Scalable HOGWILD! for Big Models

Chengjie Qin
University of California Merced
cqin3@ucmerced.edu

Martin Torres
University of California Merced
mtorres58@ucmerced.edu

Florin Rusu
University of California Merced
frusu@ucmerced.edu

Abstract

Existing data analytics systems have approached predictive model training exclusively from a data-parallel perspective. Data examples are partitioned to multiple workers and training is executed concurrently over different partitions, under various synchronization policies that emphasize speedup or convergence. Since models with millions and even billions of features become increasingly common nowadays, model management becomes an equally important task for effective training. In this paper, we present a general framework for parallelizing stochastic optimization algorithms over massive models that cannot fit in memory. We extend the lock-free HOGWILD!-family of algorithms to disk-resident models by vertically partitioning the model offline and asynchronously updating the resulting partitions online. Unlike HOGWILD!, concurrent requests to the common model are minimized by a preemptive push-based sharing mechanism that reduces both the number of disk accesses as well as the cache coherency messages between workers. Extensive experimental results for three widespread analytics tasks on real and synthetic datasets show that the proposed framework achieves similar convergence to HOGWILD!, while being the only scalable solution to disk-resident models.

1 Big Model Problem

Due to the explosive growth in data acquisition, the current trend is to devise prediction models with an ever-increasing number of features, i.e., big models. For example, Google has reported models with billions of features for predicting ad click-through rates as early as 2013. Big models also appear in recommender systems. Spotify applies Low-rank Matrix Factorization (LMF) for 24 million users and 20 million songs, which leads to 4.4 billion features at a relatively small rank of 100.

Since HOGWILD! is an in-memory algorithm, it cannot handle these big models – models that go beyond the available memory of the system – directly. Parameter Server is an indirect approach that resorts to distributed shared memory. The big model is partitioned across several servers, with each server storing a sufficiently small model partition that fits in its local memory. Model updates are processed in two HOGWILD! stages. Each client copies a portion of the model over which it executes HOGWILD! on its training data—training data are partitioned across clients. The updated model is then pushed to the servers asynchronously, following the HOGWILD! paradigm. In addition to the complexity incurred by model partitioning and replication across servers, the limitations of Parameter Server are the expensive price in hardware and network traffic.

2 Scalable HOGWILD!

In this work, we investigate parallel stochastic optimization methods over big models that cannot fit in memory. Specifically, we focus on designing a scalable HOGWILD! algorithm. Our setting is a single multi-core server with attached storage (Figure 1). There is a worker thread associated with each core in the system. The training data as well as the model are stored on disk and moved into memory only when accessed. Training data are partitioned into partitions that are accessed and processed as a unit. Several partitions are processed concurrently by multiple worker threads—data-parallel processing. While access to the training data follows a well-behaved sequential pattern, the access to the model is unpredictable and random. Moreover, there are many model accesses for each training example—the number of non-zero entries in the example. In many cases, this number is several orders of magnitude larger than the number of examples. Thus, the challenge in handling big models is how to efficiently schedule access to the model. In the worst case, each model access may require a disk access. This condition is worsened in data-parallel processing by the fact that multiple model accesses are made concurrently by the worker threads—model-parallel processing.

We design a scalable model and data-parallel framework for parallelizing stochastic optimization algorithms over big models. At a high-level, our approach targets the main source that impacts performance – the massive number of concurrent model accesses – with two classical database processing techniques—*vertical partitioning* and *model access sharing*.

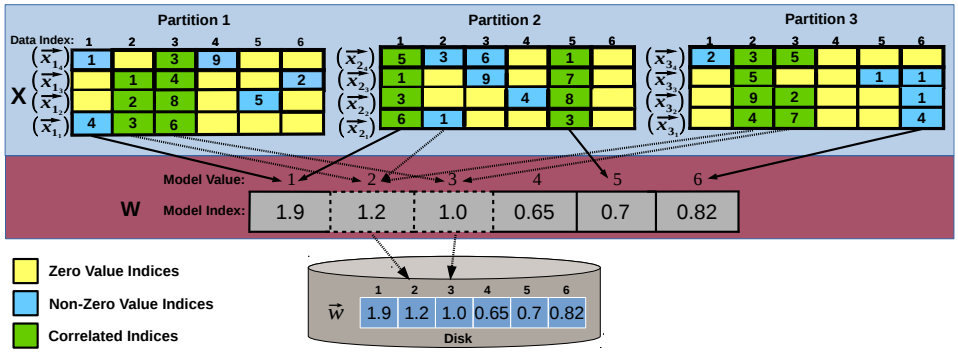

Figure 1: High-level approach of the scalable HOGWILD! framework for big models.

The model is vertically partitioned offline based on the concept of "feature occurrence" – we say a feature "occurs" when it has a non-zero value in a training example – such that features that co-occur together require a single model access. Feature co-occurrence is a common characteristic of big models in many analytics tasks. It is important to notice that feature co-occurrence is fundamentally different from the feature correlation that standard feature engineering processes try to eliminate. In feature engineering, correlation between features is measured by coefficients such as Pearson's coefficient instead of co-occurrence. In this work, we are interested exclusively in what features co-appear together.

During online training, access sharing is maximized at several stages in the processing hierarchy in order to reduce the number of disk-level model accesses. The data examples inside a chunk are logically partitioned according to the model partitions generated offline. The goal of this stage is to cluster together accesses to model features even across examples—vertical partitioning achieves this only for the features that co-occur in the same example. In order to guarantee that access sharing occurs across partitions, we introduce a novel push-based mechanism to enforce sharing by vertical traversals of the example data and partial dot-product materialization. Workers preemptively push the features they acquire to all the other threads asynchronously.

Our contributions can be summarized as follows. We design a scalable model and data-parallel framework for parallelizing stochastic optimization algorithms over big models. We formalize model partitioning as vertical partitioning and design a scalable frequency-based model vertical partitioning algorithm. We devise an asynchronous method to traverse vertically the training examples in all the data partitions according to the model partitions generated offline. We design a push-based model sharing mechanism for incremental gradient computation based on partial dot-products.