SCHED^2: Scheduling *D*eep Learning Training via *D*eep Reinforcement Learning

Yunteng Luan, Xukun Chen, Hanyu Zhao, Zhi Yang, Yafei Dai
Computer Science Department, Peking University
{luanyunteng, chenxukun, zhaohanyu, yangzhi, dyf}@pku.edu.cn

***Abstract*—Today's companies and organizations build GPU clusters for efficient deep learning training (DLT). However, the inherent heterogeneity of DLT workloads makes it challenging to perform efficient scheduling of the GPUs. On one hand, DLT jobs typically exhibit diverse performance sensitivity to GPU locality; the scheduler should allocate GPUs with appropriate degree of locality for better performance and utilization. On the other hand, DLT jobs are also diverse in terms of size and duration, which can lead to severe cluster fragmentation and less chance for finding GPUs with good locality.**

In this paper, we present SCHED^2, a GPU cluster scheduler that leverages deep reinforcement learning (DRL) to perform smart locality-aware scheduling of DLT jobs. This is achieved by a novel design which captures both jobs' locality-sensitivity and cluster fragmentation condition in the whole learning stack, *i.e.,* from job and cluster state definitions to the neural network architecture. Through this awareness, the DRL model can adjust its scheduling decisions dynamically and adaptively, to react to individual jobs' different locality-sensitivity and changing cluster fragmentation level. Experiments using realistic workloads demonstrate that SCHED^2 reduces average JCT by 4.6x and makespan by 2.1x, compared to heuristic-based schedulers.

I. INTRODUCTION

We have witnessed the great success by deep learning (DL) in many practical domains such as image classification [1], recommendation [2] and machine translation [3]. DL usually trains an extremely complex model using a huge amount of data to achieve high accuracy, hence is compute-intensive and heavily reliant on powerful accelerators like GPUs.

Today's companies and organizations build large-scale GPU clusters to facilitate efficient training. In such clusters, efficient scheduling of the expensive GPU resources is critical: as a deep learning training (DLT) job usually takes hours or days to run, a bad scheduling decision could lead to poor training performance and low GPU utilization for a long period.

The key challenge of scheduling DLT jobs is the performance sensitivity to *locality* of GPUs. For example, a job could run much faster with localized GPUs (*e.g.,* located in the same server), than with GPUs distributed in multiple servers. Such sensitivity is complex: it depends on many factors such as model architecture, batch-size and allocated resource amount, *etc.,* which make it impractical to estimate a job's actual performance a priori. And this sensitivity should be taken into account when scheduling different types of jobs.

We find that the cluster is very fragmented and it is surprisingly hard to allocate good locality for jobs. Despite it is possible to optimize individual job placements [4] [5], these policies could not resolve fragmentation problem fundamentally. This is because the heterogeneity of DLT workloads in terms of job sizes and durations. In a GPU cluster, there could be experimental jobs for model exploration using few GPUs, and also production tasks demanding much more GPUs. As smaller jobs finish at different moments, the released GPUs could be highly fragmented, making it even more difficult to find good locality for subsequent larger jobs.

We argue that a good DLT job scheduling policy should consider not only the placements of scheduled jobs, but also the selection of jobs to schedule, *i.e.,* the selection policy should be locality-aware. Because a reasonable selection policy could mitigate fragmentation and improve performance. The scheduler should pick appropriate jobs to schedule to match cluster fragmentation level and jobs' requirements on locality, adaptively. Also, it should delay the scheduling of certain jobs judiciously to restrict the fragmentation.

Scheduling is essentially a sequence decision problem, *i.e.,* each decision impacts subsequent decisions. And the problems of locality and fragmentation make DLT scheduling more complex compared to traditional scheduling problems. Specifically, each decision changes current cluster fragmentation condition, and considers job locality-sensitivity. The locality-sensitivity and changed cluster fragmentation make subsequent decisions more complex. And such complexity hinders heuristic-based solutions to achieve good scheduling efficiency.

Reinforcement Learning (RL) has shown great advantages over human-generated heuristics on sequence decision problems. In this paper, we make an attempt to leverage RL for smart locality-aware scheduling: we present SCHED^2, a DRL-based GPU cluster scheduler for DLT workloads. SCHED^2 captures various factors impacting GPU locality and fragmentation in its whole learning stack. First, we describe the occupancy and remaining time to release GPUs, to reflect the current and future degree of cluster fragmentation. Second, we profile a job's performance under various placements by executing it for a few batches, and use the profiling data to reflect job's locality-sensitivity. Third, we design a shared-weight neural network architecture which can effectively capture the locality information of resources. Through the awareness of job features and cluster status, SCHED^2 achieves great adaptability to changing cluster fragmentation and various performance sensitivity of jobs.

