Fast Parameter Tuning with Approximations at Scale

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

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

Abstract. *Parameter tuning* is a fundamental problem that has to be handled by any Big Data analytics system. Identifying the optimal model parameters is an interactive, human-in-the-loop process that requires many hours – if not days and months – even for experienced data scientists. We argue that the incapacity to evaluate multiple parameter configurations simultaneously and the lack of support to quickly identify sub-optimal configurations are the principal causes.

In this paper, we develop two database-inspired techniques for efficient parameter tuning. *Speculative parameter testing* applies advanced parallel multi-query processing methods to evaluate several configurations concurrently and efficiently. *Online aggregation* is applied to identify sub-optimal configurations and halt corresponding configurations early in the processing.

We apply the proposed techniques to *distributed gradient descent optimization* – batch and stochastic – for support vector machines and logistic regression models. We evaluate their performance over terascale-size synthetic and real datasets. The results confirm that as many as 32 configurations can be evaluated concurrently almost as fast as one, while sub-optimal configurations are detected accurately in as little as a $1/20^{\text{th}}$ fraction of the time.

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: $w^{(k+1)} = w^{(k)} - \alpha_k \nabla f\left(w^{(k)}\right)$ where $\alpha_k \geq 0$ is the step size. When the gradient $\nabla f(w)$ is computed using all the tuples, the method is called batch gradient desecent (BGD). One variant of gradient descent is stochastic gradient descent (SGD) where $\nabla f(w)$ is approximated with the gradient of a single term. In SGD typically, $\alpha_k \to 0$ as $k \to \infty$ and the order of tuples has to be randomized for every iteration.

Speculative Iterations. We address two fundamental problems of gradient descent methods—convergence detection and parameter tuning. As with any iterative method, gradient descent convergence is achieved when there is no more decrease in the objective function, i.e., the loss, across consecutive iterations. While it is obvious that convergence detection requires loss evaluation at every iteration, the standard practice, e.g., Vowpal Wabbit, MLLib, is to discard detection altogether and execute the algorithm for a fixed number of iterations. The reason is simple: loss computation requires a complete pass over the data, which doubles the execution time. This approach suffers from at least two problems. First, it is impossible to detect convergence before the specified number of iterations finishes. And second, it is impossible to identify bad parameter configurations, i.e., configurations that do not lead to model convergence. Recall that both BGD and SGD depend on a series of parameters, the most important of which is the step size. Finding the optimal step size typically requires many trials. Discarding loss computation increases both the number of trials as well as the duration of each trial.

We propose a unified solution for convergence detection and parameter tuning based on speculative processing. The main idea is to *overlap gradient and loss computation for multiple parameter configurations across every data traversal*. This allows for timely convergence detection and early bad configuration identification since many trials are executed simultaneously. Overall, faster model training. The intuition behind our approach is that modern CPU architectures provide extensive parallelization opportunities, e.g., multi-core, hyper-threading, vectorization, that are difficult to use at full potential in disk-based workloads specific to Big Data analytics over massive datasets. With the right mix of tasks and judicious scheduling, the degree of parallelism supported in hardware can be fully utilized by executing multiple tasks concurrently. Moreover, the overall execution time is similar to the time it takes to execute each task separately.

Our contribution is to *design speculative gradient descent algorithms that test multiple step sizes simultaneously and overlap gradient and loss computation*. The number of step sizes used at each iteration is determined adaptively and dynamically at runtime. Only the model with the minimum loss survives each iteration, while the others are discarded (Figure 1). To the best of our knowledge, this is the first solution that uses speculative execution for gradient descent parameter tuning and loss computation.

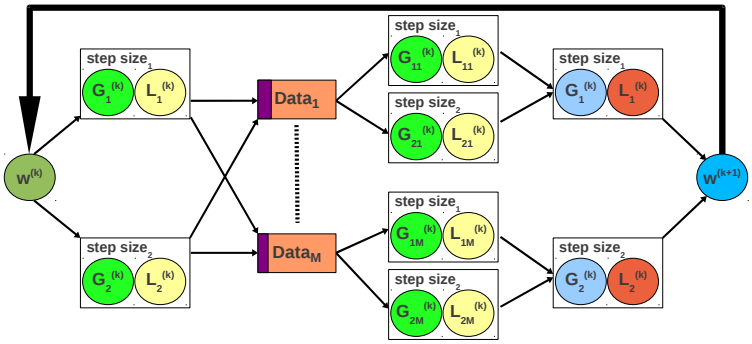

Figure 1: Speculative parameter testing with online aggregation.

Intra-Iteration Approximation. The speculative processing methods proposed in previous section allow for a more effective exploration of the parameter space. However, they still require complete passes over the entire data at each iteration in order to detect the sub-optimal parameter configurations. A complete pass is often not required in the case of massive datasets due to redundancy. It is quite likely that a small random sample summarizes the most representative characteristics of the dataset and allows for the identification of the sub-optimal configurations much earlier. This results in tremendous resource savings, more focused exploration, and faster convergence.

We present a *novel solution for using online aggregation sampling in parallel gradient descent optimization to speed-up the execution of a speculative iteration*. We generate samples with progressively larger sizes dynamically at runtime and execute gradient descent optimization incrementally, until the approximation error drops below a user-defined threshold ϵ. Relative to Figure 1, the entire process is executed multiple times during an iteration, on samples with increasing size. This is completely different from static sub-sampling methods that first extract a fixed-size random sample and then execute gradient descent on the sample, expecting that the optimal solution over the sample is also optimal for the entire dataset. Moreover, online aggregation avoids the expensive re-sampling required in sub-sampling whenever the approximation error level is not satisfied. While parallel online aggregation has been studied before for SQL aggregates, we are the first to consider online aggregation for complex analytics problems such as parallel gradient descent optimization.

Our main contributions can be summarized as follows. We formalize gradient descent optimization as aggregate estimation. We provide efficient parallel solutions for the evaluation of the speculative estimators and of their corresponding confidence bounds that guarantee fast convergence. We design halting mechanisms that allow for the speculative query execution to be stopped as early as the user-defined accuracy threshold ϵ is achieved.