Concurrent Workload Scheduling for Cloud Microservices using Multi-Instance GPU Utilization

1st Jonghwan Park
Department of Data Science
Korea Electronics Technology Institute
Seongnam, Gyeonggi-do, Republic of Korea
bomebug15@gmail.com

2nd Jaegi Son
Department of Computer Science
Korea Electronics Technology Institute
Seongnam, Gyeonggi-do, Republic of Korea
jgson@keti.re.kr

3rd Dongmin Kim
Department of Computer Science
Korea Electronics Technology Institute
Seongnam, Gyeonggi-do, Republic of Korea
dmkim@keti.re.kr

***Abstract*—The utilization of cloud infrastructure for microservices is steadily increasing. However, a specific drawback in certain microservices is that even though there is sufficient capacity for each GPU in the cloud environment, when one Pod preemptively acquires a GPU, it prevents other Pods created later from utilizing that GPU. In this paper, we propose a scheduling approach that leverages NVIDIA A30's Multi-Instance GPU (MIG) feature to partition physical GPUs into multiple GPU instances (GI). This allows for concurrent workload execution of microservices that require GPU resources.**

Keywords—Kubernetes, Microservice, Multi-Instance GPU, GPU Scheduling, MLOps

I. Introduction

In recent times, microservices architecture utilizing cloud infrastructure has been gradually gaining attention in the corporate and technological sectors. This architecture enhances modularity and scalability of services, offering advantages such as efficient resource management and rapid deployment. However, in such a microservices environment, where each service has various requirements, efficient allocation and management of resources have become prominent challenges.

Particularly, for microservices that require GPU resources[1], traditional virtualization technologies face difficulties in resource sharing and efficient allocation among multiple Pods. This can lead to the problem where once a Pod preempts GPU resources, other Pods created later are unable to utilize the same GPU resources[2]. Consequently, tasks requiring GPUs may experience indefinite waiting or errors, causing delays in GPU workload operations.

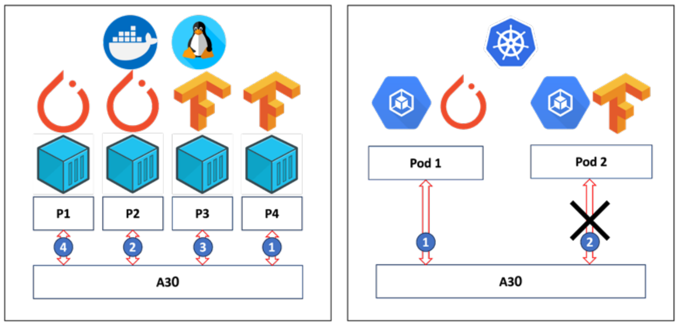

Fig. 1. Differences between local and Kubernetes GPU sharing methods.

In this paper, we propose a method for mitigating interference between concurrently executing machine learning workloads using NVIDIA's A30 graphics card's Multi-Instance GPU (MIG) feature. MIG is a technology that partitions a physical GPU into multiple virtual GPU Instances (GI), each operating in an isolated environment with dedicated resources for service provision. This enables improved task performance and stability. Through this approach, we aim to efficiently share GPU resources among multiple microservices and investigate and propose a scheduling method for concurrent workload execution. Figure 1 illustrates the phenomenon that occurs when sharing GPUs in both local and cloud environments. The utilization of MIG ensures efficiency and stability in GPU sharing within the cloud environment.

The goal of this paper is to maximize the performance and resource utilization of microservices in a cloud environment. By proposing a scheduling method based on MIG, we aim to achieve efficient distribution of GPU resources and fair resource sharing among microservices, thus harnessing the advantages of microservices architecture in the cloud environment more effectively. Additionally, this paper introduces a method for processing stable microservices with minimal resources, based on isolated GIs.

The subsequent sections will begin by reviewing relevant research trends, followed by an exploration of NVIDIA's MIG feature and its applicability in cloud microservices environments. Finally, we provide a detailed explanation of the proposed scheduling method and validate its effectiveness through experimental results.

