Summary of "Tiresias: A GPU Cluster Manager for Distributed Deep Learning"

Xiangfeng Zhu(zxfeng), Jiachen Liu(amberljc), Chris Chen(zhezheng)

Problem and Motivation

The authors argue three primary challenges faced by distributed deep learning(DDL) scheduler in production. 1) Unpredictable job duration: Some of the existing schedulers try to predict the DL job training times by assuming DL jobs have smooth loss curves. However, because of the trial-and-error characteristic of DL jobs, their loss curves are not as smooth as the curves of the best model ultimately picked at the end of exploration. Thus, the scheduler should not rely on the loss curve for predicting eventual job completion time. 2) Over-aggressive job consolidation: Because DL jobs are sensitive to GPU locality, many existing solutions assign all components of the job to the same or the minimum number of servers. As a result, jobs often wait when they cannot be consolidated, even if there are enough spare resources elsewhere in the cluster. and 3) Time overhead of preemption: The common way to preempt Unlike preemption in CPU, GPU preemption usually takes tens of milliseconds.

Solution Overview

The main objectives of Tiresias are 1) minimizing the average job completion time(JCT), 2) achieving high GPU utilization and 3) avoiding starvation

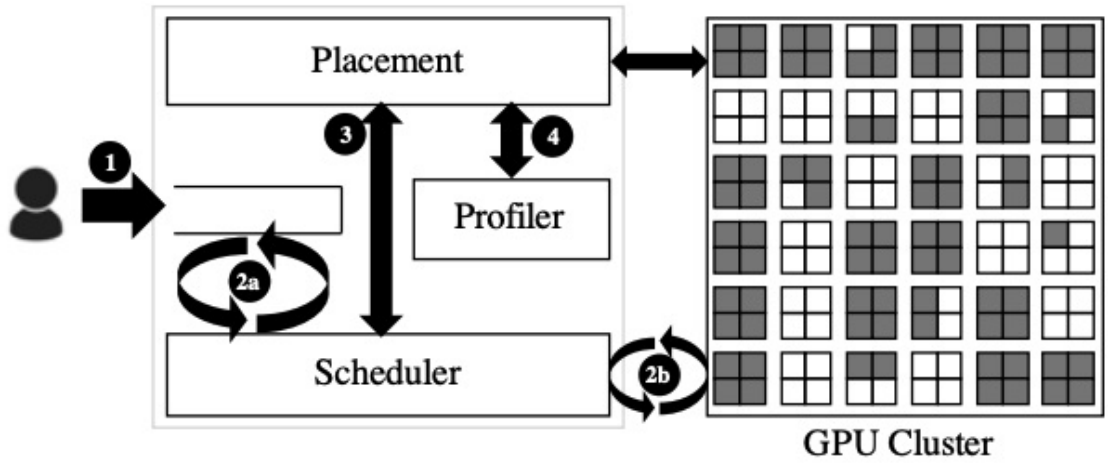
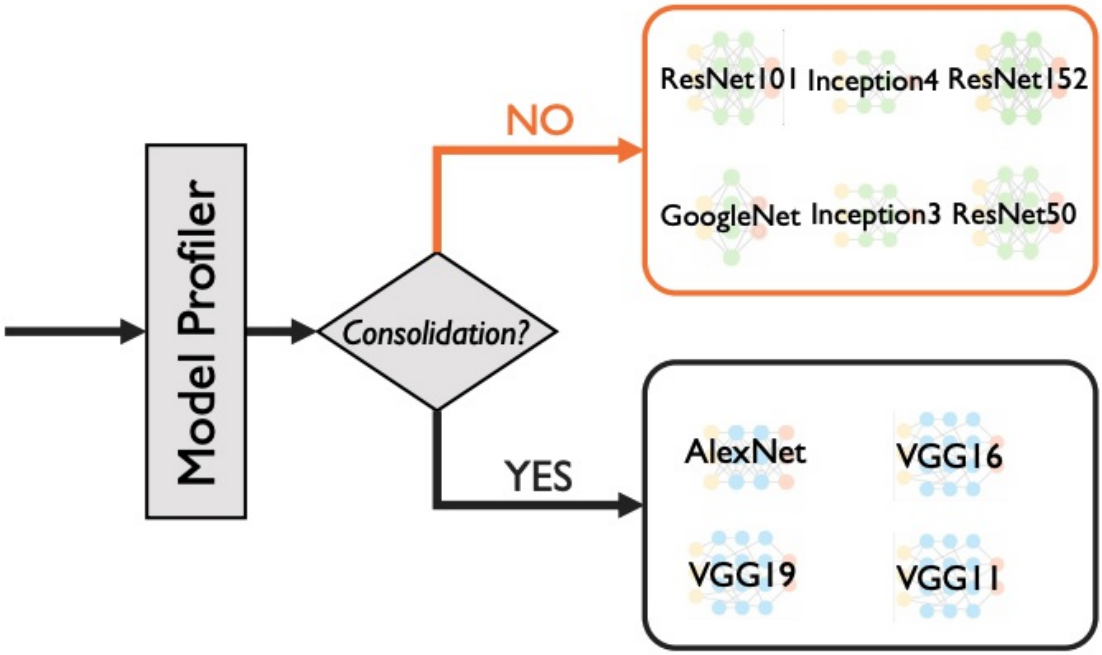


