Database Parameter Server

Chengjie Qin
University of California, Merced
`cqin3@ucmerced.edu`

Florin Rusu
University of California, Merced
`frusu@ucmerced.edu`

Abstract

In this work, we propose *Database Parameter Server*, a database-centric solution to handle *big models*. The main idea is to offload the model to secondary storage and leverage database techniques for efficient model training. The model is represented as a table rather than as an array attribute. This distinction in model representation changes fundamentally how in-database analytics is carried out. By a thorough analysis of popular tasks used in practice, we identify *dot-product* as the most critical operator in gradient-based model training. Our main contribution is a parallel dot-product physical database operator optimized to execute secondary storage dot-products effectively. Compared to standard database join operators, the proposed dot-product operator is able to produce results in both blocking and non-blocking fashion, which is required for batch and online training, respectively. In this paper, we discuss the design decisions and several optimizations.

1 Big Models

The model size in Big Data analytics tasks has grown extensively over the years, due to the wide application of models whose size grows proportionally to the number of users, e.g., recommendation systems based on low-rank matrix factorization models. This big model trend has stimulated the creation of specialized training systems such as Parameter Sever which partition the model across the memory of an entire array of servers. These distributed memory solutions do not make adequate use of system resources, e.g., they do not use local secondary storage, and they incur a prohibitive cost—affordable only to large corporations. On the database front, the existing analytics solutions, e.g., MADlib, Bismarck, and GLADE, fail to support big models due to limitations in their original design—while the training data can be stored and processed from secondary storage, the model is always assumed to fit in memory. The solution we propose – Database Parameter Server – is an in-database analytics platform that can process both training data and models stored on secondary storage, e.g., disk and SSD.

2 Model Representation

Existing in-database analytics solutions use two tables to store the training data and model: `Data(dimension INTEGER[], value DOUBLE[], label INTEGER)` and `Model(model DOUBLE[])`, respectively. Table `Data` stores the value corresponding to each non-empty dimension in the feature vector and the label for every point in the dataset. The explicit representation of the dimensions is required since the data are sparse—there are only a few dimensions with non-zero values in each point. The model is stored in table `Model` as a single tuple with array type. However, `model` is a dense array with an entry corresponding to each dimension in the feature vector. This is the reason why the dimensions do not have to be represented explicitly anymore. While this solution works adequately for models with a relatively large number of features, e.g., thousands in PostgreSQL, it fails to scale to big models.

In this work, we propose to represent the model with a relational formalism. The `Model` table is represented as `Model(dimension INTEGER, value DOUBLE)`. In this representation, each dimension in the model ends-up being a tuple in the `Model` table—stored on secondary storage. Having this relational representation allows for arbitrary large models since databases are optimized for secondary storage processing. This change in model representation, however, disables existing User-Defined Function (UDF) and User-Defined Aggregate (UDA) computation methodology to perform gradient computation and model updates. This is because UDF and UDA can process only a single tuple at a time. They cannot work over a table as a whole. We propose novel methods to efficiently support these operations in the new model representation.

3 Dot-Product Operator

After carefully examining several popular models, we find that – regardless of what numerical method is used for training – the most data-intensive operation is the dot-product between a point in the training dataset and the model. For example, the gradient computation of a logistic regression model in Eq. (1) requires dot-products between the model $\vec{w}$ and every point $\vec{x}_i$ in the training dataset. $\vec{y}_i$ is the label associated with the point. When the model fits in memory, this quantity can be easily computed by calling a customized dot-product UDF which takes the point dimension and value arrays, and the model array as parameters, e.g., `dot-product(Data.dimension, Data.value, Model.model)`. However, this is impossible for the new representation in which the model is decomposed across several tuples processed independently.

$$\sum_i \frac{e^{-y_i \vec{w}\cdot\vec{x}_i}}{1+e^{-y_i \vec{w}\cdot\vec{x}_i}} (-y_i \cdot \vec{x}_i) \qquad (1)$$

We design and implement a novel parallel dot-product physical database operator targeted specifically at in-database big model analytics. The operator takes advantage of the fact that the training data are sparse, while the model is dense. It works efficiently with big models and is able to produce results in a non-blocking fashion, which allows for use in online learning algorithms.

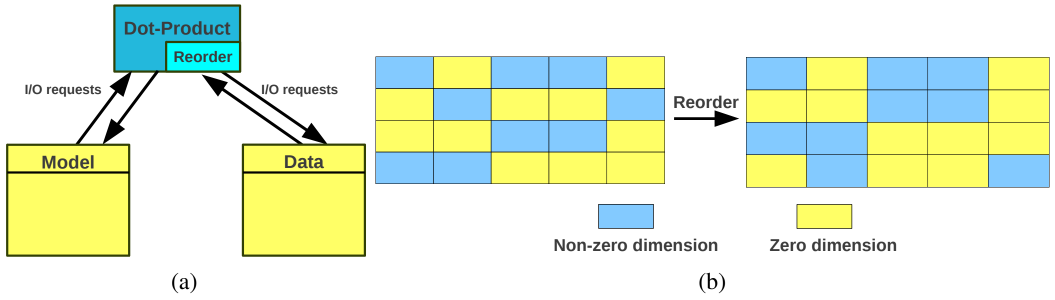

Figure 1: (a) Dot-Product operator. (b) Reorder I/O requests to `Data` table.

The architecture of the dot-product operator is depicted in Figure 1a. Every tuple in `Data` has to access the non-zero dimensions of the model. Since the model does not fit entirely in memory, this requires secondary storage access. The access pattern is determined entirely by the non-zero dimensions in each training point. If the points are processed in their original order, this has the potential to result in inefficient access to the model dimensions. The most important feature of the proposed dot-product operator is that it reorders the data points in batches to optimize the secondary storage accesses to the model. This process is depicted in Figure 1b. Since computing the optimal `Data` order is NP-complete, we use locality sensitive hashing (LSH) to achieve a good-enough order in a short amount of time.

Our main contributions can be summarized as follows. We propose a novel model representation for big model in-database analytics. We design and implement an in-database dot-product operator for the new model representation. We propose several optimizations to effectively reduce the I/O overhead over massive dot-products.