II. Background

A. Gpu Scheduling on Kubernetes

In the past, we have conducted multi-process service (MPS) based GPU sharing experiments in traditional local and Docker environments. However, Kubernetes does not provide direct support for GPU MPS, leading to GPU resource wastage in cloud environments. Figure 2 illustrates the differences between GPU scheduling using MPS and MIG.

For this reason, numerous research cases have emerged to implement MPS in Kubernetes environments to minimize resource wastage. There are instances where custom schedulers and GPU allocation mechanisms were combined to implement GPU sharing in Kubernetes environments [3]. Additionally, there is research that modified the 'k8s-device-plugin' for collecting GPU information in Kubernetes, enabling scheduling of multiple containers (Pods) on a single GPU, and developed a resource monitoring and adaptive batch placement tool called Kube-Knot [4].

While these studies have made GPU sharing possible by modifying settings to use custom MPS schedulers, even for GPUs that do not support MIG, they have the drawback that if GPU issues arise due to conflicts between local, container, or service-level collisions, it may not prevent error propagation among services. Providing an isolated environment where errors occurring on one GPU do not adversely affect all services can ensure stable service provision.

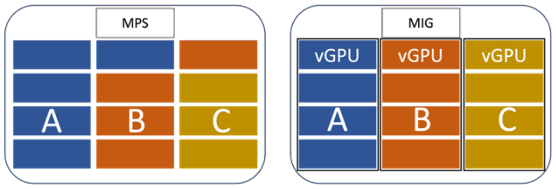

Fig. 2. GPU Scheduling in MPS and MIG.

B. MIG(Muilti-Instance GPU)

MIG(Multi-Instance GPU) is a technology that divides a single GPU device into multiple GIs (GPU Instances), providing isolated GPU environments[5]. With NVIDIA A30, up to 4 GIs can be created from a single GPU, and these divided GIs can be allocated in both Docker and Kubernetes environments. Because the units of division vary, it ensures diversity in service provision and provides isolated environments between GIs, ensuring security and quality for the services offered to users. Table 1 outlines the available GI Profiles for NVIDIA A30. 'MIG 1g.6gb+me' represents a specialized resource type for media processing tasks and offers enhanced GPU processing efficiency.

TABLE I. NVIDIA A30 MIG Profile

Profile Name	Fraction of Memory	Hardware Units	L2 Cache Size	Number of Instances Available
MIG 1g.6gb	1/4	0 NVDECs /0 JPEG /0 OFA	1/4	4
MIG 1g.6gb+me	1/4	1 NVDEC /1 JPEG /1 OFA	1/4	1
MIG 2g.12gb	2/4	2 NVDECs /0 JPEG /0 OFA	2/4	2
MIG 4g.24gb	Full	4 NVDECs /1 JPEG /1 OFA	Full	1

As products featuring MIG become available, various research studies related to MIG have emerged. In B. Li's research[6], experiments compared the training and inference speeds and energy usage of GIs on NVIDIA A100 GPUs when MIG was enabled, across various models such as ResNet50, Bert, GNN, and more. Robroek's study[7] divided experiments into three cases: Native, where users share CUDA stream memory; MPS, which isolates kernel context switching due to automatic process division by the MPS daemon; and MIG, which holds isolated vGPUs, and compared ML training time, GPU utilization, and Memory Footprint Capacity. Additionally, there are studies that have developed tools like MIGPerf[8] and MISO[9] to efficiently schedule vGPUs created by MIG, minimizing entry barriers such as manual division and resource optimization.

However, it's worth noting that the aforementioned studies were conducted in local and Docker environments rather than in a cloud environment. There is currently a lack of research comparing the performance of MIG and MPS in a cloud environment.

III. Concurrent Workload Scheduling

The purpose of this paper is to enhance the experimental environment using NVIDIA MIG to address issues arising from concurrent workload execution based on MPS in a GPU cluster. To improve ML workload execution times, experiments were conducted by measuring ML workload execution times in local MPS and MIG environments and comparing them with Kubernetes-based MPS and MIG environments. To implement concurrent ML workloads, Python-based multiprocessing was utilized in the Docker environment for time measurement, and Kubernetes employed Job-based Pod execution for deployment.

