Toward Interference-aware GPU Container Co-scheduling Learning from Application Profiles

Sejin Kim
Department of Computer Science
Sookmyung Women's University
Seoul, Korea
wonder960702@gmail.com

Yoonhee Kim
Department of Computer Science
Sookmyung Women's University
Seoul, Korea
yulan@sookmyung.ac.kr

***Abstract*—Issues related to operating Graphic Processing Unit (GPU) applications efficiently and improving overall system throughput in a GPU cluster environment exist. The platform may not utilize resource of a GPU fully depending on application characteristics because a conventional cluster orchestration platform using GPUs only supports a single execution of an application on a GPU. However, co-execution of GPU applications causes interference coming from resource contention among the applications. If various resource usage of GPU applications is not reflected, it could lead to an unbalanced usage of computing resources and consequently reduce performance in a GPU cluster. This study proposes interference-aware architecture, Co-scheML and evaluates case studies with it for workload execution of GPU applications such as High Performance Computing (HPC), Deep Learning (DL) Training, and DL Inference. Diverse resource usage is profiled to identify various degree of their interference of applications. Due to the difficulty of predicting the interference using these characteristics, interference model is generated by applying a Machine Learning (ML) model with defined GPU metrics. Proposed architecture predicts interference and deploys an application which is co-executed with a running application. Experimental results of case studies with Co-ScheML show that average job completion time is improved by 22%, and the makespan is shortened by 32% in average, as compared to baseline schedulers.**

Keywords- GPU applications, interference, co-execution, Co-scheML scheduler, resource contention, GPU utilization

I. INTRODUCTION

New and emerging general-purpose graphics processing architectures, programming environment, and platforms have various issues related to their optimization, application performance, and system throughput as graphical processing unit(GPU) becomes popular in general. Traditional cluster schedulers of cluster orchestration platforms such as Yarn [1] and Kubernetes [2] might not fully utilize GPU computing resources when they execute an application alone on a GPU at a time. To overcome this limited use of GPU resource, it is possible to co-execute multiple and diverse applications, which have miscellaneous patterns of resource usage.

To achieve General Purpose GPU (GPGPU) sharing, NVIDIA has proposed multiple process service (MPS) to execute multiple kernel processes concurrently [3]. However, performance of this technique may degrade owing to interference from co-executed kernels accessing the same device at a time. Studies on GPU sharing include co-deployment of applications with strategies coming from monitoring information, user requirements [9,10,11] or weights using GPU usage profiles [12]. However, these studies do not consider interference during co-execution of applications, which leads to performance degradation. Moreover, avoiding interference using resource usage records of diverse applications is challenging.

Recent studies have addressed interference issues caused by resource contention that occurs from its sharing [8,13,14]. A previous study defines some performance metrics of contended resources (SM, DRAM, cache, Interconnect) and applies accumulated profiling of metrics for interference avoidance scheduling based on the similarity of metrics[14]. However, it did not consider the fatalness of interference is diverse depending on resource characteristics. In addition, an execution failure such as out-of-memory (OOM) has not been fully discussed. Studies applying machine learning(ML) to scheduling to predict interference using resource metrics [14]. [13] defines detailed resources of GPU (GPU, GPU memory, PCIe) for deep learning(DL) applications. [8] defines diverse factors that affect interference according to cluster and node levels, and then provides scheduling with various machine learning (ML) model to each level depending on the factors. However, those studies have experiments on only deep learning (DL) workloads, which have uniformed resource usage patterns.

This paper introduces Co-scheML to improve performance and optimize resource usage for various applications running on GPUs. To solve interference problem that occurs during GPU sharing, it defines appropriate metrics for applications with various characteristics and proposes a scheduling method to avoid interference with ML on GPU.

Contributions of this study are as follows.

- The interference model with ML predicts the degree of interference from accumulated profiling.
- An architecture co-executes applications according to predicted interference value is proposed.

In experiments, interference-aware scheduler's performance is compared to Binpack, Loadbalance, and Mystics[14] methods. An evaluation includes job completion time (JCT), makespan depending on a variety of state-of-the-art scheduler.

