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

Joshua Segal (jbseg), Han You (youh), Shucheng (joezhong)

Problem and Motivation

Deep learning(DL) nowadays often involves training large datasets over distributed clusters. While distributed clusters are commonly shared among many users and thus require an efficient cluster manager to schedule DL jobs submitted by the users, current resource schedulers are quite primitive, which mainly relies on inaccuracy prediction on the time of completion for DL tasks based on oversimplified assumptions like assuming all deep learning model has smooth loss curve, misleading the schedulers to generating a sub-optimal scheduling plan. Another issue related to current cluster managers is its over-aggressive strategy in terms of consolidation. Current solution always attempts to consolidate jobs onto the minimum number of servers even if there are resources available for the jobs. Those designs result in high job completion time (JCT) and resources underutilization. The proposed solution, Tiresias, is motivated by these problems and aims to provide scheduling algorithms that minimizes the average JCT, while maximizing the resources utilization.

Hypothesis

The first hypothesis of Tiresias is that DDL jobs information may be unpredictable. To achieve low average JCT and high cluster utilization and also to avoid starvation, preemptive scheduling is needed so that the high-priority jobs won't be blocked by the low-priority jobs. Previous solutions only take either spatial (resources required) or temporal (job execution time) aspects when assigning priorities to the jobs, but neither spatial or temporal approaches may fail in some scenarios. Therefore Tiresias combines both of the aspects to schedule the jobs which use the shortest executed time to prioritize jobs.
The second hypothesis made by Tiresias is not all deep learning jobs require consolidation. Tiresias only applies consolidation to the jobs which are sensitive to the resources affinity in order to enhance resource utilization. Since most of the models remain constant during training, Tiresias adopts the model-profiled approach by profiling models with first several iterations to determine whether to apply consolidation or not.

Solution Overview

Tiresias places focus on both the users of the system, and the operator of the system, with two different sets of goals:

1. For the operator, the objective is to
    a. Minimize the average job complete time for all jobs in the cluster.
    b. Utilize the GPUs in the cluster as much as possible.
2. For the users, an additional objective is to avoid arbitrarily lengthy starvation.

2DAS: Two Dimensional Age-based Scheduler, where the two dimensions refer to time and space: duration and resource usage of the jobs.

- If no prior information on a job's duration, schedule based on 2D-LAS (Least-Attained Service) which is to prioritize jobs with the smallest core-hour = number of GPUs × time running.
- If the cluster can provide prior distribution of jobs' duration, a job's priority is equal to its Gittins index, which is the optimal policy to the multi-armed bandit problem, or how likely a job can complete within the next service quantum.
- GPU preemption comes with overhead. Therefore, Tiresias discretizes priorities with the Multi-level Feed-back Queue (MLFQ) algorithm. The jobs are placed into K queues with decreasing priority. When a job starts (not *submitted*), it is put into the highest priority queue and later demoted to other queues as attained service increases. Within each queue:
  - The scheduling is FIFO for 2DAS.
  - The jobs are scheduled according to their Gittins indices for Gittins.
- With 2DAS, jobs are promoted to the highest priority queue from time to time to avoid starvation. The threshold of promotion is controlled by a simple ratio between waiting time and running time.
- Some models benefit a lot from consolidation, i.e. allocate them as closely as possible. A good heuristic is how skewed the model is in tensor size: a highly skewed model should be consolidated. Tiresias profiles the first few iterations of a job to determine the amount of skewness, and if that passes some threshold, consolidates the model. The threshold is periodically calculated with a linear classifier.

Limitations and Possible Improvements

1. Tiresias uses the metric of the average job completion time (JCT) to measure the performance of a scheduler, but there also exists some other metrics such as average queueing delay. The paper omits the reasoning of choosing the job completion time for evaluation. Also, Tiresias uses many hyperparameters such as priority queue number and promotion knob, but the paper lacks formal analysis of how to configure those parameters that may vary across clusters.
2. The skewness threshold used to determine whether to consolidate or not relies on profiling past jobs and it is currently determined by a simple linear classifier. More sophisticated dynamic mechanisms can be interesting future work.
3. Tiresias uses priority discretization to reduce the overhead of preemption. In that case, some techniques of lightweight preemption can help.
4. Currently Tiresias applies either Gittins index or LAS. It might be helpful to integrate these two approaches to the cases when only some of the jobs' distributions are known in advance.

Summary of Class Discussion

1. Do we need to consider fairness when scheduling these jobs?

- Yes, starvation is avoided as shown in the paper. This is more important in public clusters than in private clusters or interactive systems, even less important for batch jobs. Furthermore, the definition of fairness varies: progress fairness, complete time fairness, etc.