Our contributions are summarized as follows:

1) We identify the problems of GPU locality and fragmen-

978-1-7281-0962-6/19/$31.00 ©2019 IEEE

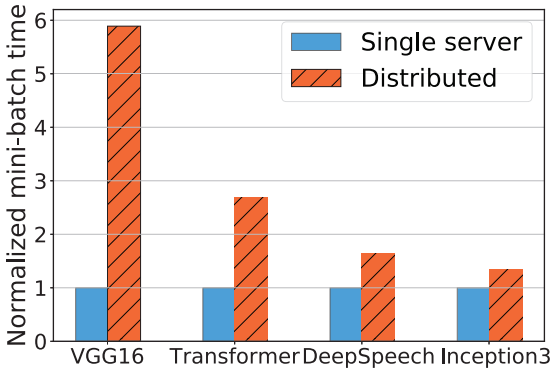


Fig. 1: DLT jobs' complex locality sensitivity.

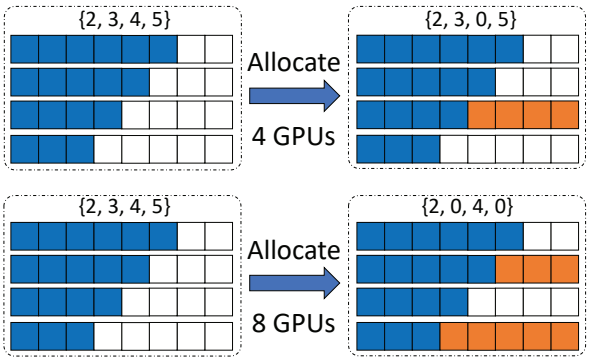


Fig. 2: An example of packing placement policy.

tation, and analyze the challenges in DLT scheduling.

2) We present SCHED^2, a DRL-based GPU cluster scheduler for DLT jobs. It adopts a locality-aware RL formulation and neural network to learn a policy that can improve performance and mitigate fragmentation. To the best of our knowledge, SCHED^2 is the first work that can capture and optimize GPU locality for DLT using DRL.
3) We evaluate SCHED^2 using realistic workloads, and the results demonstrate that SCHED^2 reduces average JCT (job completion time) by 4.6x and makespan (*i.e.,* the time to finish the whole set of jobs) by 2.1x, compared to traditional heuristic-based schedulers.
4) We summarize several patterns of the SCHED^2 policy, which might be guidance of future scheduler design.

II. MOTIVATION

In this section, we show the motivating characteristics and challenges of deep learning scheduling on GPUs form the observations and simulation with a real trace collected from a production cluster.

A. Performance Sensitivity to GPU Locality

The training performance of a DLT model is sensitive to the locality of allocated GPUs, *i.e.,* how locally in the system topology the GPUs are connected. Meanwhile, different models exhibit highly diverse sensitivity to GPU locality.

We measure mini-batch time under two types of placements of 4 popular models, and Figure 1 illustrates the result: 4 GPUs in the same server ("Single server"), and distributed in two servers ("Distributed"), respectively. There is significant performance variation between the two types of placements: the "VGG16" [6] model is 5.9x faster using a single server compared to distributed training. We also observe different degrees of the variation: 2.7x for Transformer [7], 1.6x for DeepSpeech [8], and 1.4x for Inception3 [9].

The root cause of the locality-sensitivity is the synchronization overhead between GPUs. This overhead is generally decided by the transfer-computation ratio. In a nutshell, in order to improve performance and resource utilization, it is critical to consider the locality-sensitivity of different jobs.