978-1-7281-8414-2/20/$31.00 ©2020 IEEE
DOI 10.1109/ACSOS-C51401.2020.00023

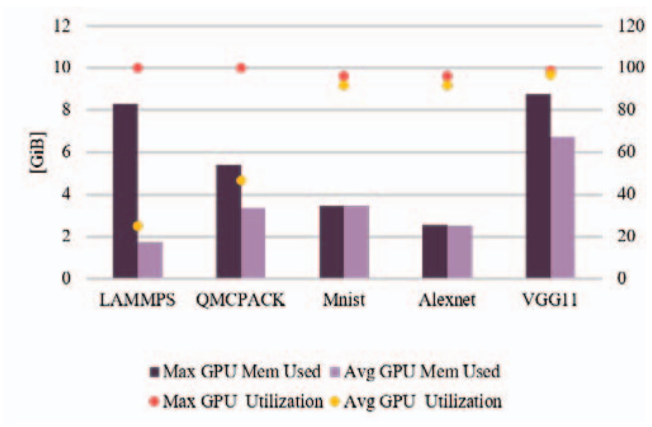

Figure 1. Resource over-commitment of GPU applications

The rest of this paper is organized as follows. Section 2 introduces motivation. Interference modeling is described in Section 3. A proposed architecture with interference aware scheduler is explained in Section 4 and its experiments are described in Section 5. The related studies are provided in Section 6 and concluding remarks are given in Section 7.

II. Motivation

Fig. 1 shows GPU resource utilization patterns of applications over NVIDIA TITAN XP GPU and i7-5820K CPU. Two HPC applications (LAMMPS[32], QMCPACK[33]) and three CNN models (mnist, alexnet, vgg11) are executed with standard input sets from NVIDIA GPU Cloud (NGC) [6] and Tensorflow CNN benchmark [15]. GPU memory is over-committed by approximately 54% in average. GPU utilization gap is by approximately 51% from average to maximum. The result drives to overstated resource requirement for a job scheduler as having relatively large deviation from mean values for the HPC applications compared to one for the deep learning (DL) applications. If a scheduler sticks to allocate average number of resources for HPC applications, an OOM failure and performance degradation may occur by overlapping peak times of resources usage at which a GPU application uses most resources. This experimental result indicates that it is necessary to prevent a resource over-commitment by utilizing profiling and predicting interference for avoiding from OOM failure or severe performance degradation.

III. Interference Modeling

We define interference value as co-executed time of an application normalized by solo-run time. If interference values are obtained directly after executing all application pairs, the most accurate and optimal results can be achieved. However, it is not realistic to calculate interference values with all pairs with large number of applications with long execution time. For N applications, interference values of (N*(N-1))/2 pairs are needed. Interference is predicted through offline profiling information without executing all application pairs for this reason. Profiling information includes hardware characteristics, which affect the co-execution of applications on a GPU, and metrics deriving from the prior observation.

TABLE Ⅰ. Resource Metrics for Interference Modeling

Metric	
GPU utilization average	GPU memory utilization average
SM efficiency average	Device to host throughput
Warp efficiency average	Host to Device throughput
IPC	GLD throughput
Occupancy average	GST throughput
GPU memory used max	Execution time
GPU memory used average	

A. *Metrics used for profiling resource usage*

We define resource metrics that influence performance at the time of actual co-execution of applications referred to the metrics in [14]. Each metric was collected using the NVIDIA profiler tool, nvprof, for predicting interference prevention during the co-execution. Table 1 shows detailed information regarding the metrics obtained during the profiling of each related resource.

GPU utilization average is the average execution time of one or more kernels on the GPU. SM efficiency average is the average time at which one or more warps are active in a particular multiprocessor on the GPU in percentage. Warp efficiency average is the average number of active threads for each warp in the SM. IPC is the number of commands executed per cycle. Occupancy average is the average number of active wraps per active cycle supported to the SM. GPU memory used max is the maximum amount of GPU memory used during the application program execution. GPU memory used average is the average GPU memory used during the application program execution. GPU memory utilization average is the average time for reading or writing GPU memory over a specific period of time during the application execution in percentage. Device to host throughput is the data throughput moving from global memory to CPU memory. Host to device throughput is the data throughput from CPU memory to GPU memory. The Cache: GLD (Global memory Load) throughput metric

includes transactions served by the L1 and L2 caches. This metric is the amount of cache hit when loading into global memory. The GST (global memory store) throughput is also related to the L1 and L2 cache, but indicates a cache hit when storing in global memory. The execution time of each application and input during an application execution are recorded because the profiling information can vary depending on the input and parameters used even with the

same applications. If an application with the same configuration is submitted, previously collected profiling information can be used.

