Self-Adaptive Distributed TensorFlow

CPSC 490 Proposal - Anushree Agrawal & Kristina Shia

Introduction

In the army, units that are moving through a battlefield are constantly generating data that can analyzed to gain a deeper understanding of adversarial environments and ultimately inform military strategy. For example, drones flying ahead, generating visual and auditory data, can be critical in identifying enemies or areas of danger for troops. To do this, data has to constantly be sent to a distributed system to be analyzed, and eventually, machine learning models have to be retrained on this data. Oftentimes, the efficiency of online retraining of complex models can be dependent on the availability of resources and performance of various parts of the system; certain tasks or machines may be significantly slower than others, thereby increasing the overall execution time. However, in an open battlefield, where network infrastructure may not be stable and resources may be constrained, some analytics and retraining jobs may take even longer because of unstable connections and random error. In our project, we plan on creating a system that sits on top of a currently used distributed ML model training platform, Spark. Our system will predict when certain training jobs on specific hardware may take longer than others, and then attempt identify and execute the most appropriate remediation. As a result, we hope to optimize model retraining efforts, even with unstable infrastructure.

Deep Learning Workloads

Many modern deep learning systems work by processing large sets of numerical computation often with the help of libraries like TensorFlow. Various algorithms can be implemented using operations provided by TensorFlow; the execution of tasks can distributed and also be optimized by modifying what devices are being used, as well as other parameters. Other libraries like PyTorch also provide related functionality and APIs for optimizing execution and task distribution.
Existing Orchestration Systems

Systems, like Spark, currently exist to orchestrate distributed deep learning model training. Spark is a “general-purpose data processing engine” which processes data in-memory, a technique that makes it faster than MapReduce [Spark]. It is utilized in systems around the world to orchestrate simple and fast data processing across virtual and physical machines. Spark is often used for generating features for machine learning models, and then training models with these features, as well as optimizing the performance of models by tuning hyperparameters or execution resources.

Problems

However, a key assumption with most systems built on Spark and TensorFlow is that they rely on stable infrastructure. Consider a military use case in which battalions are in the field actively collecting data that needs to be analyzed and incorporated into the model. They are sending this data back to multiple systems, that then need to retrain learning models with new data on the fly. With this example, there could be many reasons why a training job may fail, such as unstable network connections, low bandwidth, or lost data. Currently, systems are not optimized to retrain models while operate in environments with unstable infrastructure.

Additionally, systems currently exist to train ML models in a distributed fashion, but training time is still affected by the presence of “straggler” tasks, which are tasks in a job that take significantly longer than the rest of the tasks in the job [Stragglers]. A synchronous approach, where workers communicate amongst themselves to update parameters as specific tasks are run, is bounded by the slowest job in completion time. Asynchronous parameters end up having a lot of noise, because workers can easily get out of sync, especially with a straggler. Stragglers appear for reasons including misconfiguration (i.e., a task that is very resource intensive is being trained on a very small CPU, when it should be trained on a GPU/TPU), hardware degradation, and overloaded nodes, or improper task scheduling [Stragglers]. In our case, stragglers can also appear because the infrastructure we are working with is unstable. Unfortunately, it is hard to identify which tasks will be stragglers in a specific run of a job on specific hardware before the job is run.
Research Direction

To address these problems, we plan on creating a system that integrates with Spark that will figure out which tasks are most likely to be stragglers before they are run, and apply a remediation to prevent the job from becoming a straggler. We plan on creating two ML models, one to detect the straggler and the second to classify the straggler and figure out which remediation will be the best. We then plan on using the models to aid our modified Spark system to optimize deep learning workflows by preventing stragglers.

First, we will create sample workflows, which are basically jobs with subtasks that have to be run, probably on Spark. For each workflow, we plan on injecting latency, adding failure, starting long running tasks, or limiting resources to make sure we have a subtask that ends up being the straggler.

Second, we will work on feature generation. We plan on generating features based on task data (i.e., data size, how the job is divided into tasks, etc) and node data (CPU, memory, etc).

Third, we will train our model to detect the straggler using supervised learning - we will identify the stragglers in our example workflows, and train our model to detect them.

Fourth, we will train our second model to classify the straggler and determine which remediation method will be the best. The “best” remediation method will be the one that can reduce the time the straggler takes to be run.

Fifth, we’ll implement our system that integrates with the Spark system to use our models to detect and fix stragglers in ML model training.

Citations

[Stragglers]
https://people.eecs.berkeley.edu/~kubitronecscs262a-F12/projects/reports/project3_report_ver2.pdf

[Spark]

[TF]
https://www.tensorflow.org/

Additional Sources

https://www.oreilly.com/ideas/distributed-tensorflow
https://www.slideshare.net/databricks/operationalizing-edge-machine-learning-with-apache-spar
k-with-nisha-talagala-and-vinay-sridhar
https://thenewstack.io/configuration-management-orchestration/
https://medium.com/netflix-techblog/meson-workflow-orchestration-for-netflix-recommendations-
f932625c1d9