SLAQ: Quality-Driven Scheduling for Distributed Machine Learning[*]

Haoyu Zhang, Logan Stafman, Andrew Or, Michael J. Freedman
Princeton University

1 Background and Motivation

Machine learning (ML) is an increasingly important tool for large-scale data analytics. A key challenge in analyzing massive amounts of data with ML arises from the fact that model complexity and data volume are growing much faster than hardware speed improvements. Thus, time-sensitive ML on large datasets necessitates the use and efficient management of cluster resources. Three key features of ML are particularly relevant to resource management.

ML algorithms are intrinsically approximate. ML models are approximate functions for input-output mapping. We use *quality* to measure how well the model maps input to the correct output. Training ML models is a process of optimizing the model parameters to maximize the quality on a dataset.

ML training is typically iterative with diminishing returns. Algorithms such as Gradient Descent, L-BFGS and Expectation Maximization (EM) are widely used to *iteratively* solve the numerical optimization problem. The quality improvement *diminishes* as more iterations are completed (Figure 1).

Training ML is an exploratory process. ML practitioners retrain their models repeatedly to explore feature validity [2], tune hyperparameters [3, 4, 5, 6], and adjust model structures [7], in order to operationalize the final model with the best quality. Practitioners in experimental environments often prefer to work with more approximate models (e.g., 95% loss reduction) trained within a short period of time for preliminary testing, rather than wait a significant amount of time for a perfectly converged model with poorly tuned configurations.

Existing schedulers primarily focus on *resource fairness* [8, 9, 10, 11, 12, 13], but are agnostic to model quality and resource efficiency. With this policy, equal resources will be allocated to jobs that are in their early stages and could benefit significantly from extra resources as those that have nearly converged and cannot improve much further. This is not efficient. The key intuition behind our system is that *in the context of approximate ML training, more resources should be allocated to jobs that have the most potential for quality improvement*.

2 Design

We present SLAQ, a cluster scheduling system for ML training jobs that aims to maximize the overall job quality. To achieve this, SLAQ needs to (1) normalize the quality metrics in order to trades off resources and quality across multiple jobs; (2) predict how much progress the job would achieve if it was granted a certain amount of resources; (3) efficiently allocate cluster CPUs to maximize the system-wide quality improvement.

Normalizing Quality Metrics. While metrics like accuracy and F1 score [14] are intuitively understandable, they are not applicable to non-classification algorithms. In contrast, loss functions are internally calculated by almost all algorithms in each iteration, but each loss function has a different real-world interpretation, and its range, convexity, and monotonicity of depend on both the models and the optimization algorithms. Directly normalizing loss values requires a priori knowledge of the loss range, which is impractical in an online setting.

We choose to normalize the *change* in loss values between iterations, with respect to the largest change we have seen so far. Figure 2 shows the normalized changes of loss values for common ML algorithms. Even though the algorithms have diverse loss ranges, we observe that the changes generally follow similar convergence properties, and can be normalized to decrease from 1 to 0. This helps SLAQ track and compare the progress of different jobs, and, for each job, correctly project the time to reach a certain loss reduction with a given resource allocation. Note that this approach currently does *not* support some non-convex algorithms (such as training Deep Neural Networks) due to the lack of convergence of these analytical models.

Predicting Quality Improvement. Previous work [15, 16] estimates general-purpose big-data job runtime by analyzing the job computation and communication structure, using offline analysis or code profiling. As the computation and communication pattern changes during ML model configuration tuning, the process of offline analysis needs to be performed every time, thus incurring significant overhead.

We use *online* quality prediction by leveraging the convergence properties of the loss functions. Based on the optimizers used for minimizing the loss function, we can broadly categorize the algorithms by their convergence rate.

I. *Algorithms with sublinear convergence rate.* First-order algorithms[1] (e.g., gradient descent) have a convergence rate of $O(1/k)$, where k is the number of iterations [17]. The convergence rate could be improved to $O(1/k^2)$ with optimization.

II. *Algorithms with linear or superlinear convergence rates.* Algorithms in this category[2] have a convergence rate of $O(\mu^k), |\mu| < 1$. For example, L-BFGS, which is a widely used Quasi-Newton Method, has a superlinear convergence rate which is between linear and quadratic.

With the assumptions of loss convergence rate, we use exponentially weighted history loss values at to fit a curve $f(k) = \frac{1}{ak^2+bk+c} + d$ for sublinear algorithms, or $f(k) = \mu^{k-b} + c$ for linear and superlinear algorithms. Intuitively, loss values obtained in the near past are more informative for predicting the loss values in the near future. Experiments show that this prediction achieves less than 5% prediction errors for all the algorithms in Figure 2 when predicting the next 10th iteration.

