Low-latency Job Scheduling with Preemption for the Development of Deep Learning

Hidehito Yabuuchi, *The University of Tokyo;* Daisuke Taniwaki and Shingo Omura, *Preferred Networks, Inc.*

https://www.usenix.org/conference/opml19/presentation/yabuuchi

This paper is included in the Proceedings of the 2019 USENIX Conference on Operational Machine Learning (OpML '19).

May 20, 2019 • Santa Clara, CA, USA

ISBN 978-1-939133-00-7

Open access to the Proceedings of the 2019 USENIX Conference on Operational Machine Learning is sponsored by USENIX.

Low-latency Job Scheduling with Preemption for the Development of Deep Learning

Hidehito Yabuuchi *
The University of Tokyo
yabuuchi@os.ecc.u-tokyo.ac.jp

Daisuke Taniwaki Shingo Omura
Preferred Networks, Inc.
{dtaniwaki, omura}@preferred.jp

Abstract

Efficient job scheduling of *trial-and-error (TE)* jobs is a challenging problem in deep learning projects. Unfortunately, existing job schedulers to date do not feature well-balanced scheduling for the mixture of TE and *best-effort (BE)* jobs, or they can handle the mixture in limited situations at most. To fill in this niche, we present an algorithm that efficiently schedules both TE and BE jobs by selectively preempting the BE jobs that can be, when the time comes, resumed without much delay. In our simulation study with synthetic workloads, we were able to reduce the 95th percentile of the *slowdown rates* for the TE jobs in the standard FIFO strategy by 96.6% while compromising the median of the BE slowdown rates by only 18.0% and the 95th percentile by only 23.9%.

1 Introduction

Efficient job scheduling of clusters is in high demand these days, especially due to the recent explosive development of deep learning (DL) algorithms. One important type of jobs in the development of DL is *trial-and-error (TE)* jobs, in which the users conduct small-scale experiments on a trial basis for the debugging and the testing of prototype algorithms. In fact, for the private cluster at the authors' institution, TE jobs account for approximately 30% of all jobs in six months. Starting the TE jobs with low latency is critical because the users often want to monitor the learning curves of the prototypes immediately in order to save time for exploring numerous other options. The other jobs can be executed in the *best-effort (BE)* manner, but their delay should be minimized.

Unfortunately, most scheduling algorithms to date can handle the mixture of TE and BE jobs in certain situations at most. Big-C [2], a container-based preemptive job scheduler, does not handle multiplexing of GPUs. Optimus [5] and Gandiva [6] are efficient job schedulers for DL jobs, but they are only compatible with select DL frameworks. Reservation-based schedulers such as Hawk [4] reserve a separate portion of a cluster to guarantee the immediate scheduling for short jobs. Given highly diverse workload, however, it is often challenging to find the optimal reservation factor.

In this paper, we take the novel strategy of systematically suspending a selected set of BE jobs in favor of the TE jobs. Our proposed algorithm can handle any DL jobs that can be suspended, and it can be used in a variety of situations. We also take special care not to *neglect* the BE jobs. By selectively preempting the BE jobs for which the scheduler can re-schedule its execution in relatively short time, our algorithm makes sure not to greatly delay the BE jobs.

2 Proposed Preemption Algorithm

2.1 System Model

We built our preemption algorithm on the FIFO principle, which is widely used in production (e.g., Kubernetes [1]), so that we can easily integrate our algorithm into the existing frameworks. For simplicity, we assume that each job consists of a single task. Unlike big-data processing, a typical job that trains a DL model does not have multiple tasks.

When submitting a job, the users are asked to specify its type, either TE or BE, along with the types and the amount of the resource demanded for the job. When a TE job arrives at a job queue, one or more BE jobs are suspended to make room for the incoming TE job if the resource is insufficient. The preempted BE jobs are placed back on the top of the queue to observe the FIFO. Some jobs demand the time for suspension processing (e.g., storing data) before being suspended. We therefore allow a *grace period (GP)* of user-specified length for each suspension prompt. In this study, we propose an efficient rule for deciding which BE jobs shall be preempted.

2.2 Proposed Algorithm

Our algorithm is based on the following observations:

Minimizing the re-scheduling intervals. Since a preempted BE job is placed back on the top of the queue, it will be re-scheduled without much delay. However, if a BE

*Work done during an internship at Preferred Networks, Inc.

job that demands large resource is preempted without any consideration, other BE jobs waiting in the queue must wait until the scheduler secures a large room for the resumption of the preempted large BE job.

