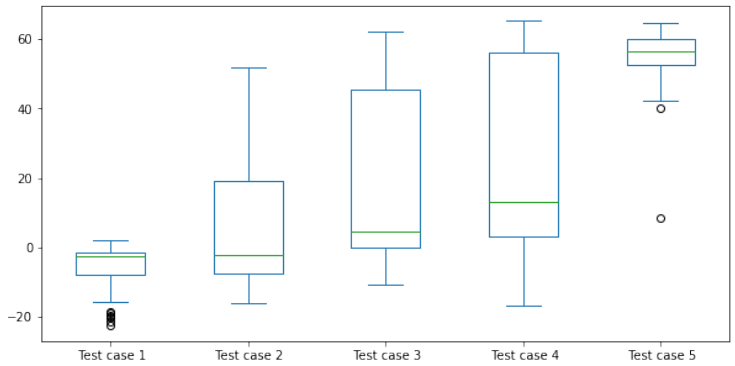


Figure 5: Box plot with MLVP and DONF comparisons for each workspace

G COMPLEXITY

Consider following table, where v is the number of nodes in DAG, p is the number of executors, o where is the maximum out-degree of any node and k is the number of maximum number of ready-to-execute nodes at any time.

Table 3: Time Complexity of Various Algorithms

Algorithm	**Time Complexity**
FIFO	$O(v)$
Optimized Min-Min	$O(v^2)$
Max-Min	$O(v^2)$
DONF	$O(v \times o \times p)$
MLVP	$O(k \times o \times p^2)$

MLVP differs mainly in the stage of executor selection, where executors need to pass information between each other in order to facilitate consensus and information exchange occurs only one once per scheduling cycle. We use k instead of v primarily because each executor selects tasks (nodes) to be scheduled after consensus is reached simultaneously, i.e. we are processing batch of nodes at the same time. Worst case scenario can be considered in two aspects: if DAG is akin to a circle, i.e. all nodes are ready to be executed from the start, that will yield complexity of $O(v \times o \times p^2)$. And if out-degree of nodes is as large as $v - 1$ complexity will become $O(v^2 \times p^2)$, which is worse than DONF worst-case scenario of $O(v^2 \times p)$.