Dynamic Resource Scheduler for Distributed Deep Learning Training in Kubernetes

Muhammad Fadhriga Bestari
School of Electrical Engineering and Informatics, ITB, Indonesia
fadhriga.bestari@gmail.com

Achmad Imam Kistijantoro[1,2]
[1]School of Electrical Engineering and Informatics, ITB, Indonesia
[2]University Center of Excellence on Artificial Intelligence for Vision, Natural Language Processing & Big Data Analytics (U-CoE AI-VLB), Indonesia
imam@stei.itb.ac.id

Anggrahita Bayu Sasmita
School of Electrical Engineering and Informatics, ITB, Indonesia
angga@stei.itb.ac.id

2020 7th International Conference on Advance Informatics: Concepts, Theory and Applications (ICAICTA) | 978-1-7281-8038-0/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICAICTA49861.2020.9429033

***Abstract*—Distributed deep learning is a method of machine learning that is used today due to its many advantages. One of the many tools used to train distributed deep learning model is Kubeflow, which runs on top of Kubernetes. Kubernetes is a containerized application orchestrator that ease the deployment process of applications. This in turn makes distributed deep learning training done in Kubeflow easier and manageable. Works on a dynamic resource scheduler in Kubernetes for deep learning training have been done before, such as DRAGON that proposed scheduler with autoscaling and gang scheduling capabilities, and OASIS that proposed a utility system with price function. In this work, we propose to combine DRAGON's and OASIS' approach to make a scheduler with weighted autoscaling capabilities and schedule its jobs with gang scheduling. Some modifications are done on DRAGON's autoscaling function. We try to increase the frequency of scaling up function calls and reduce the frequency of scaling down function to make the training process more efficient. Weights are used to determine the priority of each jobs, where jobs with higher resource requirements are considered more important. Weight of each jobs will influence the autoscaling function of the scheduler. Experiment and evaluation done using a set of Tensorflow jobs results in an increase of training speed by over 26% in comparison with the default Kubernetes scheduler.**

Keywords—Kubernetes, Kubeflow, deep learning job, job scheduling, scale up, scale down, gang scheduling, weighted job

I. INTRODUCTION

Deep learning is a subset of machine learning that similarly to humans, uses neurons to learn. These neurons combined creates a neural network that receive data, process it, and generate a model that can be used to predicts new data based on the data used in the training process. Deep learning is more widely used nowadays due to its generated models having a more flexible, scalable and available in comparison with other machine learning methods [9].

To generate a model that's able to predict real data requires a large amount of data to train with. This large data requirement for deep learning increase the model training time considerably. Number of layers in the neural network and epochs also determines the training speed of the model. Complex deep learning training tends to have a larger amount of data and higher number of layers and epochs, resulting in a lower training speed.

Kubernetes [1] is a platform that manages deployments for containerized applications. Kubeflow [3] is a tool developed on top of Kubernetes that allows distributed deep learning training to be done on Kubernetes with relative ease. However, Kubernetes scheduling functionality are not specifically designed to do deep learning training, resulting many problems during the scheduling process of deploying distributed jobs on Kubernetes.

DRAGON [4] is a resource scheduler that schedules distributed jobs using gang scheduling and autoscaling that improves upon the scheduling capabilities of Kubernetes. In this work we propose to further improve on DRAGON's approach by weighting each job to determine its priority and integrated the weighting system with the autoscaling function.

The scheduler prioritizes jobs with higher resource requirements to maximize resource utilization without the need to call scale up function. We try to schedule higher priority jobs first to reduce the need to call scale down function as well. Autoscaling functions cause overhead in the system, so the reduce in number of calls of these functions may improve the overall training speed.

This paper follows the following structure. In Section 2, we discuss related works used as a foundation for our work. Details of our implementation is discussed in Section 3, followed by experiments and evaluation in Section 4, and conclusions in section 5.

978-1-7281-8038-0/20/$31.00 ©2020 IEEE

II. Related Works

A. DRAGON

DRAGON is an extended controller for managing jobs on Kubernetes [4]. DRAGON uses two approach to increase scheduling efficiency in Kubernetes, which is autoscaling and gang scheduling. Autoscaling is useful because system loading changes as new jobs enter the system and old jobs finish training. DRAGON utilize scale up function to maximize resource utilization when system loading is low, and utilize scale down function to reduce resource utilization when system loading is high. Although scale down function reduce resource utilization, it allows other jobs to be scheduled, possibly increasing the overall resource utilization.

