Abstract

Distributed Machine Learning training is emerging in today’s datacenters. The training time of a distributed ML job is closely tied to the parallelism technique used to distribute the job across worker nodes, as well as the available compute, storage, and network capabilities in the datacenter. Current parallelism techniques often implicitly make two simplifying assumptions: (i) datacenter nodes are homogeneous and (ii) there is little to no congestion in the network. However, at scale, these assumptions do not always hold and stragglers are a pervasive challenge in today’s datacenter job scheduling. A straw-man approach to reduce the impact of stragglers is to replicate (sometimes speculatively) each sub-task. However, this approach commonly incorporates a timeout component to recover from straggler sub-tasks. In this work, we address the problem of how to curb the impact of stragglers in distributed ML jobs without having to speculate. Our approach leverages the unique nature of image recognition workloads and the theory of graph entropy. Our proposal, FlexEnt, identifies the entropy present in a training batch using a set of “feature filters” and replicates only a fraction of the training data on a predetermined set of “shadow workers”. In doing so, FlexEnt will decode the training parameters as soon as a sub-set of sub-tasks are completed. This approach will enable datacenter operators to adjust a knob between the number of additional shadow workers and the degree of stragglers protection.

1 Introduction

Today’s distributed Machine Learning (ML) workloads rely on either synchronous [1–3] or asynchronous updates between nodes [4–7]. A synchronous approach loads a fraction of the training data on to each node, then waits for the training algorithm to run, aggregating the results at each iteration using a parameter server or ring-reduce arrangement [3, 8–12]. In an asynchronous approach, the goal is to remove the dependency on scheduling and the behaviour of other machines to remove the effect of slow machines or congestion in the network [6, 12–15].

Synchronous approaches are broadly successful for small numbers of workers, as the synchronisation constraints are not overly limiting at small scale [7, 16]. However, the probability of a delay caused by a congested link or a slow worker grows exponentially as the system is scaled up [17]. This problem is exacerbated when a training task is run across heterogeneous nodes and workloads, since heterogeneity is a major source of stragglers [16, 18–20]. New approaches, such as Horovod [3], reduce the dependency of training on the network bandwidth available for sharing gradient updates between workers using the all-reduce algorithm’s ring structure [3]. However, by its nature this structure is vulnerable to stragglers, as a single slow machine or a congested link can limit the completion of the overall calculation [21].

Large-scale distributed ML workloads are expected to run in the heterogeneous cloud environments, hence many jobs of various sizes share the same infrastructure [13, 16, 20, 22]. Therefore, stragglers will be prevalent due to outdated hardware, failures, or congested links. Many systems use a hard deadline or a speculative approach to decide when to abandon the slow workers [18, 23]. To avoid using deadlines, one approach is to replicate the entire training task [23]. While this approach ensures there is always a redundant copy available of each worker, it also consumes a significant quantity of additional resources per level of protection obtained. In this work, we present FlexEnt, a coding-based scheduling system for training jobs. FlexEnt curbs the effect of stragglers in large-scale training jobs by selectively replicating high-entropy data as described in the next section.

2 FlexEnt System Description

The key to FlexEnt is to create sub-sampled sets of ‘shadow’ training data, formed from coding amalgams of the original training data. These high-entropy subsets can be trained on ‘shadow’ workers, and made available to the ‘primary’ workers performing the main training for a just-in-time replacement of gradient updates from stragglers. The number of shadow workers creating coded jobs is dependent on the desired level of straggler protection. The effectiveness of coding techniques for ML inference has been demonstrated in prior work [24]. However, a unique challenge of using any coding theory for training workloads is that the fundamental assumption of linear processing does not apply to ML training. We solve this challenge by ‘porting’ the required coding to the gradient aggregation step, which is a linear operation carried out on the results of a non-linear computation, and hence can be coded.

Our approach is designed to be applicable to commonly used stochastic ML techniques such as stochastic gradient descent [2, 8, 25]. FlexEnt’s goal is to build a reasonably accurate estimate of the output gradient vector from training on a quasi-redundant, but entropy rich, copy, based on the
entropy graph colouring of the input data. To do so, FlexEnt goes through the following steps:

**Setup Phase**
- Given \( n \) worker nodes, select \( s \) nodes as shadow workers and \( n - s \) nodes as primary workers;
- Assign a fraction of data (batch) to each primary worker.

**Sampling Phase**
- Calculate the joint entropy of each entry in the dataset under a set of features as explained in section 3;
- Select the entries that maximise the joint entropy and assign these to the shadow workers.

**Training Phase**
- Repeatedly run training iterations on both primary and shadow workers;
- Use shadow workers’ approximate gradients to replace a straggler’s gradients during update steps.

## 3 Entropy Graph Colouring

In information theory, an entropy function is built from a model function’s characteristic graph. The colouring of this graph represents the entropy available in the data [26, 27]. Figure 1 illustrates an example of two entropy graphs and their associated colouring. We take a dataset of five images to be processed in a distributed fashion. Our goal is to produce an entropy-rich sub-set of the images to be replicated across shadow workers. We demonstrate this process using two features appropriate to capture the highest entropies across the dataset. Images that contain a feature, shown by boxes in Figure 1, essentially have the same output in this dimension, and are therefore not connected in the graph. This allows us to draw and colour the entropy graph, as shown. We then sample the entropy graphs to obtain the maximum diversity across all colours, represented by the joint entropy shown in Figure 2, allowing a sampling rate that is below the straight sum of each dimension’s rate.

**Figure 1.** Intuitive examples of entropy graphs where emojis with different features are connected to each other. We then colour these graphs based on adjacency. To capture multiple dimensions, we repeat this process over many features, which produce different graphs such as (a) and (b). From information theory, illustrated in Fig. 2, we can use the joint entropy to identify the ‘entropy-rich’ samples.

Our preliminary results with the MNIST dataset confirm this theory. We find that when a primary worker is straggling behind, replacing its missing gradients with a shadow worker that trains on a randomly sub-sampled data achieves better accuracy compared to abandoning the primary worker. In future work, we plan to use the features during training iterations to select entropy rich data. Intuitively, gradient estimates produced from high-entropy samples should be significantly improved compared to random sampling, as they contain more information about the training space.

## 4 Conclusion

FlexEnt is a system designed to mitigate the impact of stragglers on distributed machine learning jobs in datacenters. We leverage concepts from coding and information theory to provide a good estimate at a predictable soft-threshold, rather than enforcing a hard deadline or using speculating approaches to recover from stragglers. Our approach is designed to provide an adjustable knob to specify the level of straggler protection required, so that the system provides the maximum speedup within constraints.

**Acknowledgments**

We would like to thank Ken Duffy and Seva Shneer for helpful discussions and guidance throughout this research.
References