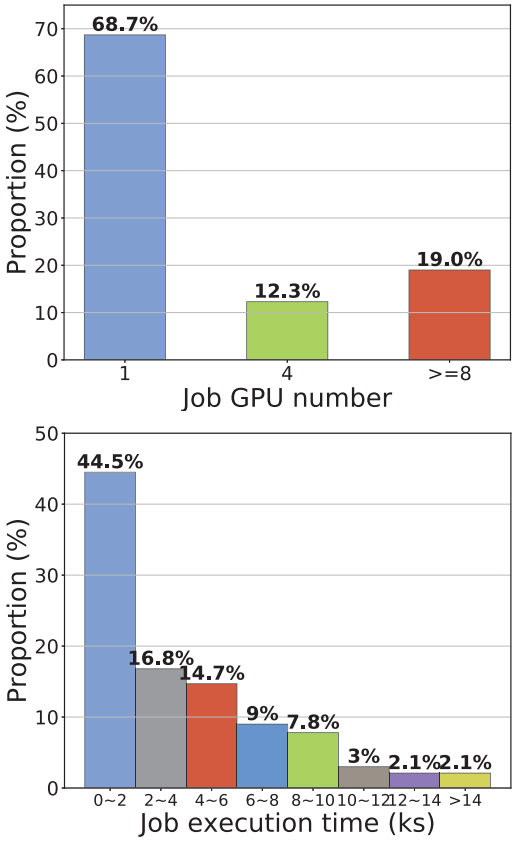


Fig. 3: Distribution of GPU demands and execution times of the production trace.

B. Fragmentation in GPU Clusters

To enforce the DLT locality, a scheduler usually use *packing* job placement policy, as it is friendly to DLT locality-sensitivity. The core idea is to allocate as few servers as possible (for better locality) with as few free GPUs as possible (for lower fragmentation). Specifically, the algorithm first tries to find the server with the fewest free GPUs that can satisfy the job's GPU demand if possible. If there is no such server, the algorithm chooses the server with the most amount of free GPUs, and then the remaining GPU requirement will be satisfied by the above logic again. Figure 2 illustrates an example of the algorithm.

However, we find that using packing strategy alone cannot solve the problem of GPU fragmentation problem: it is an

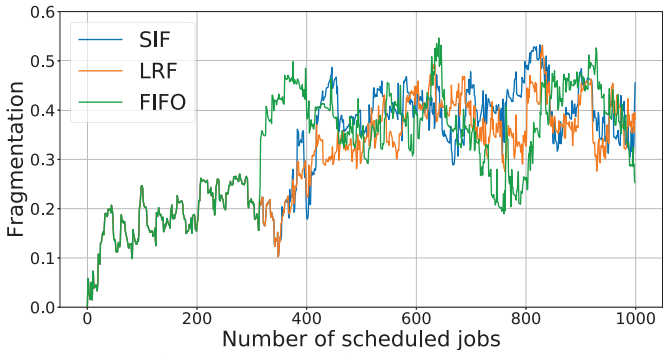


Fig. 4: Fragmentation reenactment.

intrinsic problem in GPU clusters due to various sizes and duration of jobs. On one hand, jobs have different demands on GPUs. We collect a real trace from a production GPU cluster in an anonymous company, a large proportion (68%) of jobs are small only using 1 GPU (Figure 3(a)). These small jobs make free GPUs scattered in different servers. On the other hand, jobs also have various execution time, ranging from several minutes to hours (Figure 3(b)). The two factors together make it hard to align the resource slots.

To demonstrate this, we reenact the scheduling progress of the real trace in a 15-server cluster by three common used scheduling policies and utilize packing placement policy. To measure and visualize fragmentation, we define a metric inspired by Jain's Index:

$$frag = 1 - (\sum_{i=1}^{n} x_i)^2 / (n \cdot \sum_{i=1}^{n} (x_i)^2) \qquad (1)$$

where x_i is the remaining time of GPU i, n is the GPU number in each server. This metric is higher when the remaining times of GPUs are more diverse. And we show the average fragmentation of severs in Figure 4. SIF schedules the shortest duration job; LRF schedules the fewest GPU job; FIFO schedules by order. As we see, the fragmentation is worse as scheduling progress no matter which policy we apply. This shows that only optimizing job placements (*i.e.,* packing) or locality-unaware scheduling policies (*e.g.,* FIFO) could not solve fragmentation problem.

To address the fragmentation problem, we should not only optimize jobs' locality, but also design a locality-aware scheduling policy, *i.e.,* it should schedule jobs whose locality-sensitivity match the current level of cluster fragmentation.

C. Challenges

The twin problems of locality and fragmentation introduce the following challenges to DLT scheduling:

Intricate impact of decisions. DLT scheduling is an sequence decision problem. Each scheduling decision changes not only the occupancy of GPUs, but also the fragmentation conditions. And the changed fragmentation impacts the performance of subsequent jobs. Moreover, the impact depends on the characteristics of specific jobs. So the scheduler should consider more factors than traditional job scheduling problems.