B. *Model construction*

For interference modeling, we established three machine learning models- linear regression, random forest regression referred to [13], and decision tree regression[34]. We used total of 12 applications, described in Section 3. To establish the models, a total of 144 datasets were used as their pairs were modeled. We used a combination of metrics for each application as an input of the model. Metric values were normalized because the units and scales used for each metric differ. The model output is interference value, which is represented as a ratio between the time of co-execution and the time when the application is executed alone. We executed all application pairs and obtained the execution time to calculate the interference for constructing the model. The interference value applied the average value from three experimental results for accurate measurements. In addition, 5-fold validation was used to improve the model accuracy and reliability of the performance evaluation. The entire dataset is divided into five subsets, four of which are designated as the training data, with the remainder designated as the validation data. This process is repeated five times.

Table 2 shows the mean squared error (MSE) values and R2 scores for three types of machine learning models. The MSE represents the difference between the predicted and actual values, and the closer it is to zero the higher the accuracy. R-Square is a validation measure of a regression model, the explanatory power of which is higher as it reaches closer to 1. The random forest regression model showed the best performance in a container environment. The random forest model was used in this experiment for this reason.

TABLE Ⅱ. MEAN SQUARED ERROR AND R-SQUARE OF REGRESSION MODELS

	Linear Regression	**Decision Tree Regression**	**Random Forest Regression**
MSE	0.0546	0.0269	0.0222
R-Square	0.6946	0.8500	0.8758

IV. IMPLEMENTATION DETAILS

Kubernetes manages containers with modified device-plugin for sharing of GPUs. If an application is first submitted, it is executed alone and profiled to collect metrics. Its profiling information is labeled as application name and input data and stored in the Profile repository. Metrics are stored in a time-series-based database, influxDB[5] and a profiling step is carried out offline. Co-scheML requests interference value from the Profile repository and Model and selects pairs with minimum interference value. The decision of Co-scheML is sent to each worker node's Kubelet. Kubelet launches applications using the modified GPU Device Plugin for sharing resources. While the application is being executed, the progress of the application is continuously monitored and the profiling information is updated to improve the accuracy in Monitor.

V. EVALUATION

A. Evaluation methodology

1) Experiment environment: The Kubernetes-based private GPU cluster is used for the evaluation. The cluster is comprised of one master node and three computing nodes: the GPU node, the work node as shown in Table 3.

TABLE Ⅲ. EXPERIMENTAL SETTING

	Node Details	
	CPU(master)	***GPU***
Architecture	Intel® Core™ i7-5820K	Nvidia GeForce Titan Xp D5x
Core Clock	3.30GHz	1.58GHz
Num of Cores	6	3840
Mem. Size	32GB	12GB
Threading API	-	Nvidia CUDA 10.0
OS	Ubuntu 16.04.6 LTS	Ubuntu 16.04.6 LTS

2) Workload: The workload is consisted of twelve real-world applications. Four HPC applications (LAMMPS, GROMACS, QMCPACK, HOOMD) from NVIDIA GPU Cloud (NGC) [6], five DL training jobs (mnist, googlenet, alexnet, vgg16, vgg11) [15], and three DL inference jobs, classification, regression, and multiout of DJINN workload suite [7] were used for experiments. All DL tasks used Tensorflow, executed in the GPU and containerized as a Docker container.

3) Evaluation metrics:

- The average job completion time (JCT) is the average completion time from when each job is submitted.
- Makespan is the time when all jobs in the workload are completed.

4) Baseline schedulers: A max-memory-based Binpack scheduler and interference aware schedulers, such as the Loadbalance and Mystic [14] schedulers, are used. The Loadbalance scheduler is based on the average GPU utilization and selects the pairs to co-execute such that it has the minimum GPU utilization. The Mystic scheduler calculates the similarity among the application metrics and schedules in order of low similarity.

B. Case studies with Co-scheML

Fig. 2 shows the average JCT of a workload. At this time, the arrival interval was arbitrarily designated as 15s. The overall applications of the workload were sorted by the JCT.