Figure 6: Tiresias components and their interactions. Job lifecycle under Tiresias is described in Section 3.1. In this figure, each machine has four GPUs; shaded ones represent GPUs in use.

To address the aforementioned challenges, Tiresias uses an aged based scheduler called Two-dimensional Attained Service-Based Scheduler(2DAS). 2DAS assigns each job a priority based on its attained service. The attained service of a job is calculated based on the number of GPUs it uses and the amount of time it has been running so far.

When no job duration information is provided, the priority function applies the Least-Attained-Service(LAS) algorithm where a job's priority is inverse to its attained service. If the distribution of job duration is provided, then a job's priority equals its Gittins index value.



Using continuous priorities can lead to a sequence of preemptions(preemption is both time-consuming and expensive in GPUs) and subsequent resumptions for all jobs. Tiresias address this challenge by using the classic Multi-Level Feedback Queue algorithm.

Summary of Class Discussion

Tiresias, along with other existing DL scheduler, seems only considers GPU for training, but, as we see in the Facebook paper, training can also be performed on CPUs for cost-effective reasons. Thus, will the existing scheduler work well in these CPU-GPU heterogeneous clusters?

Tiresias assumes the training jobs can only run on GPUs. Because CPUs have different characteristics than GPUs, we have to redesign Tiresias in these CPU-GPU heterogeneous clusters.

There is a paper in this year NSDI called Themis, which proposes another DL scheduler, but its objective is fairness rather than average JCT. Which objective makes more sense for DL jobs?

Tiresias assumes that jobs are executed in a private cluster. In general, in private clusters, it makes more sense to use JCT as the main objective, whereas, in shared clusters, fairness should be the primary goal.

One way to mitigate this performance and fairness tradeoff is to use a two-level scheduler. The scheduler ensures fairness across organizations and minimize JCT within an organization.

Summary of "Gandiva: Introspective Cluster Scheduling for Deep Learning"

Problem and Motivation

Existing schedulers mainly treat deep learning training(DLT) job as yet another big-data job that is allocated a set of GPUs at job startup and holds exclusive access to its GPUs until completion. However, it is inefficient for two major reasons: 1. Head-of-line blocking, as long DLT jobs can run for days and 2. Low Efficiency. The job placement is fixed at startup time, but some DLT jobs are sensitive to locality.

The authors point out several unique characteristics for DLT jobs.

1. Locality: The performance of a multi-GPU DLT job depends on the affinity of the allocated GPUs. Different DLT jobs exhibit different levels of sensitivity to inter-GPU and intra-GPU affinity. (Two GPUs in the same machine might be located in different sockets or PCIe switches.)