Complex locality-sensitivity. The locality-sensitivity brings performance difference, and its degree varies among different jobs (just like Figure 1). This is different from traditional job scheduling where only basic job features like duration or resource demand are considered. And how to express and utilize the sensitivity for scheduling is very challenging.

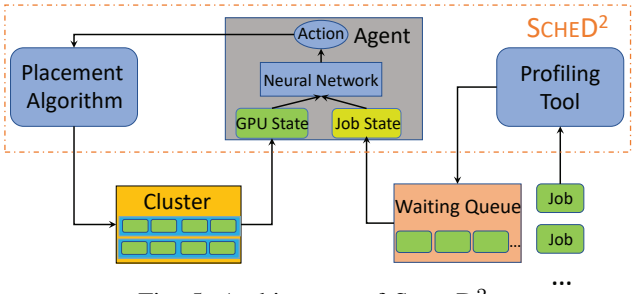


Fig. 5: Architecture of SCHED².

Larger state space. Considering locality and fragmentation, we should express not only the numerical aspect, but also the topological state of the resources. Topology information leads to a significantly larger state space than traditional scheduling problem where only the amount of resources matters.

These challenges motivate our attempts to adopt RL to mitigate fragmentation and improve performance.

III. SCHED² DESIGN

Reinforcement learning (RL) is a branch of machine learning, which is powerful hence broadly applied in sequential decision making problems. Its core task is to let an *agent* take *action*s in an *environment*, and improve its decision-making policy in a trial-and-error manner. Specifically, the agent will take an action based on its current *state*, and then observe the feedback (called *reward*) and next state from the environment. The agent will adjust its policy according to the reward, and gradually learn a better policy. We call a complete sequence an *episode*. The agent aims to maximize the *return*, *i.e.,* the accumulated rewards of an episode.

We apply DQN [10] framework to our DLT scheduling problem, as DQN has advantage on problem with discontinuous action space, compared with other RL frameworks.

A. Overview

We consider a GPU cluster comprising N servers, each equipped with M GPUs. We assume that all servers and GPUs have the same performance. Every job is submitted with the number of GPUs needed and mini-batches to execute. Every job occupies GPUs exclusively from start till completion.

As depicted in Figure 5, the SCHED² framework consists of three major components: (1) RL agent, (2) profiling tool, and (3) heuristic placement algorithm. Each incoming job is firstly profiled by the profiling tool. The RL agent feeds the job state and GPU state into a neural network. The neural network then outputs a selected job. The placement algorithm is responsible for assigning a scheduled job to a specific placement. And we use *packing* policy as the placement algorithm (Figure 2).

TABLE I: State definition.

	State Component	Num of Dimensions
GPU State	remaining time of GPUs	$N \times M$
Job State	profiling	k
	resource amount	1
	waiting time	1
		$\times J$
Statistics	remaining job num	1
	avg. resource amount	1
	avg. execution time	1
	avg. waiting time	1
		$NM + (k+2)J + 4$

B. States

We design the state definition of our RL problem to give an accurate characterization of the fragmentation level of the cluster and the locality-sensitivity of jobs. The state S is composed of two parts, GPU state s_{gpu} and job state s_{job}.

GPU state. The definition of GPU state aims to describe the GPU occupancy and fragmentation level of the cluster. Toward this end, we define GPU state as a matrix recording the remaining time of the job running on each GPU: $s_{gpu} = (t_{ij})_{N \times M}$, where t_{i*} is the remaining time of the job on each GPU of server i. The remaining time of a DLT job is easy to estimate as it typically exhibits stable performance and has a specified number of mini-batches. We could obtain its mini-batch time when it only finish a few mini-batches, and calculate the remaining time [4]. We use remaining time of the GPUs, because it enables RL to be aware of not only the current but also the future state of the cluster.

Job state. Job state describes the information of jobs in the waiting queue. Profile is the key to making RL aware of job's locality-sensitivity. Many works on DLT scheduling utilize profile to obtain job remaining time and other features [4] [5]. As profiling only cost several seconds, it is practical and almost costless. We profile each job under k different placements (from centrated to scattered) by executing it for a few mini-batches using the cluster profiling tool. In our experiment, we find that $k = 4$ is enough to express job locality-sensitivity. With the k measured performance numbers, we expect RL agent could estimate a job's performance under different degrees of locality accurately. For the other fields, *resource amount* is the number of GPUs of the job needs; *waiting time* is the total time that the job has been waiting in the queue. We consider the first J jobs among all waiting jobs based on their arrival time for scheduling for fairness among jobs, thus we only maintain the state for these jobs.