The MIG profile configuration used was 'MIG 1g.6gb,' with settings designed to enable each process to handle tasks without pending, as the number of splits increased. Each GI possessed 6016MiB of VRAM, and this profile was chosen to accommodate a large number of workloads with the smallest instance. In this study, 2 A30 GPUs were used, and the divided GIs were split into 4 GIs per A30 GPU, totaling 8 GIs used for experimentation. Additionally, NVIDIA's NVLink was utilized to enable communication between the 2 A30 GPUs, allowing the allocation of 8 GIs to a single container. An environment capable of recognizing MIG successfully was configured in the GPU cluster through the 'GPU Feature Discovery (GFD)' package provided by NVIDIA.

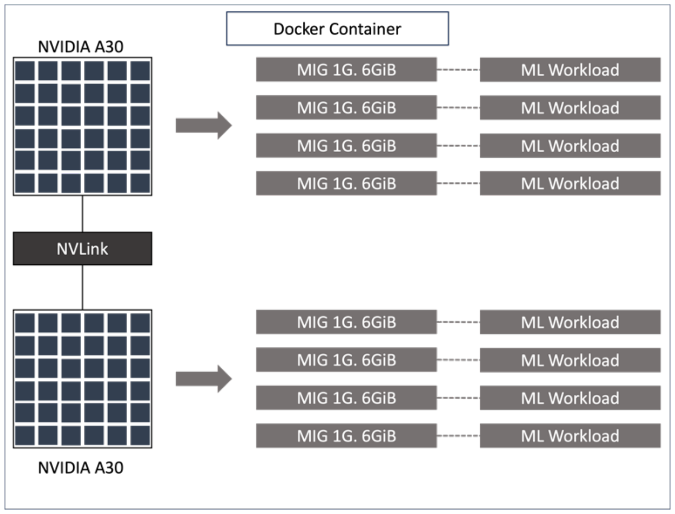

Fig. 3. NVLink MIG GPU Structure.

IV. Experiment

In this paper, the ML workload used was constructed to perform sentiment analysis by training an LSTM model on the 'Naver sentiment movie corpus' dataset. To maximize the memory usage of each GI, a batch size of 1024 was configured, and the training data was composed of 150,000 rows, with a test data ratio of 50,000 rows. For each workload, the complete tasks of data preprocessing, training, and inference were executed, with the training epochs set to 5.

The measurement environment for ML workloads was configured into three categories: GPU Batch, GPU Parallel, and MIG Parallel.

GPU Batch: This environment operates with processes using GPU resources sequentially, without duplication between them.

GPU Parallel: In this setup, processes are duplicated as per the environment requirements, allowing parallel execution.

MIG Parallel: This environment is structured to execute processes on each GI separately within the MIG setup.

TABLE II. Concurrent ML Workload Training Time at Docker

Process Num	2	4	6	8
GPU Batch	35m 11s	66m 49s	99m 21s	145m 41s
GPU Parallel	**19m 01s**	**23m 08s**	**32m 57s**	**38m 26s**
MIG Parallel	22m 49s	23m 4s	22m 31s	23m 21s

TABLE III. Concurrent ML Workload Training Time at Kubernetes

Process Num	2	4	6	8
GPU Batch	**34m 58s**	**64m 42s**	**87m 04s**	**122m 28s**
GPU Parallel	17m 43s	36m 04s	52m 51s	69m 13s
MIG Parallel	**21m 9s**	**21m 46s**	**21m 55s**	**24m 27s**

A. Docker MPS & MIG

In the case of Docker, all 8 GIs were integrated into a single container. It exhibited faster speeds in GPU Parallel compared to Kubernetes due to the fact that, in Kubernetes, when a GPU is preempted, other Pods may come to a complete halt. Docker's ability to utilize MPS contributes to its faster performance compared to Kubernetes.