Gang scheduling is a scheduling method that DRAGON use to reduce communication overhead in the system. Distributed deep learning divide training process into two entities, i.e. parameter server and workers. During the training process, parameter server and workers needs to keep exchanging gradients and weights. Gang scheduling is a scheduling algorithm that schedule parameter server and workers in the same node in Kubernetes, reducing the communication overhead needed for parameter server and workers to synchronize during training.

DRAGON, unlike Kubernetes' default scheduler, uses an Adapted First Come First Served (AFCFS) scheduling algorithm. AFCFS still queue jobs according to its arrival time, but different from regular First In First Out (FIFO) scheduling algorithm, it schedules the first feasible job in the queue. DRAGON prioritizes jobs that come first, but doesn't guarantee that jobs will be scheduled in order.

DRAGON's control flow design is detailed on fig. 1. First, scheduler waits until job enters the system. It queues jobs according to its arrival time, where jobs that arrive first is prioritized. The scheduler then checks whether an urgent job that has been waiting inside the queue for more than the assigned threshold is present or not. If it doesn't, then the scheduler will find schedulable job using AFCFS and schedule that job using gang scheduling. However, if it found an urgent job in the queue, it will try to schedule the urgent job. If it failed, the scheduler will call scale down function to reduce system loading so the urgent job can be scheduled.

Similar to scale down function, scale up functionality is called if the assigned threshold for scale up function has been reached. Different from scale down function, scale up function timer is not tied to the jobs, rather to how long to system has been in an idle state.

B. OASIS

OASIS [5] is a scheduling algorithm that's implemented as a custom scheduler on Kubernetes. It utilizes a different scheduling algorithm than the default FIFO approach. OASIS calculates the optimal job to be scheduled in a given queue. It does so by calculating required resource and potential utility of each job.

OASIS calculate a payoff of each job in the queue using the job's utility and a price function. OASIS will maximize the payoff of jobs that are scheduled. Jobs that have payoff less than zero will be declined, whereas jobs that have payoff higher than zero can be considered to be scheduled.

The price function dynamically changes depending on the availability of resource. The lower the resource availability, the higher the minimum price of a job to be scheduled. Hence, as the resource availability decreases, the scheduler will prioritize jobs with higher utility.

OASIS proposes three rules regarding price function algorithm.

1. Price function have a minimum value that allows every job to be scheduled. The lowest value of the job utility needs to at least the same as the initial price function value. This ensure that every job can be scheduled at some point.
2. Price function increases exponentially when allocated resource increases so that the scheduler can deny jobs with low utility value and save available resource for jobs with high utility that may come later.
3. Price function have a maximum value that denies any job. This ensure that every job will be denied when there is no available resource to be allocated for jobs in the queue.

III. Proposed Method

We proposed a further development in dynamic resource scheduler by combining both approach of DRAGON and OASIS, where we add a weighting algorithm inspired from OASIS in deep learning jobs and integrate it with autoscaling algorithm inspired by DRAGON. The architecture of our proposed method can be seen in fig. 2.

As jobs enter the system, our scheduler will add them into the queue using priority queueing, where jobs with higher resource requirements. Scheduler then checks for jobs with high priority inside the queue. If the scheduler doesn't find a high priority job, it will find the first feasible job using AFCFS scheduling algorithm and use gang scheduling to schedule the job.

Differently from DRAGON, we proposed to remove the threshold waiting time for autoscaling and call it as needed. We propose for scale up to be called every time there is no available feasible job to be scheduled. The overhead of restarting jobs for scaling up, we found, is relatively low in comparison with the increase of resource utilization.

If the scheduler found a high priority job in the queue, it will try to schedule the high priority job first. If it failed, then the scheduler will call scale down function until the high priority job can be scheduled. However, if the scale down function still fails to enable to high priority job to be scheduled, the scheduler will call scale up function to maximize the remaining resource that's available.

