COLTI: Towards Concurrent and Co-located DNN Training and Inference

Jaiaid Mobin[1]
Rochester Institute of Technology
Rochester, New York, USA
jm5071@rit.edu

Avinash Maurya[1]
Rochester Institute of Technology
Rochester, New York, USA
am6429@cs.rit.edu

M. Mustafa Rafique
Rochester Institute of Technology
Rochester, New York, USA
mrafique@cs.rit.edu

ABSTRACT

Deep learning models are extensively used in a wide range of domains, e.g., scientific simulations, predictions, and modeling. However, training these dense networks is both compute and memory intensive, and typically requires accelerators such as Graphics Processing Units (GPUs). While such DNN workloads consume a major proportion of the limited onboard high-bandwidth memory (HBM), they typically underutilize the GPU compute resources. In such scenarios, the idle compute resources on the GPU can be leveraged to run pending jobs that can either be (1) accommodated on the remainder HBM, or (2) can share memory resources with other concurrent workloads. However, state-of-the-art workload schedulers and DNN runtimes are not designed to leverage HBM co-location to improve resource utilization and throughput. In this work, we propose *COLTI*, which introduces a set of novel techniques to solve the aforementioned challenges by co-locating DNN training and inference on memory-constrained GPU devices. Our preliminary evaluations of three different DNN models implemented in the PyTorch framework demonstrate up to 37% and 40% improvement in makespan and memory utilization, respectively.

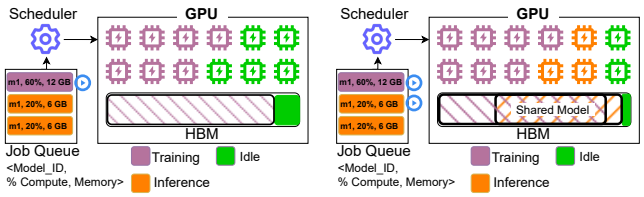
KEYWORDS

Systems in AI; GPU; Shared Memory; Neural Networks; Job Scheduler; Deep Learning

ACM Reference Format:
Jaiaid Mobin[1], Avinash Maurya[1], and M. Mustafa Rafique. 2023. *COLTI*: Towards Concurrent and Co-located DNN Training and Inference. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '23), June 16–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3588195.3595940

1 INTRODUCTION

Deep learning (DL) based services and knowledge discovery are accelerating a broad range of use cases, from businesses and commerce to research industry and defense systems. DNN with large size (e.g. LLM, stable diffusion) has gained rapid traction from a diverse set of audiences for personal and professional assistance. However, such DL models are compelled to undergo continuous training processes to learn from the complex data patterns of different users and/or applications. Additionally, these models often have to serve inference requests at a rate high enough to saturate the system resources. While accelerators, such as GPUs, are imperative to run DL training and inference workloads due to their embarrassingly parallel SIMT architecture, even when the compute resources are underutilized, the limited onboard GPU HBM restricts the number of jobs that can be run concurrently for improving utilization of expensive resources.



(a) Baseline System **(b) Co-located System**

Figure 1: Comparison of GPU HBM occupancy and compute resource occupancy scenarios.

In scenarios where multiple training and inference jobs are enqueued in the job scheduler, as shown in Figure 1a, the scheduler typically dispatches these jobs sequentially to the GPU or allows for co-location only when the desired number of memory and compute units are available on the GPU. However, from the characterization of Alibaba traces [2], only 7% of GPUs have a utilization rate of more than 95%. Another finding is that about 80% of high GPU utilizing tasks, e.g., image processing and NLP tasks, use less than one GPU. Based on these findings, it can be surmised that co-locating DL jobs that have little to no impact on the I/O (host-to-device) and HBM allocations of the existing jobs can lead to significant improvements in system throughput and job completion times. However, such co-location necessitates coordination between the job scheduler and the DL workloads running on different GPUs to determine the best candidate for co-location with minimal interference between the co-located workloads.

This challenge of resource underutilization and memory-compute skew on GPU devices has been extensively studied in the past. A commonly adopted approach is to use independent clusters for inference and training. While this approach is simple and effective in controlling service-level agreements (SLAs) of each workload, it suffers from inadequate system throughput and unbalanced cluster utilization during load spikes in training or inference jobs. Techniques such as GSLICE [1] attempt to improve resource utilization by co-locating multiple inference jobs by sharing a single DL model

[1]Student author

HPDC '23, June 16–23, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0155-9/23/06.
https://doi.org/10.1145/3588195.3595940

across multiple jobs. However, none of them contribute towards the co-location of both training and inference workloads. To this end, we study the system of sharing the DL model across both training and inference workloads to improve system throughput and reduce job completion times.