[*]This work has been previously published in ACM SoCC '17 [1].

[1]Assume f is convex, differentiable, and ∇f is Lipschitz continuous.

[2]Assume f is convex and twice continuously differentiable, optimizers can take advantage of the second-order derivative to get faster convergence.

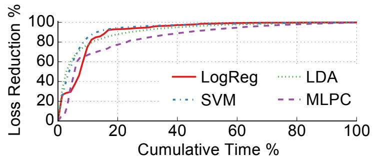
Figure 1: $> 80\%$ of work is done in $< 20\%$ of time.

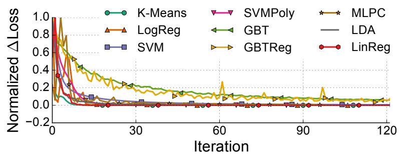
Figure 2: Normalized ΔLoss for ML algorithms.

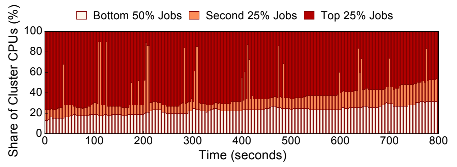
Figure 3: Resource allocation across job groups.

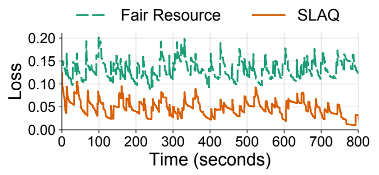
Figure 4: Average of normalized loss values.

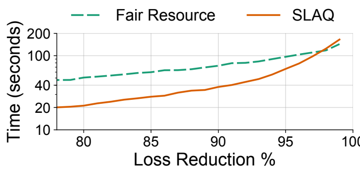
Figure 5: Time to achieve loss reduction percentage.

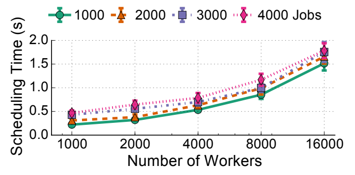
Figure 6: Scheduling time.

Scheduling Based on Quality Improvements. We schedule a set of J jobs running concurrently on the shared cluster for a fixed scheduling epoch T. The optimization problem for maximizing the total normalized loss reduction over a short time horizon T is as follows. Sum of allocated resources a_j cannot exceed the cluster resource capacity C.

$$\begin{aligned} \max_{j \in J} \quad & \sum_j Loss_j(a_j, t) - Loss_j(a_j, t+T) \\ s.t. \quad & \sum_j a_j \leq C \end{aligned}$$

The algorithm starts with $a_j = 1$ for each job to prevent starvation. At each step we consider increasing a_i (for all jobs i) by one unit (i.e., one CPU core) and get the predicted loss reduction. Among these jobs, we pick the job j that gives the highest loss reduction, and increase a_j by one unit. We repeat this until we run out of available resources.

3 Evaluation

Setup. We implemented SLAQ within the Apache Spark framework [18] and utilize its accompanying MLlib machine learning library [19]. Our testbed consists of a cluster of 20 c3.8xlarge EC2 instances on the AWS Cloud. We tested SLAQ with the most common ML algorithms, including (i) classification: SVM, Neural Network (MLPC), Logistic Regression, GBT, and our extension to Spark MLlib with SVM polynomial kernels; (ii) regression: Linear/GBT Regression; (iii) unsupervised learning: K-Means, LDA. Each algorithm is further diversified to construct different models. We collected more than 200GB datasets from various online sources, spanning numerical, plain texts [20], images [21], audio meta features [22], and so on [23]. The baseline we compare against is a work-conserving fair scheduler, which is the widely-used scheduling policy for cluster computing frameworks [8, 9, 10, 12, 13].

Scheduler Quality and Runtime Improvement. We submit a set of 160 ML training jobs with different models, following a Poisson distribution (mean arrival time 15s). Figure 4 shows the average normalized loss values across running jobs in an 800s time window of the experiment. When a new job arrives, its initial loss is 1.0, raising the average loss value; the spikes indicate new job arrivals. The average loss value achieved by SLAQ is on average 73% lower than that of the fair scheduler.

Figure 5 shows the average time it takes a job to achieve different loss values. As SLAQ allocates more resources to jobs that have the most potential for quality improvement, it reduces the average time to reach 90% (95%) loss reduction from 71s (98s) down to 39s (68s), 45% (30%) faster. For exploratory training, this level of accuracy is frequently sufficient. Thus, in an environment where users submit exploratory ML training jobs, SLAQ could substantially reduce users' wait times.

