ABSTRACT

Execution of operations in distributed machine learning systems has largely ignored dependencies between communication and computation ops. In this paper, we make the case that model-aware ordering of operations at individual machines can decrease the step time of training iteration in distributed machine learning systems while also improving network utilization. The contributions of this work are:

- We introduce a metric for quantitatively measuring the efficiency of ordering of ops (§1).
- We propose an ordering heuristic for Model-Replica with Parameter Server systems (§2.1).
- We chalk out a roadmap for developing fast heuristics for model-aware ordering of ops in Model-Replica systems with all-reduce synchronization (§2.2).
- We evaluate our ordering mechanism on Model-Replica systems with Parameter Server on TensorFlow and show that the training efficiency can be improved by up to 78% through better ordering of tasks with 46% reduction in step time (§3).

1 PROBLEM DEFINITION

Graph-Based Machine Learning Systems such as TensorFlow [1] and PyTorch [12] evolved as a response to the growing complexity of problems that artificial intelligence is trying to solve. In these systems, a machine learning model (model) is represented by a directed acyclic graph (DAG) with predefined operations (ops) as nodes and their data/control dependencies as edges.

When a distributed model is sent for execution, each op is assigned a tag [1]. This tag determines the device on which the op will run. Using the type of op, we can also infer the associated cost of running an op on a resource, the upper and lower boundaries of the makespan respectively are:

\[
\text{Cost}(G) = \sum_{\text{op} \in G} \text{Cost}(\text{op}), \quad \text{Cost}(G) = \max_{r \in R} \sum_{\text{ops} \in G_r} \text{Cost}(\text{op})
\]

1

where \( G \) is the set of all ops, \( R \) refers to all resources, and \( G_r \) refers to all ops assigned to the resource \( r \). For a given makespan, \( t \), "Ordering Efficiency" (\( E \)) is defined as:

\[
E(G, \text{Cost}, t) = \frac{\text{Cost}(G) - t}{\text{Cost}(G) - \text{Cost}(G)}
\]

With Ordering Efficiency, we can compare two execution order quantitatively regardless of variations in execution scenarios. \( E = 1 \) means perfect ordering (which may not be possible at all), and \( E = 0 \) means serialized execution.

2 MODEL-REPLICA DISTRIBUTION

Many large-scale machine learning systems in practice are distributed using a paradigm called Model-Replica [5–7]. In MR, a base model is replicated on several devices (also known as workers) and the input data is partitioned among these replicas. The goal of replicas is to collectively update a persistent collection of parameters. Therefore, at each step of training, MR requires synchronization among workers.

There are two major patterns for synchronization: (a) Parameter server: where one or more special devices called Parameter Servers (PS) store the master copy of the parameters and aggregate

\[\text{This metric is an update to "Scheduling Efficiency" metric in the original job shop problem}\]
updates to them([1, 4, 6, 11]). A worker loads a fresh copy of parameters at the beginning of each step and sends updates back to PS during the step. (b) All Reduce: where each worker keeps a copy of all parameters([2, 5, 7]). Changes are propagated directly to other workers with collective algorithms such as bucket algorithm [3] or recursive halving-doubling [15].

2.1 Ordering in Parameter Server

Search space for ordering in PS pattern can be reduced to the ordering of read ops on a worker. We observe that (i) Computation load on PS is insignificant compared to communication, which makes the ordering less influential on the overall performance, and (ii) read ops on workers are leaves in the model, that are immediately executable at the start of the step. Hence, the order of the computation ops does not impact the makespan.

We calculate the order of a read op as:

\[ \text{Order(read)} = \min \{|\text{Dep}(\text{op})| \text{ for op in G} \}
\]

if \( \text{read} \in \text{Dep}(\text{op}), |\text{Dep}(\text{op})| > 1 \) \hspace{1cm} \hspace{1cm} (3)

where \( G \) is the set of all the ops in the DAG, and \( \text{Dep}(\text{op}) \) is the set of read ops that \( \text{op} \) depends on. Intuitively this heuristic prioritizes read ops that are connected to computation ops with less dependencies. Evaluation is presented in §3.

2.2 Ordering in All Reduce

All Reduce requires all nodes to actively synchronize at the same time. As a result, in MR+AR, a perfect execution order should ensures that all workers reach the sync ops for each parameter simultaneously. Hence, we need to order both computation and communication ops.

Additionally, there are two windows for synchronizing each parameter: (a) at the beginning of a step before the parameter is modified or (b) at the end of a step after the parameter is updated. This adds further complexity to the ordering problem\(^5\). Ordering in MR+AR is ongoing work.

3 EVALUATION

We extend TensorFlow\(^4\) tracing capability to measure the cost of both computational and communication ops\(^5\) to calculate the “Ordering Efficiency”\(^6\). We implement the ordering algorithm as a static analyzer and make changes in TensorFlow core to enforce a given order on the sender side just before the response is sent to the destination.\(^6\). The experiments were run on a commodity cluster\(^7\) with one PS and one worker on separate nodes. We choose a batch size for each model in a way that balances communication and computation loads \(\left( \frac{\text{Cost}(G) - \text{Cost}(G_{\text{w/ Ordering}})}{\text{Cost}(G)} \right) > 90\% \). We tested our method on well-known models, Alexnet[10], ResNetV2-152[8], InceptionV2[14], VGG16[13] as well as two extreme toy models:

- Par-32: A flat model with 32 concurrent layers. All the topological orders are the best order in this model.
- Seq-32: A sequential model with 32 layers similar to recurrent models. Only one topological order out of 32! is the best order.

3.1 Results

Figure 2 shows the forward-pass step time cdf of 1-Worker 1-PS training on InceptionV2. We repeat 1000 runs each with and without the ordering heuristic. Enforcing ordering reduces the step time by 46% on average. Moreover, the standard deviation in step time is reduced by nearly 4x (48.3ms vs 187.1ms). We have observed similar pattern with other models except Par-32.

Figure 3 shows the average speedup of different models in forward-pass. Notably, Resnet-152 has 78% higher ordering efficiency (which results in 65% speed up in step time). Par-32, with all topological orderings already optimum, is the only model whose performance is hit marginally due to the ordering overhead.
REFERENCES