However, to make the RL agent aware of future job information, we add a sketch of the other waiting jobs as another field in the state called *statistics*. We concatenate the GPU state, job state and statistics as the full state (Table I).

C. Actions

We define actions as choosing jobs from the waiting queue (Figure 5), which does not cover placing the jobs to specific GPUs. We adopt this design for the following reasons. On one hand, the focus of SCHED^2 is to control fragmentation by smartly picking jobs. On the other hand, considering placements will make the action space explode. We use *packing* policy as the placement algorithm (Figure 2), but our framework easily allows other locality-enforcement placement policy to be used.

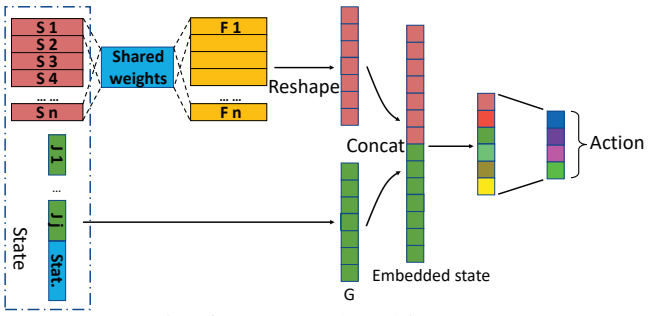


Fig. 6: Q-Network architecture.

At each time slot, the scheduler may schedule any combination of jobs from the first J jobs. This means that there would be 2^J possible scheduling decisions, which makes the action space still too large. Inspired by [11], to reduce action space, the agent selects only one job in each action, and executes multiple actions when an event occurs (event includes that a job is *submitted*, *scheduled* or *finished*). This reduces the action space to $J + 1$ ($\{1, 2, ..., J, \phi\}$ where action = i means schedule the i job in the waiting queue, and ϕ means do not schedule any jobs). In details, we froze the time, and the agent continuously selects jobs and changes state until it takes a ϕ action, then time 'goes' to next event and state changes. Its advantage is that the agent does not need to generate actions at each time slot, and it also reduces the action space.

D. Objective and Reward

We use *average execution effectiveness* as the primary system objective. Formally, for each job j, the execution effectiveness E_j is defined by

$$E_j = T_{ideal}/(T_{wait} + T_{actual}) \tag{2}$$

where T_{ideal} is its ideal execution time (it has been measured by profiling), and T_{wait} is its waiting time, and T_{actual} is its actual execution time under the assigned placement (this time is the same as remaining time in GPU state). This is a metric between 0 and 1. It is essentially a normalized JCT, and prevents the situation where the policy greedily schedule short jobs. RL agent takes the execution effectiveness of each scheduled job E_j as the reward signal of each action.

E. Network Architecture

We propose a new shared-weight network architecture to better extract locality information from the states. As shown in Figure 6, S_i represents the GPU state of server i; J_i represents the job state of job i; $Stat$ is the *statistics*. We define $f(x; w)$ as a forward network with weights w. Then the network architecture could formulated as:

$$F_i = f(S_i; w_{shared}), G = f(J_1, ..., J_j, Stat; w1) \tag{3}$$

$$E = Concat(F_1, ..., F_n, G) \tag{4}$$

$$A = f(f(E; w2); w3) \quad (5)$$

where w_{shared} is our shared weights, E is the *embedding state* and A is the *action*.

The key insight of this design is that each server is the boundary of a set of servers, and each job is the unit for scheduling. Therefore, we can train a set of parameters to process each set of the GPUs and each job. This design has the following advantages. First, it has fewer weights hence trains and converges faster than fully connected networks. Second, it can better preserve locality relationship among servers as the shared weights are applied inside server boundary.

F. Training Algorithm

We use episodic training method, and each episode is a job trace selected from the trace set. We use the DQN [10] algorithm to train our Q-network. We adopt ϵ-greedy policy and prioritized replay [12] techniques for the exploration.

IV. Evaluation

A. Methodology

Workload We consider a GPU cluster composed of 15 servers, each with 8 GPUs with identical performance (120 GPUs in total). We collect a real trace from a production GPU cluster for DLT in a large technology company. The trace includes 6,261 DLT jobs from three categories: computer vision (10%), Natural Language Processing (60%), and Speech (30%), just as reported in [13]. The mean interval of arrival times is 172s. The job distribution is shown in Figure 3. Due to privacy reasons, we cannot access the code of the jobs. In their place, we pick 6 state-of-the-art deep learning models from GitHub (Table II) to replace according to the categories. For each job, we assign it a mini-batch number calculated according to its execution time.