Figure 3 explains SLAQ's benefits by plotting the allocation of CPU cores in the cluster over time. Here we group the active jobs by their normalized loss: (i) 25% jobs with high loss values; (ii) 25% jobs with medium loss values; (iii) 50% jobs with low loss values (almost converged). With a fair scheduler, the cluster CPUs should be allocated to the three groups proportionally to the number of jobs. In contrast, SLAQ allocates much more resource to (i) (60%) than to (iii) (22%), which is the underlying reason for the improvement in Figures 4 and 5.

Scalability and efficiency. SLAQ is a *fine-grained* job-level scheduler: it allocates resources between competing ML *jobs*, but does so over *short time intervals* to ensure the continued rebalancing of resources across jobs. Figure 6 plots the time to schedule tens of thousands of concurrent jobs on large clusters (simulating both the jobs and worker nodes). SLAQ makes its scheduling decisions in hundreds of milliseconds to a few seconds, even when scheduling 4,000 jobs across 16K worker cores. SLAQ is sufficiently fast and scalable for (rather aggressive) real-world needs.

4 Conclusion and Future Work

SLAQ is a quality-driven scheduling system designed for large-scale ML training jobs in shared clusters. SLAQ leverages the iterative nature of ML algorithms and obtains highly-tailored prediction to maximize the quality of models produced by a large class of ML training jobs. As a result, SLAQ improves the overall quality of executing ML jobs faster, particularly under resource contention.

Non-convex optimization. Loss functions of non-convex optimization are not guaranteed to converge to global minima, nor do they necessarily decrease monotonically. The lack of an analytical model of the convergence properties interferes with our prediction mechanism, causing SLAQ to underestimate or overestimate the potential loss reduction. One potential solution is to let users provide the scheduler with hint of their target loss or performance, which could be acquired from state-of-the-art results on similar problems or previous training trials. The convergence properties of non-convex algorithms is being actively studied in the ML research community [24, 25]. We leave modeling the convergence of these algorithms to future work, and an interesting topic for future discussion at SysML.

References

[1] H. Zhang, L. Stafman, A. Or, and M. J. Freedman. SLAQ: Quality-Driven Scheduling for Distributed Machine Learning. In *ACM SoCC*, 2017.

[2] M. Anderson, D. Antenucci, V. Bittorf, M. Burgess, M. Cafarella, A. Kumar, F. Niu, Y. Park, C. Ré, and C. Zhang. Brainwash: A Data System for Feature Engineering. In *CIDR*, 2013.

[3] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS*, 2012.

[4] D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based Hyperparameter Optimization through Reversible Learning. 2015.

[5] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits. *ArXiv*, abs/1603.06560.

[6] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population Based Training of Neural Networks. *ArXiv*, abs/1711.09846.

[7] S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *ArXiv*, abs/1510.00149, 2015.

[8] Apache Hadoop YARN. Retrieved 02/08/2017, URL: `http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html`.

[9] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *USENIX NSDI*, 2011.

[10] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *USENIX NSDI*, 2011.

[11] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica. Hierarchical Scheduling for Diverse Datacenter Workloads. In *ACM SoCC*, 2013.

[12] Capacity Scheduler. Retrieved 04/20/2017, URL: `https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html`.

[13] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair Scheduling for Distributed Computing Clusters. In *ACM SOSP*, 2009.

[14] D. Powers. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[15] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *USENIX NSDI*, 2016.

[16] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *USENIX NSDI*, 2017.

[17] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2nd edition, 2009.

[18] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *USENIX NSDI*, 2012.

[19] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: Machine Learning in Apache Spark. *ArXiv*, abs/1505.06807, 2015.

[20] Associated Press Dataset - LDA. Retrieved 04/20/2017, URL: `http://www.cs.columbia.edu/~blei/lda-c/`.

[21] MNIST Database. Retrieved 04/20/2017, URL: `http://yann.lecun.com/exdb/mnist/`.

[22] Million Song Dataset. Retrieved 04/20/2017, URL: `https://labrosa.ee.columbia.edu/millionsong/`.

[23] LibSVM Data. Retrieved 04/20/2017, URL: `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

[24] N. Boumal, P.-A. Absil, and C. Cartis. Global Rates of Convergence for Nonconvex Optimization on Manifolds. *ArXiv e-prints*, May 2016.

[25] S. Lacoste-Julien. Convergence Rate of Frank-Wolfe for Non-Convex Objectives. *ArXiv e-prints*, July 2016.