2. If priority changes in realtime, do preemptions happen very often?

- Priority swapping is used to prevent preempting too frequently. Described in the paper as discretization.

3. Is job switching transparent to the users?

- Users are not aware of the switchings.

4. What if we only have the historical data for some of the jobs? How do Gittins/LAS work in that case?

- Possibly per-job scheduling or classify jobs to use either priority functions. Theoretical analysis is hard when both are used. A possible method is to use Gittins and learn the information on-the-fly.

5. Is it possible to use Tiresias with model parallelism or even pipeline parallelism? What was the main motivation for data parallelism?

- Yes, Tiresias supports model parallelism or even pipeline parallelism. At the high level, Tiresias only schedules jobs based on its resource requirements. Once the resource requirements are satisfied, how to use these resources in detail (model/pipeline parallelism) is kind of decoupled from the scheduler itself. The authors choose to use data parallelism as they think it's the most popular one.

Summary of HiveD

Joshua Segal (jbseg), Han You (youh), Shucheng (joezhong)

Problem and Motivation

Current GPU reservation algorithms only guarantee the number of resources per tenant. For deep learning and other related tasks, it's important to reserve GPUs with affinity for it has a large impact on its performance. For example, a DL job on two GPUs can be five times slower when they're distributed versus when the GPUs are on the same machine.

In current reservation algorithms, training jobs wait much longer in a multi-tenant cluster than in a private cluster. This has made many users stop using shared clusters and just buy their own private one. GPU reservation on multi-tenant clusters is based on quota where the only guarantee is the number of GPUs, not affinity. This gives DL jobs two options; either wait for the affinity requirement to be met which causes long queueing delays or train with a relaxed affinity which causes slow train speed.

A new reservation method is needed to optimize affinity and decrease fragmentation is multi-tenant clusters.

Hypothesis

If we create an abstraction of a cluster that captures the idea of affinity, we can use those to reserve clusters with more affinity and improve deep learning training.
If we're able to split and merge clusters in our reservation algorithm, we can reduce fragmentation.

Solution Overview

HiveD has the goals of scheduling to optimize affinity and completion time. To achieve this, the first thing they do is virtualize clusters to abstract all of this logic from the users. Users simply specify how many GPUs they need. The main abstraction that the user deals with is a cell. A cell has different levels based on how many GPUs are in it and GPU affinity. GPU affinity varies based on if the GPUs are in the same machine and there method of communication (PCI, sockets, etc.). HiveD strives for affinity because that matters a lot in machine learning, so it will try to schedule so that the same task is run on GPUs that are on the same machine or close to each other.

HiveD prevents fragmentations in clusters by using the Buddy Cell Algorithm for cell allocation and deallocation. This is all abstracted away from the tenants, and is decided on the physical cluster layer. The Buddy Cell algorithm will first see if there is an available cell that's already the size that was requested. If there's not it will find a larger cell and recursively split it, until it gets the cluster size it needs. Once that task is complete, the cell is released and is merged back with the other cells. Keeping cells as large as possible is important to reduce fragmentation.

The Buddy Cell Algorithm is also extended to support low priority jobs for higher cluster utilization. HiveD maintains two views of the cells; a high priority view and a low priority view. When low priority tasks are scheduled to a cell, they can be put on pause by a higher priority task that needs to be run. Priority is something designated by a third party scheduler and could be something like Tiresias. The Buddy Cell Algorithm will try to schedule the low priority tasks far from the high priority tasks to minimize the chance that they will be put on pause. Putting low priority tasks on hold for higher priority tasks is necessary so low priority jobs wont impact the safety of high priority jobs.

Limitations and Possible Improvements

Some limitations is that the resources aren't used to capacity. This can happen when you have some scrap GPUs left that don't meet the affinity requirements for any job in the queue. In that case, those GPUs are left idle and underutilized.

A possible improvement to HiveD is making sure a GPU is never idle while there is a task in the queue. You could at least get the next task running on the leftover GPUs even if it doesn't have good affinity or the resource needs for the task. At the very least, the task can get started which is better than having the task just wait in line. When the next cell that fits this tasks need is available, we move it there.

Summary of Class Discussion

1. HiveD is works very similarly to OS memory management in how it deals with fragmentation and virtualization
    a. Both HiveD and MMU use virtual memory to abstract the complexity of the physical layer
    b. MMU uses fixed page sizes in order to prevent fragmentation, while HiveD only allows powers of 2 sized clusters. HiveD also has a system that merges clusters when possible which helps prevent fragmentation.