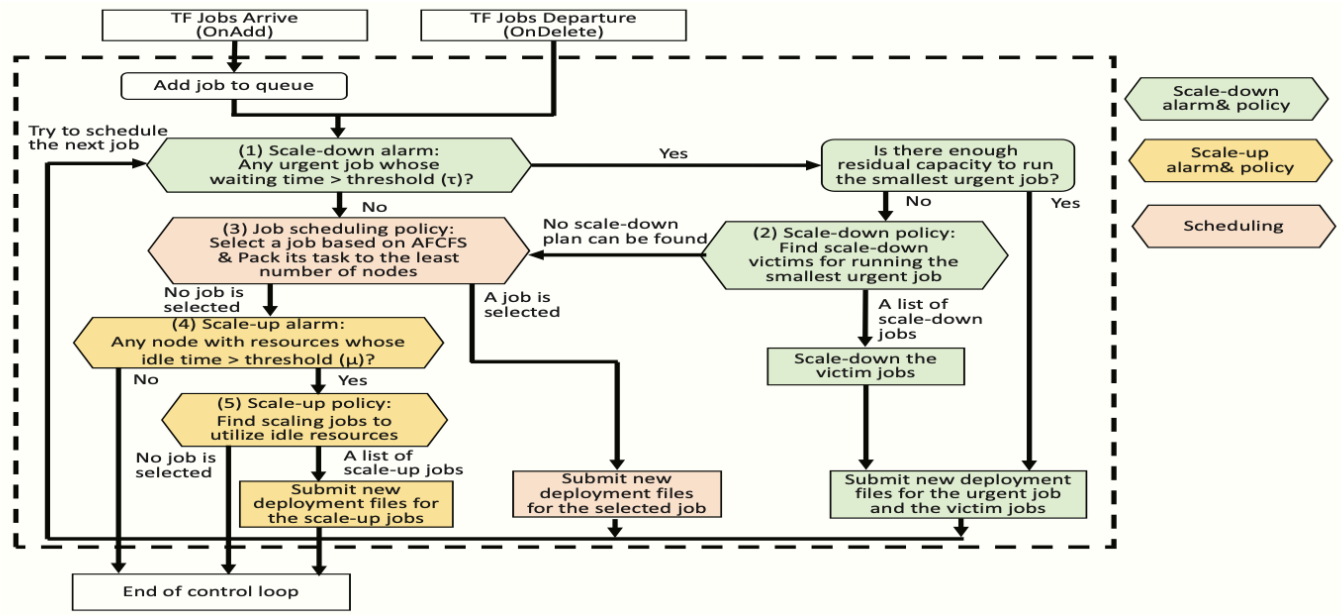

Fig. 1. Control flow and policy of DRAGON [4]

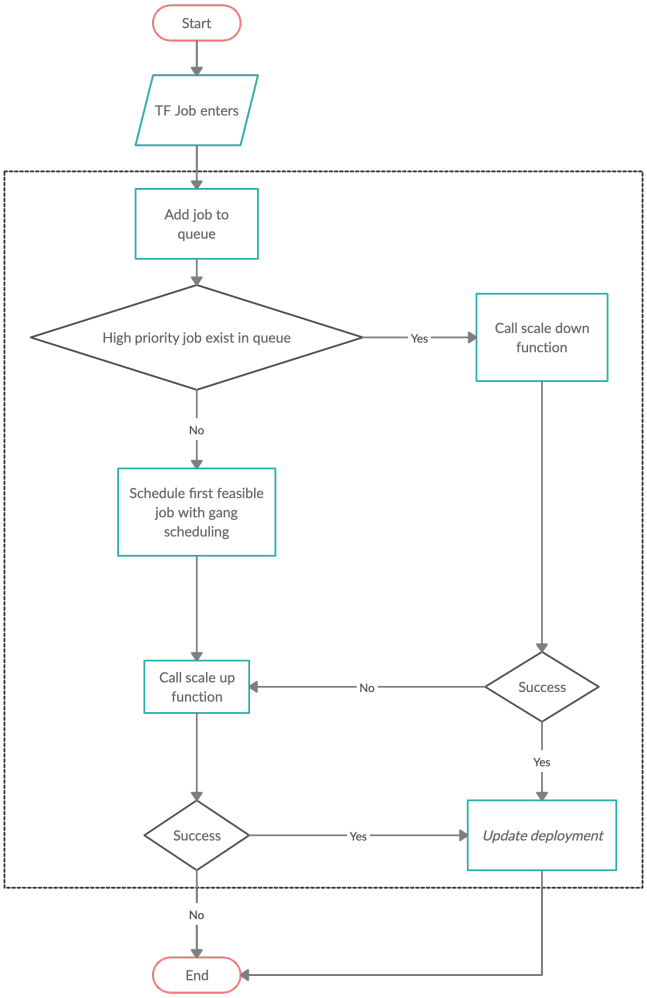

Fig. 2. Proposed system architecture

A. *Weighting Algorithm*

Our proposed scheduler utilizes two weighting method. First, the scheduler calculates the weight of each job using the resource requirements. Scheduler maximizes resource utilization by prioritizing jobs with high resource requirements first. By doing so, scheduler can maximize resource utilization without the need of calling scale up function.

