Lightning-Fast, Dirt-Cheap Parallel Stochastic Gradient Descent for Big Data in GLADE

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

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

Abstract

Stochastic gradient descent is a general technique to solve a large class of convex optimization problems arising in many machine learning tasks. GLADE is a parallel infrastructure for Big Data Analytics providing a generic task specification interface. We present a scalable and efficient parallel solution for stochastic gradient descent in GLADE. Empirical evidence confirms that our solution is limited only by the physical hardware characteristics, uses effectively the available resources, and achieves maximum scalability. As a concrete example, we are able to find the optimal low-rank factorization of a sparse 1 million X 1 million matrix with 10 billion non-empty cells in 80 seconds on a $30,000 9-node cluster.

1 Parallel stochastic gradient descent

Consider the following optimization problem with a linearly separable objective function: $\min_{w \in \mathbb{R}^d} \sum_{i=1}^{N} f(w, z_i)$ in which a d-dimensional vector $w \in \mathbb{R}^d$, $d \geq 1$ has to be found such that the objective function is minimized. The constants z_i, $1 \leq i \leq N$ correspond to tuples in a database table. Essentially, each term in the objective function corresponding to a tuple z_i can be viewed as a separate function $f_i(w) = f(w, z_i)$.

Gradient descent is an iterative method. The main idea is to start from an arbitrary vector $w^{(0)}$ and then to determine iteratively new vectors $w^{(k+1)}$ such that the objective function at each iteration decreases, i.e., $f(w^{(k+1)}) > f(w^{(k)})$. $w^{(k+1)}$ is determined by moving in the opposite direction to the gradient or subgradient of function f. Formally, this can be written as: $w^{(k+1)} = w^{(k)} - \alpha_k \nabla f_{\eta(k)} \left(w^{(k)}\right)$ where $\alpha_k \geq 0$ is the step size and $\nabla f_{\eta(k)}(w)$ is the approximation to the gradient $\nabla f(w)$ based on a single term $f_{\eta(k)}(w)$ at iteration k, respectively. Taking steps based on the tuple-based approximation instead of the actual gradient is the characteristic property of stochastic gradient descent (SGD). Typically, $\alpha_k \to 0$ as $k \to \infty$ and $\eta(k) \in \{1, \ldots, N\}$ represents a random permutation, i.e., all f_i have to be considered before any term is repeated. Convergence to a global optimal solution is theoretically guaranteed when $\sum_{i=1}^{N} f_i(w)$ is convex.

In parallel SGD, the tuples z_i, $1 \leq i \leq N$ are partitioned into multiple groups. A partial model – the vector w – is computed independently for each data partition. The optimal model is then obtained by merging the partial models together. Averaging is the standard method to merge models in parallel SGD computation. The components of the partial w vectors are pair-wise averaged, possibly weighted on the number of examples, in order to obtain the optimal w vector. An alternative is to average the gradients instead of the models and then to compute w based on the resulting gradient.

2 GLADE

GLADE is a parallel data processing system executing any computation specified as a Generalized Linear Aggregate (GLA) using a merge-oriented strategy supported by a push-based storage man-

ager that drives the execution. Essentially, GLADE provides an infrastructure abstraction for parallel processing that decouples the algorithm from the runtime execution. The algorithm has to be specified in terms of a clean interface, while the runtime takes care of all the execution details including data management, memory management, and scheduling. Figure 1 depicts the stages of the GLADE execution strategy expressed in terms of the GLA interface abstraction. The GLA interface extends the common UDAs implemented in every major RDBMS to parallel chunk-at-a-time or vectorized execution. Intuitively, GLAs are objects corresponding to the state of the aggregate upon which the methods in the GLA interface are invoked following a well-defined pattern.

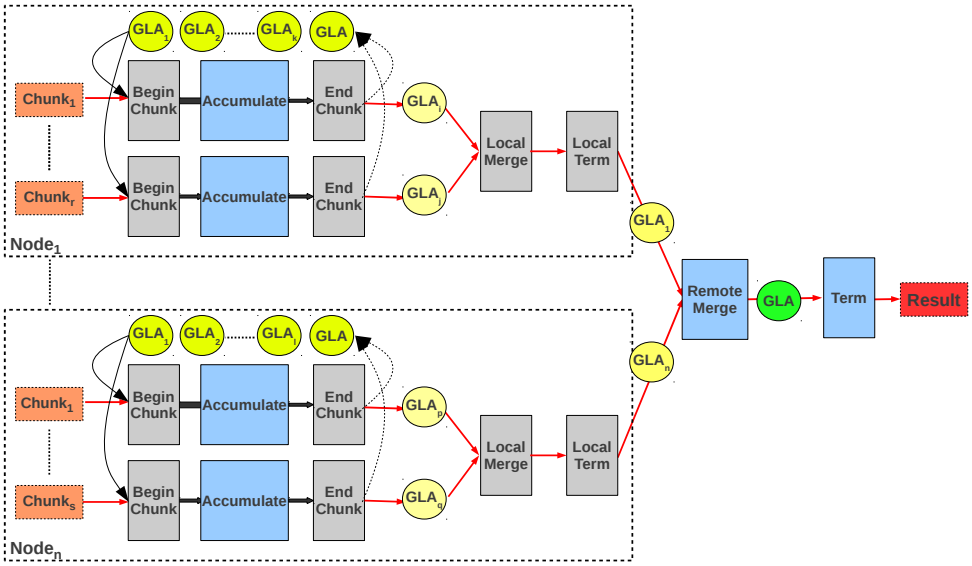

Figure 1: GLADE architecture: chunk-at-a-time merge-oriented parallel processing.

3 SGD in GLADE

Two GLAs are required to implement SGD in GLADE—one for computing the optimal model (`SGD GLA`) and one for computing the objective function (`Loss GLA`). All the action happens in `SGD GLA` where the optimal model is computed. The state of `SGD GLA` contains the model $w^{(k)}$, initialized either with the original starting point $w^{(0)}$ for the first iteration or with the model computed at the previous iteration $w^{(k-1)}$. For each tuple in the chunk, the approximation to the gradient $\nabla f(w)$ is computed and the model is updated accordingly in `Accumulate`. Since multiple chunks are processed in parallel, `Merge` is called after all the chunks are processed to combine together the partial models in the resulting GLAs. The two versions of `Merge` – local and remote – allow for different merging strategies. Once the complete model is computed, it is post-processed for the subsequent iteration in `Terminate`.

The order in which tuples are processed to update the model in `Accumulate` determines the SGD convergence rate. Typically, random orders not correlated with the tuple values provide better convergence. In GLADE, multiple levels of randomization are embedded in the execution strategy. Randomization across processing nodes is realized at data loading. It is a one time process that randomly partitions data across nodes. The order in which each data partition is traversed is non-deterministic from one iteration to another due to the intrinsic execution mechanism. Even more, the order is also quasi-random and impossible to determine prior to runtime. As a result, randomization is embedded implicitly in GLADE and it does not require special consideration.

Scalable I/O-Bound Parallel Incremental Gradient Descent for Big Data Analytics in GLADE by Qin and Rusu contains a more detailed presentation of the issues encountered when implementing SGD in GLADE. It also provides experimental results that support the claim made in the abstract. We argue that the SGD implementation in GLADE is the fastest available and it is the only scalable to the largest datasets without breaking the bank.