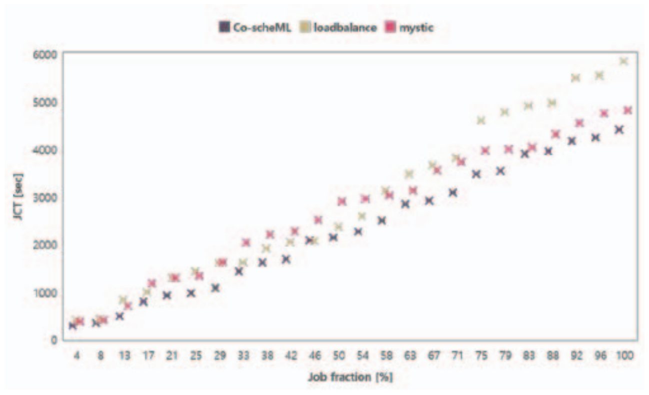

Figure 2. Performance of Co-scheML and others in Average JCT

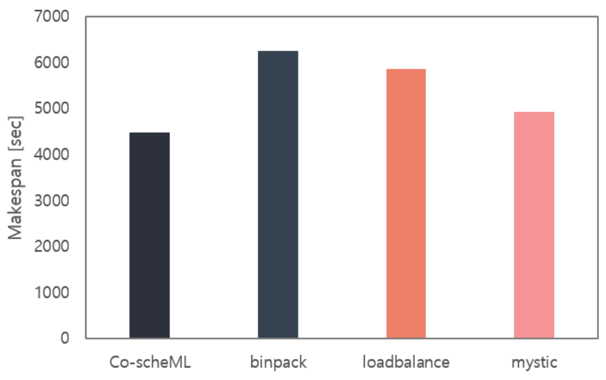

Figure 3. Performance of Co-scheML and others in Makespan

The average JCT of Co-scheML was lower than that of Loadbalance and Mystic by 32% and 12%.

Fig. 3 shows the makespan of a workload. The interval was selected as 15s. Co-scheML showed a 1.53-, 1.32-, and 1.12-fold better makespan than that of Binpack, Loadbalance, and Mystic, respectively. It was confirmed that Co-scheML achieved the best average JCT and makespan. Loadbalance shows better performance than the simplest scheduler, binpack scheduler, it has limitation of scheduling only considering average gpu utilization. Mystic scheduler is the best scheduler among baseline schedulers. However, it still has a chance to improve performance, as the scheduler doesn't deal with the degree of influence on interference for each metrics. Co-scheML allows each application to use complementary resources by considering interference, resulting in improved not only average JCT but also makespan.

VI. Related Works

A. GPU resource sharing scheduling

Many GPU sharing technologies have been introduced in an effort to improve the resource utilization of GPU clusters and cloud servers. Diab [17] proposed a system in which many users can share GPU resources while co-executing tasks by intercepting a CUDA API to enable the execution of two kernels. However, this scheduling method can only be used for applications that have repetitive or distinct forms of resource usage. In [10] and [11], the GPU resources are classified into a certain size and allocated to a container that executes a cost tree to allocate and provide the resources to the GPU. However, this only considers the minimum resource requirement of the running task, which can result in a degraded performance during a co-execution. There are cluster schedulers for DL workload [19-22]. [20] predicts the GPU utilization, PCIe bandwidth, and memory to guarantee the QoS, and minimize the energy efficiency. Although the DL workload is static and predictable, dynamic applications such as HPC applications are difficult to predict. Therefore, we conducted resource provisioning through profiling and monitoring in this study. In addition, the purpose of this study is to improve performance rather than achieve energy efficiency. Using GPU utilization, the PCIe bandwidth and memory usage alone are insufficient to improve the performance.

B. Interference-aware scheduling on GPU

With the introduction of technologies that can co-execute numerous applications on the GPU, scheduling methods to avoid resource contention that may occur during the co-execution in this environment have been proposed. [14] suggests a collaborate filtering (CF) based interference recognition scheduler for the co-execution of applications. It profiles interference metrics, and if a new application is applied, CF is used for prediction after lightweight profiling. The method used to obtain the interference values of each application pairs is based on the similarities in the vector of metrics, in which the lower the similarity, the lower the interference. Although the degree of influence of the interference for each metric is different and leads to different weights, this is not reflected. Rather, similarities are simply obtained, and the OOM failure issue that can occur during the co-execution of the GPU applications is not considered. [16] analyzes the characteristics of the DL and carries out a performance prediction modeling based on the results as a suggestion to the QoS-aware scheduler. This requires domain-specific knowledge of the DL, and thus cannot be applied to all applications, such as HPC applications. [13] defined features of the application characteristics executed on a GPU to implement an ML-based interference recognition scheduler. Although the performance of a simple application is significantly affected by the kernel length, a difference in interference for ML applications is shown. However, because the actual evaluation was conducted on ML applications that have repetitive resource usage patterns, additional feature definitions of the affected resources and other metric definitions for the container environment targeted to the VM are required. Moreover, a method for applying interference values to the cluster scheduling method is necessary because a simple round-robin scheduler was implemented. In [18], a learning placement framework using a DRL model in the cluster environment with GPU servers is proposed. The authors explained that learning is carried out by inputting the worker id, CPU, and GPU, and tasks are placed by a low level of interference when multiple applications coexist. However, detailed information on the resources is necessary to decrease the interference effect with the CPU and GPU usage values. [19] considers interference values and uses max-pair algorithms to decrease the overall workload execution time. However, the lengthy time required to execute all pairs each time a new application enters can be a disadvantage. In this study, we demonstrated that a prediction of interference is possible with existing profiling information without executing all pairs, which is not applied to dynamic scheduling.