Second, scheduler can prioritize jobs by using a given priority value by the user. This weighting method gives the scheduler flexibility to be able to adhere to user's specifications. Scheduler will first prioritize jobs with higher priority value assigned by users, then sort the jobs based on the resource requirements if two or more jobs have the same priority.

The weighting algorithm used in our proposed method is displayed in (1) and the notation description is detailed in Table I.

$$f(\gamma_1, \gamma_2) = \frac{\gamma_1}{1+ e^{\gamma_2}} \quad (1)$$

TABLE I. NOTATION DESCRIPTION FOR WEIGHTING ALGORITHM

Notation	Description
γ_1	Priority of job [1,100]
γ_2	Decay factor of job [0,1]

B. *Gang Scheduling Algorithm*

Gang scheduling is done by grouping all pod replicas of a job. The algorithm will calculate the aggregate resource requirements of all replicas for that job. It will then iterate every node present in the cluster to find a node that has the resource

availability to satisfy the job resource requirements. If there were no node that can satisfy all replica of the job, then the scheduler won't force the gang scheduling to be done, but rather scheduling the replicas in the available nodes.

C. Scale Down Function

Scale down function is used to reduce the number of replicas of a job that's currently being trained. Scale down function will iterate every job that's currently being trained and tries to reduce its number of replicas. If a job has more replica than the minimum number of replicas, then the number of replicas for that job will be reduced.

The iteration will continue until the available resource can satisfy the resource requirement of the high priority job. If after all jobs' replicas have been reduced to its minimum values and the high priority job still can't be scheduled, then the scale down function will result in a failure state. If it is successful, the scheduler will reduce the selected jobs' replicas and schedule the high priority job.

D. Scale Up Function

Scale up function is used to increase the number of replicas of a job that's currently being trained. Scale up function will iterate every job that's currently being trained and tries to increase its number of replicas. If a job has less replica than the maximum number of replicas, then the number of replicas for that job will be increased. However, if the available resource can't satisfy the addition of replicas, then scale up function won't be called.

IV. Evaluation

A. Experimental Environment

The purpose of our experiments is to measure the training speed of deep learning jobs using our scheduler, DRAGON, and Kubeflow's default tf-operator. We evaluate both DRAGON and default tf-operator to compare our scheduler's performance with a scheduler from the related works section and also the controller Kubeflow use today. To evaluate this, we constructed a few experiments with different job specifications to evaluate the schedulers in different job environments. All experiments are done on Google Kubernetes Engine with 3 nodes. Each node has 8 vCPU and 30GB Memory, totaling to 24 vCPU and 90 GB Memory in the cluster.

Each experiment scenario is done multiple time to avoid fluke result. The job that was used for our experiments are a linear regression model using MNIST dataset. Specifications of jobs that are modified for each experiment mostly revolves around maximum number of workers, initial number of workers, CPU requirement, Memory requirement, and value of job priority.

For every job in the experiments, we assume its time criticality to be not critical. Hence, all jobs in the experiments have a time critical value of 0. This in turn, cause the priority and resource requirements of a job to be the only things that affect the weighting algorithm.

B. Autoscaling and Gang Scheduling Evaluation

We conducted 3 experiments to evaluate the effects of autoscaling and gang scheduling on the training speed of deep learning jobs. Jobs specifications for each experiment is detailed on Table II, Table III, and Table IV. These experiments are done to compare the training speed of our scheduler, DRAGON, and tf-operator in three different scenarios.