2 SYSTEM DESIGN

2.1 *COLTI* Architecture

To co-locate and run multiple DL training and inference jobs concurrently, *COLTI* proposes the following set of modules:

- **Model-aware Job Scheduler:** We propose a specialized job scheduler that records the meta-data pertaining to model characteristics in addition to launch parameters. This design enables us to efficiently co-locate multiple jobs running on the same model. As outlined in Figure 1b, the job queue accepts a reference to the model ID, e.g. $m1$, which is being utilized by each of the enqueued workloads.
- **Inter-process CUDA Memory Sharing:** The inter-process communication (IPC) module of the CUDA runtime allows for independent processes to share memory using the IPC handles. We leverage this functionality to store the location of different models loaded on the GPU HBM, thereby allowing the co-located jobs to share models.

2.2 Implementation

We implement *COLTI* in *PyTorch v1.13*. We expose a single API that can be invoked during model loading, to lookup the location of the shared model. If a model is not found on any GPU HBM, then it is loaded from the disk into the GPU, and its corresponding CUDA IPC handle is cached for future jobs (that arrive before the completion of the current job).

3 PRELIMINARY EVALUATION

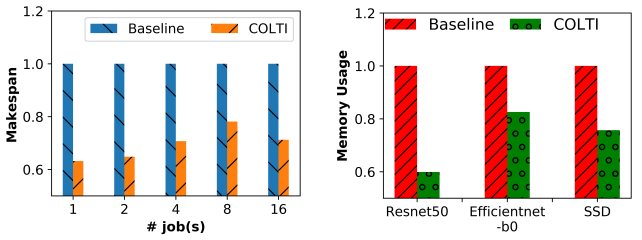
3.1 Evaluation Setup

We evaluate our proposal on a single node with Nvidia GeForce RTX 3060 GPU (12 GiB HBM). Our server contains a single-socket Intel(R) Xeon(R) W-2145 CPU and 32 GiB DRAM, running a total of 16 threads (hyper-threading enabled). We use Ubuntu 20.04.6 LTS, CUDA version 11.7, Python version 3.8, and the `/dev/shm/` directory is used for caching model IPC handles.

3.2 Methodology

Throughout our evaluations, we compare our proposed *COLTI* based DL model-sharing approach against the baseline, which is representative of sequential scheduling when required memory is not available, as observed in real-world scenarios. In our evaluation, we use three popular DL models, i.e., Resnet50, Efficientnet-b0, and SSD. For each of the aforementioned approaches and models, we measure the following:

(1) Makespan and throughput for varying number of processes.
(2) Memory utilization when two jobs are co-located and share the same model.
(3) Accuracy of the training and inference workloads [shown only in the poster due to limited space].



(a) Makespan for increasing number of processes **(b) Memory utilization for co-located shared-model jobs**

Figure 2: Makespan and memory utilization normalized with respect to the baseline approach. Lower is better.

3.3 Experimental Results

3.3.1 Makespan. We measure the overall makespan of all enqueued jobs in the scheduler. As observed in Figure 2a, for an increasing number of jobs, the overall makespan using *COLTI* is between 22% and 37% faster as compared to the baseline approach. Moreover, for 16 jobs co-located on the same GPU, we observe a significantly higher baseline makespan as compared to the case of 8 jobs, due to which *COLTI* performs better for 16 jobs.

3.3.2 Memory Utilization. We evaluate the difference in the memory utilization of the GPU when a single inference job is co-located with the already running training job. Figure 2b depicts the amount of HBM savings achieved for different models, ranging from 17% to 40%. While we expect to observe a flat 50% memory saving since both jobs use the same model, and the inference job typically has negligible input size (as compared to the model), this is not the case because PyTorch has an independent caching mechanism that interferes with the expected memory savings.

4 CONCLUSION

We design and develop *COLTI* to co-locate model-sharing between training and inference jobs to increase system throughput in GPU memory-constrained environments. Our preliminary evaluations yield up to 37% faster job completion time as compared to the baseline approaches. Currently, our system is for a single GPU, single node. In the future, we will extend *COLTI* to multi-GPU settings. Current design should be applicable for distributed training framework unless the model is distributed across nodes. How co-location will affect the jobs individual performance (e.g. latency) is another scope of future investigation.

5 ACKNOWLEDGEMENT

This work is supported by the National Science Foundation (NSF) under Awards No. 2106634 and 2106635.

REFERENCES

[1] Aditya Dhakal, Sameer G Kulkarni, and KK Ramakrishnan. 2020. Gslice: controlled spatial sharing of gpus for a scalable inference platform. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 492–506.
[2] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. [n. d.]. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *In Proc. of NSDI'22*. USENIX Association.