Minimizing the number of preemptions. On the other hand, preempting too small a BE job can also increase the overall slowdown of BE jobs. If a single preemption cannot make enough room for an incoming TE job, the scheduler has to preempt still another BE job. Many numbers of preemptions increase the total time loss incurred by the re-scheduling.

Minimizing the preemption-incurred time loss. It is also not preferable to preempt a BE job with too long a GP, because the length of GP affects the time until the execution of the incoming TE jobs.

Thus, we shall always preferentially preempt BE jobs with (1) small resource demand, (2) an ability to offer enough resource for the incoming TE job, and (3) short GPs. Our *Fitting Grace Period Preemption (FitGpp)* algorithm evaluates the following score for each BE job j:

$$\text{Score}(j) := \frac{\|D_j\|}{\max_{j\in\mathcal{J}} \|D_j\|} + s \times \frac{\text{GP}_j}{\max_{j\in\mathcal{J}} \text{GP}_j} \qquad (1)$$

where D_j is the vector of resource quantities demanded by the job j [1], and $\mathcal{J}$ is the set of all running BE jobs. The parameter s determines the importance of the GP relative to the resource demand. At all time, FitGpp preempts the BE job that solves:

$$\arg\min \left\{ \text{Score}(j) \mid D_{\text{TE}} \leq D_j + N \ \wedge \ \text{PC}_j < P \right\} \qquad (2)$$

where N is the amount of free resource of the node on which j is running, PC_j is the number of times that j has been preempted, and P is the maximum number of times a given BE job can be preempted, which guards the job against starvation.

Note that the FitGpp's criterion of preemption does not depend on the execution time of jobs, so that it is not affected by the algorithm's ability to estimate the execution time. This is an important advantage of FitGpp because the estimation is generally hard [3]. This is especially true for DL jobs, whose execution time are sensitive to the hyper-parameters.

3 Evaluation

Here we briefly describe our simulation study. The more comprehensive evaluation can be found in our report [7].

We evaluated our FitGpp algorithm in a simulated environment, which consisted of 84 nodes, each having 32 CPUs, 256 GB RAM, and 8 GPUs. We compared FitGpp against (non-preemptive) vanilla FIFO, *Longest Remaining Time Preemption (LRTP)*, and *RAND*. LRTP is the algorithm used in Big-C [2], and it preferentially preempts the job with the longest remaining execution time. RAND is an algorithm that preempts a randomly selected running BE job. We compared the performance of the algorithms based on the *slowdown rate* computed by the formula $1 + \frac{\text{WaitingTime}}{\text{ExecutionTime}}$.

In order to synthesize a realistic set of workloads, we analyzed a trace of the cluster at the authors' institution, which consists of over 50,000 jobs. We approximated the mixture of TE and BE jobs in the trace with a mixture of truncated normal distributions. For the lengths of GPs, we used a normal distribution with the mean of 3 min. We set the maximum preemption limit P to 1. We evaluated the algorithms on a set of 2^{19} jobs generated from the distributions with 30% of them being TE. In the simulation, the jobs were submitted at such a rate that the cluster load would be kept at 2.0 if they were scheduled by FIFO. Additional details are in Appendix A.

The results are given in Fig. 1. FitGpp with $s = 4.0$ was able to reduce the 95th percentile of the slowdown rates of the TE jobs by 96.6% relative to that of FIFO. Our algorithm increased the median of the slowdown rates of BE jobs by only 18.0% and the 95th percentile by only 23.9%.

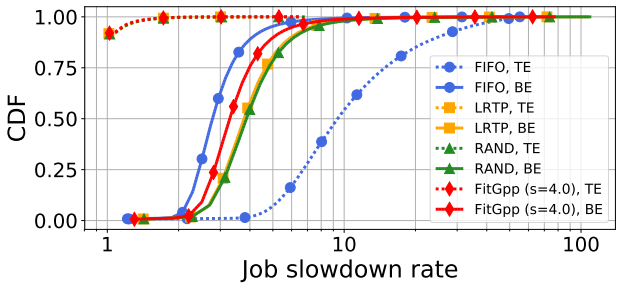

Figure 1: Job slowdown rates with synthetic workloads.