TABLE II: Deep learning models used in the trace.

Model	Dataset
Alexnet [1]	ImageNet [14]
InceptionV3 [9]	ImageNet
Resnet-50 [15]	ImageNet
VGG16 [6]	ImageNet
DeepSpeech [8]	Common Voice [16]
Transformer [7]	WMT16 [17]

Because DRL training requires massive training data, we generate a data set by *sampling* jobs from the original trace. The intervals of arrival times of the sampled jobs also follow the distribution of the original trace, to make the synthetic data as realistic as possible. We call this data set *realistic trace*.

Profile-driven simulator Due to the long-running nature of DLT jobs and the scale of our trace, it is extremely time- and resource-consuming to train our RL agent in a real cluster. Therefore, we implement a profile-driven simulator to simulate the execution of a given scheduling. For each job, we profiled its performance under the assigned locality to calculate the JCT, and we will save the profiling results for reuse.

RL Details Our training data contains 30000 DLT jobs. We divide it into 30 traces (episodes) in order (1000 jobs per trace). And we use another 30 traces as the test set. All the presented results are from the test set. And RL selects jobs from the first 10 jobs in waiting queue. It takes 15 hours and 2400 episodes to train the Q-network in a Nvidia 1080Ti GPU.

Baselines We compare SCHED^2 with following baselines, as they are widely used by most clusters. All of these baselines use the *packing* placement algorithm.

* ***SIF (Shortest Ideal time job First)***, schedules the job with the shortest ideal execution time from waiting queue.
* ***DSIF (Delayed Shortest Ideal time job First)***, also schedules the job with the shortest ideal execution time, but if the selected job could not get the best locality, it will delay the job for at most 3 times.
* ***SAF (Shortest Actual time job First)***, schedules the job with the shortest actual execution time given the current cluster state. We assume there is an oracle that can tell the scheduler what placement each job will be assigned and the corresponding mini-batch time.
* ***LRF (Least Resource job First)***, schedules the job demanding the fewest GPUs.
* ***SPF (Smallest Product job First)***, schedules the job with the smallest product of GPU demand and the ideal execution time (GPU * ideal time).
* ***FIFO***, schedules jobs in order.

B. Performance Comparison of Schedulers

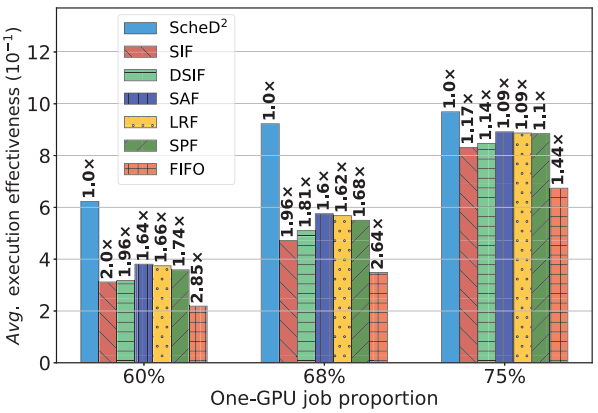
Performance of realistic traces The results are shown in the middle bars (68%) of Figure 7. SCHED^2 achieves significant improvement over all the baseline approaches on each performance metric: it outperforms the best among the baselines on execution effectivenss by 1.6x, makespan by 2.1x and JCT by 4.6x, respectively.

Performance under different GPU demand distributions To observe the impact of the GPU demands of jobs, we generate two new data sets containing 60% 1-GPU jobs and 75% 1-GPU jobs (68% for the realistic trace). Figure 7 shows the performance results of the three traces. SCHED^2 achieves higher performance gain with lower 1-GPU proportion in most cases. For example, in terms of execution effectiveness, SCHED^2 outperforms the best baseline by 1.64x with the proportion of 60%, 1.6x for 68% and only 1.09x for 75%.

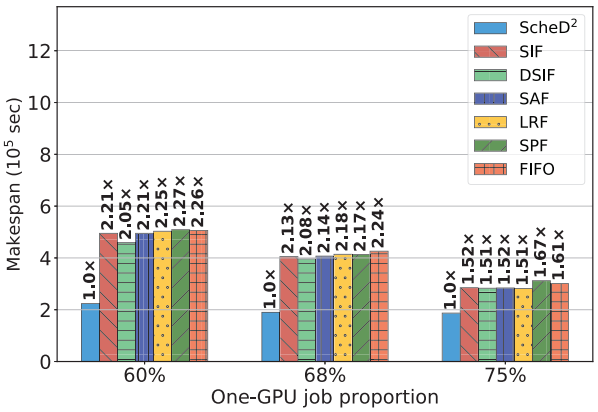
Performance under different intervals We generate traces with different intervals by scaling the arrival time intervals of the original trace. Figure 8 shows the job execution effectiveness of different schedulers at different intervals. SCHED^2 achieves the best performance at every intervals.