B. Kubernetes MPS & MIG

While there may not be a significant difference compared to Docker, we observed a notable decrease in speed in Batch execution as the number of processes increased. Additionally, when handling multiple workloads using MIG, it is possible to achieve processing speeds similar to local environments. This signifies the ability to provide services at a faster pace and with high efficiency. Moreover, by integrating MIG into pure Kubernetes without the need for custom schedulers, it allows for the simplest and fastest service provision. Furthermore, by offering isolated GIs, it helps avoid situations where service disruptions occur across the entire GPU due to errors.

V. Result

This paper conducted research on providing services in an isolated environment for multiple workloads using NVIDIA MIG devices in a cloud environment. It was confirmed that, when using MIG in cloud environments that do not support traditional MPS operations, GPU resources can be shared without the need for custom schedulers. Furthermore, when comparing performance with local environments, it was observed that there is not a significant difference when applying it to actual services. Using MIG, it is possible to provide services with guaranteed stability through isolated GIs. Currently, NVIDIA offers MIG on three GPU models: H100, A100, and A30. As GPUs with better capacity than A30 become available, it will be possible to handle a larger volume of ML workloads for large-scale services in the cloud.

Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2022-0-00047, Development of microservices development/operation platform technology that supports application service operation intelligence)

References

[1] Y. Zhou, Y. Yu and B. Ding, "Towards MLOps: A Case Study of ML Pipeline Platform," 2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE), Beijing, China, 2020, pp. 494-500, https://doi.org/doi:10.1109/ICAICE51518.2020.00102.

[2] Zeineb Rejiba and Javad Chamanara. 2022. "Custom Scheduling in Kubernetes: A Survey on Common Problems and Solution Approaches," ACM Comput. Surv. 55, 7, Article 151 (July 2023), 37 pages. https://doi.org/doi:10.1145/3544788

[3] Shaoqi Wang, Oscar J. Gonzalez, Xiaobo Zhou, Thomas Williams, Brian D. Friedman, Martin Havemann, and Thomas Woo. 2020. "An efficient and non-intrusive GPU scheduling framework for deep learning training systems," In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20). IEEE Press, Article 90, 1–13. https://dl.acm.org/doi/10.5555/3433701.3433820

[4] P. Thinakaran, J. R. Gunasekaran, B. Sharma, M. T. Kandemir and C. R. Das, "Kube-Knots: Resource Harvesting through Dynamic Container Orchestration in GPU-based Datacenters," 2019 IEEE International Conference on Cluster Computing (CLUSTER), Albuquerque, NM, USA, 2019, pp. 1-13, https://doi.org/doi:10.1109/CLUSTER.2019.8891040

[5] "NVIDIA Multi-Instance GPU," NVIDIA, accessed Sep 1, 2023,https://www.nvidia.com/en-us/technologies/multi-instance-gpu/

[6] B. Li, V. Gadepally, S. Samsi and D. Tiwari, "Characterizing Multi-Instance GPU for Machine Learning Workloads," 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lyon, France, 2022, pp. 724-731, https://doi.org/doi:10.1109/IPDPSW55747.2022.00124.

[7] Robroek, Ties, Ehsan Yousefzadeh-Asl-Miandoab, and Pınar Tözün. "An Analysis of Collocation on GPUs for Deep Learning Training." arXiv e-prints (2022): arXiv-2209. https://doi.org/10.48550/arXiv.2209.06018

[8] Zhang, Huaizheng, et al. "MIGPerf: A Comprehensive Benchmark for Deep Learning Training and Inference Workloads on Multi-Instance GPUs." arXiv preprint arXiv:2301.00407 (2023). https://doi.org/10.48550/arXiv.2301.00407

[9] Baolin Li, Tirthak Patel, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. i. MISO: exploiting multi-instance GPU capability on multi-tenant GPU clusters. In Proceedings of the 13th Symposium on Cloud Computing (SoCC '22). Association for Computing Machinery, New York, NY, USA, 173–189. https://doi.org/10.1145/3542929.3563510