VII. Conclusion

This paper proposes an architecture with an interference-aware scheduler, which provides a ML model that predicts interference values using application profiling information and minimizes interference among GPU applications in a

GPU cluster. The experiments with Co-scheML as a proof-of-concept show that Co-scheML reduce potential interference during co-execution of applications and improve the completion of workload execution.

Future studies include refinement and generalization of interference-aware co-scheduling for diverse workloads.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2015M3C4A7065646, No. 2020R1H1A2011685).

REFERENCES

[1] Yarn, https://hadoop.apache.org/docs/r3.1.0/hadoop-yarn/hadoop-yarn-site/UsingGpus.html

[2] Kubernetes, https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/

[3] NVIDIA Multi Process Service, https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

[4] InfuxDB, https://www.influxdata.com/

[5] NGC, https://ngc.nvidia.com/

[6] DJINN, https://github.com/LLNL/DJINN

[7] Geng X, Zhang H, Zhao Z, Ma H. Interference-aware parallelization for deep learning workload in GPU cluster. Cluster Computing. 2020 Jan 2:1-4.

[8] Chang CC, Yang SR, Yeh EH, Lin P, Jeng JY. A kubernetes-based monitoring platform for dynamic cloud resource provisioning. in GLOBECOM 2017-2017 IEEE Global Communications Conference 2017 Dec 4 (pp. 1-6). IEEE.

[9] Gu, Jing, et al. "GaiaGPU: Sharing GPUs in Container Clouds." 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, 2018.

[10] Song, Shengbo, et al. "Gaia Scheduler: A Kubernetes-Based Scheduler Framework." 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, 2018.

[11] Hong, Cheol-Ho, Ivor Spence, and Dimitrios S. Nikolopoulos. "FairGV: fair and fast GPU virtualization." IEEE Transactions on Parallel and Distributed Systems 28.12 (2017): 3472-3485.

[12] Xu, Xin, et al. "Characterization and prediction of performance interference on mediated passthrough GPUs for interference-aware scheduler." 11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19). 2019.

[13] Ukidave, Yash, Xiangyu Li, and David Kaeli. "Mystic: Predictive scheduling for gpu based cloud servers using machine learning." 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2016.

[14] Tensorflow CNN benchmarks, https://github.com/tensorflow/benchmarks/tree/master/scripts/tf_cnn_benchmarks

[15] Chen Z, Quan W, Wen M, Fang J, Yu J, Zhang C, Luo L. Deep Learning Research and Development Platform: Characterizing and Scheduling with QoS Guarantees on GPU Clusters. IEEE Transactions on Parallel and Distributed Systems. 2019 Jul 29;31(1):34-50.

[16] Diab, Khaled M., M. Mustafa Rafique, and Mohamed Hefeeda. "Dynamic sharing of GPUs in cloud systems." 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. IEEE, 2013.

[17] Bao, Yixin, Yanghua Peng, and Chuan Wu. "Deep Learning-based Job Placement in Distributed Machine Learning Clusters." IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE, 2019.

[18] Wen Y, O'Boyle MF, Fensch C. MaxPair: enhance OpenCL concurrent kernel execution by weighted maximum matching. InProceedings of the 11th Workshop on General Purpose GPUs 2018 Feb 24 (pp. 40-49).

[19] Thinakaran P, Gunasekaran JR, Sharma B, Kandemir MT, Das CR. Kube-Knots: Resource Harvesting through Dynamic Container Orchestration in GPU-based Datacenters. In2019 IEEE International Conference on Cluster Computing (CLUSTER) 2019 Sep 23 (pp. 1-13). IEEE.

[20] Gu J, Chowdhury M, Shin KG, Zhu Y, Jeon M, Qian J, Liu H, Guo C. Tiresias: A {GPU} cluster manager for distributed deep learning. In16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19) 2019 (pp. 485-500).

[21] Peng Y, Bao Y, Chen Y, Wu C, Guo C. Optimus: an efficient dynamic resource scheduler for deep learning clusters. InProceedings of the Thirteenth EuroSys Conference 2018 Apr 23 (pp. 1-14).

[22] Xiao W, Bhardwaj R, Ramjee R, Sivathanu M, Kwatra N, Han Z, Patel P, Peng X, Zhao H, Zhang Q, Yang F. Gandiva: Introspective cluster scheduling for deep learning. In13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18) 2018 (pp. 595-610).