C. Understanding SCHED^2 Behavior

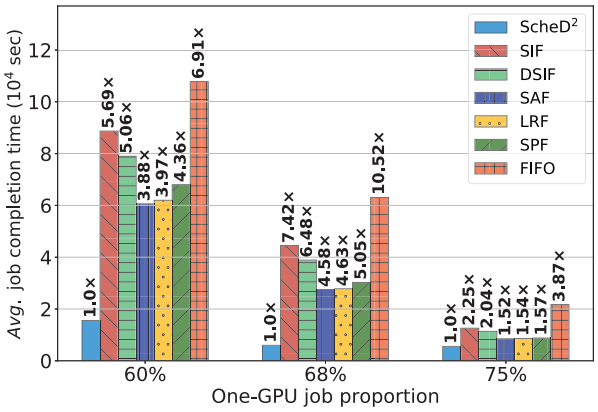
We analysis the behaviors of SCHED^2, in order to figure out the insight of its performance gain. The results in this section are all from the experiment using the realistic trace in §IV-B.



(a) *Avg.* execution effectiveness.



(b) Makespan



(c) *Avg.* JCT

Fig. 7: Comparison of performance metrics.

SCHED^2 **effectively mitigates cluster fragmentation.** Figure 9 shows the average fragmentation of all servers over time. The fragmentation of the baselines gets worse over time; the fragmentation of SCHED^2 is always lower, and periodically decreases. We also illustrate the total required GPU amount of jobs in waiting queue over time in Figure 9 (solid blue line). We compare it with the SCHED^2 fragmentation curve, and find that the changes of the two curves are synchronous sometimes (*e.g.,* from 800 to 1000). This proves that SCHED^2 actively defragmentation by holding resources sometimes.

Lower fragmentation can translate to better locality, hence

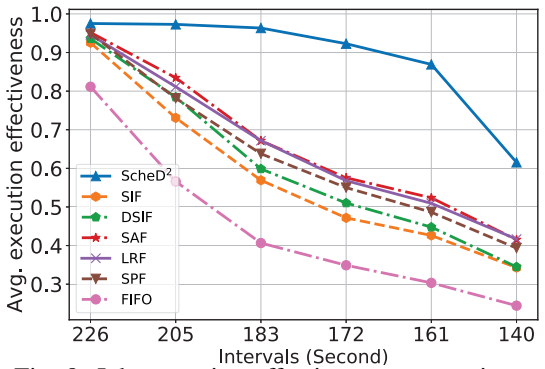


Fig. 8: Job execution effectiveness comparison.

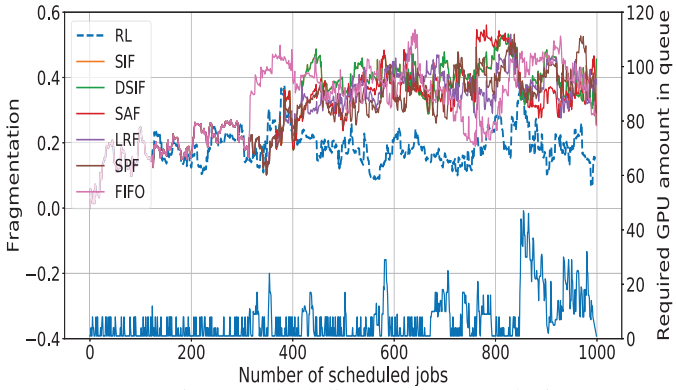


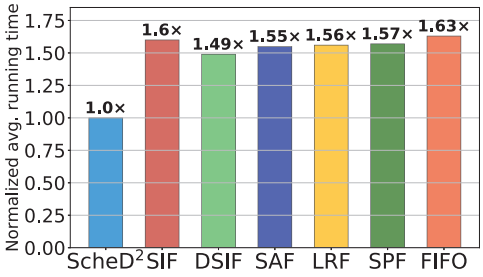
Fig. 9: Scheduling progress analysis.

shorter running time and higher resource utilization. As shown in Figure 10(a), the average execution time of SCHED^2 is much shorter (5422s vs 8427s). Moreover, SCHED^2 also significantly reduces the waiting time (Figure 10(b), 34x improvement, 557s vs 19075s), even if SCHED^2 uses resource holding for defragmentation. And the over 19 thousand second waiting time means that the baseline method is not available under the load level. In another word, SCHED^2 could utilize a smaller cluster to serve a higher load level.