TABLE II. Job Spesification for Scenario 1

Category	Job 1	Job 2	Job 3	Job 4
Maximum workers	4	6	6	6
Minimum workers	1	2	1	2
Initial worker replicas	2	2	2	2
CPU requirement	1	2	2	3
Memory requirement	2 Gi	4 Gi	4 Gi	6 Gi
Priority	1	1	1	100

TABLE III. Job Spesification for Scenario 2

Category	Job 1	Job 2	Job 3	Job 4
Maximum workers	4	6	6	6
Minimum workers	1	2	1	2
Initial worker replicas	2	2	2	2
CPU requirement	1	1	1	1
Memory requirement	2 Gi	2 Gi	2 Gi	2 Gi
Priority	1	1	1	100

TABLE IV. Job Spesification for Scenario 3

Category	Job 1	Job 2	Job 3	Job 4
Maximum workers	4	6	6	6
Minimum workers	1	2	1	2
Initial worker replicas	2	2	2	2
CPU requirement	3	3	3	3
Memory requirement	6 Gi	6 Gi	6 Gi	6 Gi
Priority	1	1	1	100

C. Weighting Function Evaluation

We conducted 1 experiment to evaluate the effects of weighting function on the training speed of deep learning jobs. Jobs specifications for the experiment is detailed on Table V. This experiment is done to compare schedulers with and without weighting functionality.

TABLE V. JOB SPESIFICATION FOR SCENARIO 4

Job	Maximum Workers	Minimum Workers	Initial Workers	CPU requirement	Memory requirement	Priority
1	4	2	4	2	4	1
2	4	2	4	2	4	1
3	4	1	2	2	4	1
4	4	1	2	3	6	1
5	4	2	3	2	4	1
6	4	2	2	2	4	1
7	6	2	2	2	4	1
8	6	1	1	2	4	1
9	4	2	2	2	4	1
10	4	2	2	4	8	1
11	4	3	3	2	4	1
12	4	3	3	2	4	1

D. Model Accuracy Evaluation

Lastly, we also conducted an experiment to evaluate the model generated from distributed deep learning job. Job specifications for the experiment is detailed on Table VI. In this experiment we compare models generated using distributed deep learning job and centralized deep learning job.

TABLE VI. JOB SPESIFICATION FOR SCENARIO 5

Variable	Value
Batch size	100
Learning rate	0.05
Global step	100000

E. Evaluation Result

The result of autoscaling and gang scheduling, weighting function, and model accuracy evaluations are displayed on fig.3, Table VII, and Table VIII respectively.

Evaluation result of each scenario shows that for autoscaling and gang scheduling evaluation, on the first scenario, our scheduler resulted in an increase of training speed up to 5.06%. On the second scenario, our scheduler resulted in an increase of training speed up to 35.05%. On the third scenario, our scheduler resulted in an increase of training speed up to 22.38%.

On the first experiment, the difference of training speed between the three schedulers are not that noticeable. This is caused due to job 4, the last job in the queue, having a relatively high resource requirement. Job 4 most likely can't be scheduled using gang scheduling, as other jobs have utilized most of the available resource. Higher resource requirements also hinder the scale up function, as the scheduler may not be able to scale up function successfully even though there are still some available resource left.

On the second experiment, we observed the highest speed increase between our scheduler, DRAGON, and tf-operator. Low resource requirement for each job enables the scheduler to maximally utilize the scale up function to increase the resource utilization when training. Low resource requirements for each job also increase the probability of job replicas to be placed in the same node, since each node can schedule a handful number of replicas. Unsurprisingly, this experiment also resulted in the fastest training speed for every scheduler in comparison with experiment one and three.

On the third experiment, even though we have a relatively high-speed increase between our scheduler, DRAGON, and tf-operator, the third experiment resulted in the lowest training speed for every scheduler in comparison with the previous experiments. This experiment shows that having jobs with high resource requirements will not always improves the training speed. The communication overhead between parameter server and workers will be higher, resulting an overall slower training speed.

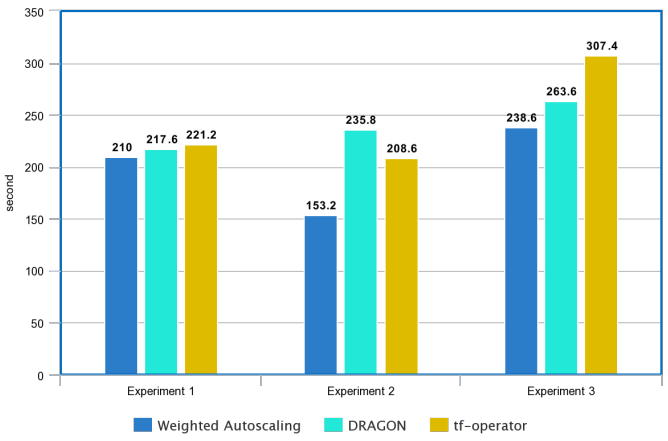

Fig. 3. Evaluation result for experiment 1, 2, and 3. Lower value is better.