The superiority of FitGpp in this experiment was most likely due to its ability to shorten the intervals between preemptions and re-scheduling. In fact, the median of the intervals with FitGpp was almost half compared to that of LRTP and RAND, and the 95th percentile was 20% shorter than that of LRTP and 33% shorter than that of RAND. We shall also not forget that FitGpp makes an effort to reduce the total number of preemptions. When $P = 1$, it reduced the total number of preempted jobs to less than 7.0% relative to that of LRTP and RAND.

4 Conclusion

In this paper, we presented FitGpp, a preemption algorithm that reduces the latency of the TE jobs while controlling the slowdown of the BE jobs incurred by the preemption processes. Future directions include extending of this work to non-FIFO based setting and scheduling of multi-node jobs in distributed DL. Finally, the application of our algorithm is not necessarily limited to the scheduling of DL jobs. We shall be able to extend our algorithm to any type of workload that consists of a mixture of TE-like jobs and BE-like jobs.

[1] Each coordinate entry of D_j is the amount of a type of resource (e.g., CPU and RAM) relative to the capacity of the node.

Acknowledgments

We thank K. Uenishi, K. Fukuda, S. Maeda, and Y. Doi for fruitful discussion and reviewing this paper. We also thank M. Koyama for a help in the composition of the paper.

References

[1] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *ACM Queue*, 14:70–93, 2016.

[2] Wei Chen, Jia Rao, and Xiaobo Zhou. Preemptive, Low Latency Datacenter Scheduling via Lightweight Virtualization. In *Proceedings of 2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 251–263, 2017.

[3] Pamela Delgado, Diego Didona, Florin Dinu, and Willy Zwaenepoel. Kairos: Preemptive Data Center Scheduling Without Runtime Estimates. In *Proceedings of ACM Symposium of Cloud Computing conference (SoCC)*, pages 135–148, 2018.

[4] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. Hawk: Hybrid Datacenter Scheduling. In *Proceedings of 2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 499–510, 2015.

[5] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *Proceedings of Thirteenth EuroSys Conference (EuroSys '18)*, 2018.

[6] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 515–610, 2018.

[7] Hidehito Yabuuchi, Daisuke Taniwaki, and Singo Omura. Low-latency job scheduling with preemption for the development of deep learning. *ArXiv e-prints*, arXiv:1902.01613v1 [cs.DC], 2019.

Appendix A Experimental Details

We simulated LongestRemainingTimePreemption (LRTP) algorithm on the assumption that it can perfectly estimate the execution time of each job. Both LRTP and RAND continue the preemption process until they can prepare enough resource for an incoming TE job. For the evaluation of RAND, we repeated the same experiment four times and report the average statistics.

In order to synthesize realistic workloads, we analyzed a trace of the cluster at the authors' institution. The trace consisted of approximately 50,000 jobs with about 30% of them being TE. Fig. 2 shows the brief statistics of the trace.

To create a realistic sequence of synthetic workloads, we approximated the empirical distributions of (1) the execution time, (2) the number of demanded CPUs, (3) the amount of demanded RAM, and (4) the number of demanded GPUs for both the TE jobs and the BE jobs with separate normal distributions, and artificially generated typical jobs from their truncated versions. The means of the fitted normal distributions for the execution time of the TE jobs and the BE jobs were respectively 5 min and 30 min. We truncated these distributions at 30 min and 24 hours, in this order.

For the lengths of GPs, we prepared the normal distribution with the mean of 3 min and truncated the distribution at 20 min. We set the length of GPs at such large values for the following three reasons: (1) typical DL jobs tend to accompany large data to store before the suspension, (2) the data often requires preprocessing step for the storage, such as serialization, and (3) we expect the developers of DL algorithms to specify long GPs because a prematurely suspended job is destined to fail.

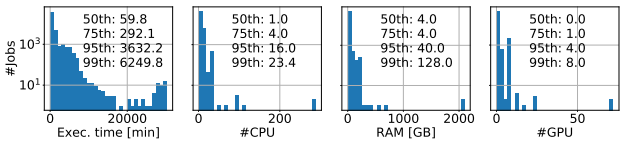

Figure 2: Statistics of jobs on the cluster at the authors' institution.