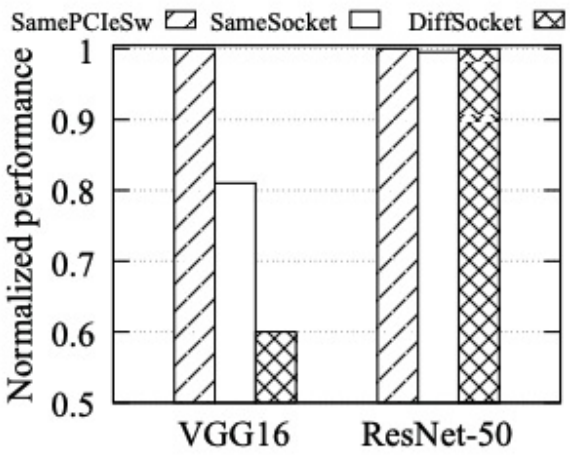


Figure 1: Intra-server locality.

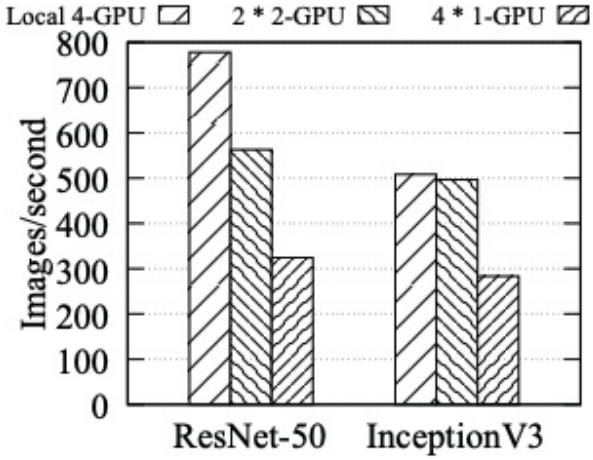


Figure 2: Inter-server locality.

2. Interference: When running in a shared execution environment(e.g., PCIe switch), DLT jobs might interfere with each other due to resource contention, and, again, different jobs have different degree of interferences. For example, when two language model(LM) jobs run together, both jobs suffer 19% slowdown. However, ResNet-50 does not suffer from GPU co-location with LM.

3. Intra-job predictability: A DLT jobs consists of numerous mini-batch iterations. Thus, the GPU memory used clearly follows a cyclic pattern, where each cycle corresponds to the processing of a single mini-batch. The maximum GPU memory used can be an order of magnitude larger than the minimum memory used. e

Solution Overview

To address the aforementioned problems, this paper proposes Gandiva, a cluster scheduling framework that utilizes DL-specific characteristics to improve latency and efficiency in a GPU cluster. Gandiva removes the exclusivity and fixed assignment of GPUs in the following ways:

- Time-Slicing: DLT jobs are split into 60s subtasks and Gandiva allows incoming jobs to time-share GPUs with existing jobs. Leveraging the cyclic pattern of DLT jobs, when a suspend is issued, Gandiva waits until the minimum of the memory usage cycle. In the evaluation, the authors show that this suspend-and-resume can be accomplished under O(100ms). Packing, used only during overload, is another mechanism than allows multiple DTL jobs run on the same GPU simultaneously and let the GPU time-share the jobs.

Packing is efficient only when the packed jobs do not exceed the GPU resources and do not negatively interfere with each other.

- Migration: Migration can improve the efficiency by 1) moving time-sliced jobs to vacated GPUs 2)migrating interfering jobs away from each other and 3) de-fragmentation of the cluster so that incoming jobs can get GPU locality. It is implemented using model checkpoints. Migration happens when a job departs and Gandiva pick jobs that are not co-located and try to find a new co-located placement.

Limitation and improvement

1. Gandiva could use more prior knowledge to profile jobs and avoid performance degrades after migrating or packing.
2. Gandiva doesn’t take average job completion time and job starvation into account (in Q&A discussion).

Summary of Class Discussion

For time-slicing strategy of Gandiva, is it possible that small jobs will interrupt the jobs in cluster most of the time?

Yes, the long-running DLT jobs will be interrupted by the flow of incoming short jobs under time-slicing strategy. Therefore, some strategies have to be applied to avoid job starvation. One example might be: when job's pending (waiting) time exceeds a threshold (which means the job has been waitting for too much time), it should have higher priorities or get more time slots in the time-slicing strategy.

What kind of packing is Gandiva using?

Common packing : run multiple DLT jobs on a GPU simultaneously and let the GPU time-share the jobs.