Evaluation result of the fourth experiment shows that for weighting function evaluation, scheduler with weighting functionality resulted in an increase of training speed up to 6.74% in comparison with a scheduler that has no weighting functionality. This shows that prioritizing jobs that has higher resource requirements first results in a faster training speed. This is caused due to schedulers with weighting functionality can maximize available resource without calling scale up function that case a little bit of overhead in the system.

Evaluation result of the fifth experiment shows that for model accuracy, our scheduler still falls behind centralized deep learning training. The maximum accuracy our scheduler can achieve is merely a 70%. Also, the accuracy seems to be fluctuating and our scheduler still unable to generate a consistent model. The tradeoff between training speed and model accuracy needs to be heavily considered, and should be addressed in further development for our work

TABLE VII. EVALUATION RESULT FOR EXPERIMENT 4

i	*Scheduler* without Weighting (s)	*Scheduler with* Weighting (s)	Speed Increase (%)
1	654	564	13.76
2	637	670	-4.93
3	667	680	-1.95
4	671	511	23.85
5	675	673	2.96
Mean	660.8 ± 15.47	619.6 ± 77.34	6.74 ± 11.92

TABLE VIII. EVALUATION RESULT FOR EXPERIMENT 5

i	Distributed (%)				Centralized (%)
	1 worker	2 workers	3 workers	4 workers	
1	32	51	39	53	91
2	40	28	60	70	90
3	24	62	52	52	90
4	15	57	42	51	91
5	31	38	38	40	92
Mean	28.4 ± 9.3	47.2 ± 14	46.2 ± 9.5	53.2 ± 11	90.8 ± 0.8

V. CONCLUSION

Our proposed method of combining DRAGON's and OASIS' approach by creating a scheduler with weighting, autoscaling, and gang scheduling capabilities resulted in an increase of speed for deep learning training jobs. Evaluation done using MNIST dataset to each functionality shows that an increase of up to 26.56% is achieved due to the autoscaling and gang scheduling functionalities, while the weighting algorithm also resulted in an increase of training speed up to 6.23%.

Further improvement to our work should be focused on improving weighting algorithm to also consider the estimation of training time for each job. Improvement on the generated model should also be done, so that our scheduler can generate models with comparable quality with models generated using centralized deep learning.

ACKNOWLEDGEMENT

We are grateful for receiving funding for publication from the P3MI program at Institut Teknologi Bandung.

REFERENCES

[1] Kubernetes. (2019). Accessed on 8 Oktober, 2019, from https://www.kubernetes.io/.

[2] Docker. (2019). Accessed on 8 Oktober, 2019, from https://www.docker.com/.

[3] Kubeflow. (2019). Accessed on 12 Oktober, 2019, dari https://www.kubeflow.org/.

[4] Lin, C., Yeh, T., dan Chou, J. (2019). DRAGON: A Dynamic Scheduling and Scaling Controller for Managing Distributed Deep Learning Jobs in Kubernetes Cluster. DOI:10.5220/00077007605690577.

[5] Bao, Y., Peng, Y., Wu, C., dan Li, Z. (2018). Online Job Scheduling in Distributed Machine Learning Clusters. CoRR, abs/1801.00936.

[6] Rodriguez, M. dan Buyya, R. (2018). Containers Orchestration with Cost-Efficient Autoscaling in Cloud Computing Environments. CoRR, abs/1812.00300.

[7] Jeon, M., Venkataraman, S., Phanishayee, A., Qian, J., Xiao, W., dan Yang, F. (2018). Multi-tenant GPU Clusters for Deep Learning Workloads: Analysis and Implications. Microsoft Research Technical Report.

[8] Amaral, M., Polo, J., Carrera, D., Seelam, S., dan Steinder, M. (2017). *Topology-Aware GPU Scheduling for Learning Workloads in Cloud Environments.* DOI:10.1145/3126908.3126933.

[9] Peteiro-Barral, D. dan Guijarro-Bernidas, B. (2013). *A Survey of Methods for Distributed Machine Learning.* DOI:10.1007/s13748-012-0035-5.