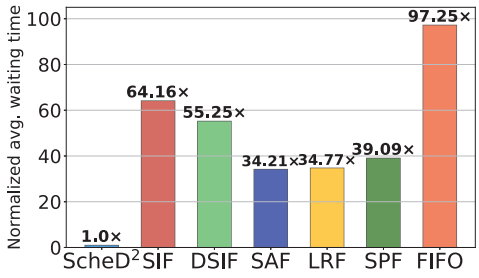
SCHED^2 **smartly adjusts its scheduling behavior according to various locality-sensitivity of individual jobs.** We divide all the jobs in our trace into two categories: "sensitive" and "insensitive" by their performance variance under different levels of locality (*i.e.,* profiling data in Table I). As shown in Table III, only 15.4% of the sensitive jobs are not assigned the best locality. In comparison, this proportion of the insensitive jobs is 32.5%. Besides, the average waiting times of the two categories of jobs are 1466s and 741s, respectively.

These results demonstrate that SCHED^2 is aware of the locality-sensitivity of different jobs, and will make decisions accordingly: it tends to make the sensitive jobs wait longer, for better locality and defragmentation; while it will schedule the insensitive jobs with shorter delay, as bad locality will not introduce much performance loss.

SCHED^2 **reuses good locality adaptively.** SCHED^2 is capable of protecting a good locality, so that the locality could be reused for the coming multi-GPU jobs. This makes multi-GPU jobs achieve good localities more easily. For example,



(a) $Avg.$ execution time



(b) $Avg.$ waiting time

Fig. 10: $Avg.$ running time and waiting time comparison.

TABLE III: Comparison between different jobs ("relaxed locality" means a job is not assigned the best locality).

Job type	Sensitive	Insensitive
Proportion of relaxed locality	15.4%	32.5%
Avg. waiting time (second)	1466	741

suppose a multi-GPU job gets a good resource locality, when this job finishes, SCHED^2 tends to hold the released GPUs to serve subsequent multi-GPUs jobs, so that the locality will not be polluted by smaller jobs. Our statistics show that over **50%** of the multi-GPU jobs are assigned reused locality while the number of the baselines is only about **20%**. Moreover, whether to protect the good locality is adaptive to the coming jobs.

V. Related Work

DLT workload scheduling has become a new research topic and is drawing more attention in recent years. Optimus [4] is a DLT scheduler that uses online fitting of model convergence and job performance to derive a proper number of parameter-servers and workers for each DLT job. Amaral *et al.* propose a graph-mapping algorithm to match DLT jobs to good GPU locality in multi-GPU systems [5]. Gandiva [13] proposes to leverage profiling and several other scheduling primitives to dynamically improve GPU locality of jobs. DeepRM [11] should be the first work to apply DRL on job scheduling. However, DeepRM does not consider fragmentation and locality-sensitivity, and it focuses on a small toy trace set which obeys the simple Poisson distribution. However, we find the realistic trace does not obey the feature.

VI. Conclusion

We present SCHED^2, a locality-aware scheduling framework for DLT workloads in GPU clusters using DRL. The main design goal of SCHED^2 is to control fragmentation in GPU clusters and improve cluster utilization and job performance.We show via evaluation using realistic workloads that SCHED^2 can effectively control cluster fragmentation and job locality, and reduces JCT by 4.6x and makespan by 2.1x.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in neural information processing systems*, 2013, pp. 2643–2651.

[3] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv:1409.0473*, 2014.

[4] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 3.

[5] M. Amaral, J. Polo, D. Carrera, S. Seelam, and M. Steinder, "Topology-aware gpu scheduling for learning workloads in cloud environments," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 17.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[8] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv:1412.5567*, 2014.

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[11] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 50–56.

[12] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv:1511.05952*, 2015.

[13] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation*, 2018, pp. 595–610.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2009, pp. 248–255.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[16] "Common voice dataset." [Online]. Available: https://voice.mozilla.org/

[17] "Wmt16 dataset." [Online]. Available: http://www.statmt.org/wmt16/