Distributed Delayed Proximal Gradient Methods

Mu Li[1], **David G. Andersen**[1] **and Alexander Smola**[1,2]
[1]Carnegie Mellon University
[2]Google Strategic Technologies

Abstract

We analyze distributed optimization algorithms where parts of data and variables are distributed over several machines and synchronization occurs asynchronously. We prove convergence for the general case of a nonconvex objective plus a convex and possibly nonsmooth penalty. We demonstrate two challenging applications, ℓ_1-regularized logistic regression and reconstruction ICA, and present experiments on real datasets with billions of variables using both CPUs and GPUs.

1 Introduction

Distributed optimization and inference has become a staple to large scale machine learning methods. It is a necessity since the amounts of data, both in terms of parameters and observations are so large that no single machine can solve the problems in sufficiently short time.

We propose a unified strategy for solving nonconvex optimization problems of the form

$$\text{minimize } F(x) := f(x) + h(x) \text{ for } x \in \mathcal{X}. \tag{1}$$

Here $f : \mathbb{R}^p \to \mathbb{R}$ is continuously differentiable but not necessary convex and $h : \mathbb{R}^p \to \mathbb{R} \cup \{\infty\}$ is LSC, convex, block separable, but possibly nonconvex. The proposed algorithm, which derives from proximal gradient methods, is fairly simple. However, making this algorithm efficient on datasets with billions of parameters and observations poses a number of challenges.

The memory footprint for storing parameters is enormous. For instance, storing 10 billion variables in double pfecision requires 80G space. Furthermore, the number of observations is typically orders of magnitude larger. Hundreds billions of observations are not unusual today[8]. We propose two levels of partitioning to alleviate this problem. First, only a block of parameters is updated in each iteration. This way only the corresponding subset of parameters and associated data need to be accessed in memory. Furthermore, the data are distributed over several machines.

The cost of synchronization across machines is considerable, due to limited network bandwidth and the fact that machine responses may be quite nonuniform. Waiting for the slowest machine can affect efficiency significantly [20, 3]. We explore asynchronous synchronizations to hide the communication cost. Different to the bulk synchronous parallel (BSP) model, we calculate inexact gradients based on out-of-date shared variables. Recently [19] showed convergence results for the nonconvex minimization problem (1) by assuming the approximation error of inexact gradients is bounded. We remove this requirement and extend the convergence analysis to the two-level partitions.

Moreover, we demonstrate the efficiency of this approach experimentally in two challenging applications. One is non-smooth ℓ_1-regularized logistic regression on sparse text datasets with billions of features, the other is a non-convex and non-smooth ICA reconstruction problem [15] which extracts billions of sparse features from dense image data. Promising results are presented by utilizing both CPUs and GPUs. To our best knowledge, this is the first public attempt to scale these two applications to billions of variables. The most closely related work is [10] who infer parameters of a large scale deep belief network using BSP.

2 Related Work

Asynchronous algorithms have recently received significant interests, largely due to their efficient usage of both CPU and network bandwidth. For instance, Shotgun [7] and Hogwild [18] perform parallel asynchronous descent in shared memory processes. Other methods partition observations over several machines and compute statistics of these subblocks to make progress in a data parallel fashion [22, 14, 27, 3, 1, 13, 23]. Lastly, the NIPS framework [19] discusses general non-convex approximate proximal methods.

Our work differs primarily in two aspects: First we focus on solving problems with billions of observations and parameters. Hence the shared memory approach explored by Shotgun and Hogwild does not apply. Secondly, we aim to solve general non-convex and non-smooth composite objective functions. A similar problem was examined by [19], however, we derive a convergence theorem with weaker assumptions and furthermore we carry experiments on orders of magnitude larger scale.

3 Distributed Delayed Proximal Optimization

3.1 Proximal Gradient Methods and Gauss-Seidel Iterations

We give a brief overview of proximal gradient methods below for the sake of being self-contained. Much more detailed surveys can be found in [9, 16]. Given a closed proper convex function $h(x) : \mathbb{R}^p \to \mathbb{R} \cup \{+\infty\}$, we define the generalized proximal operator as

$$\text{Prox}_\gamma^U(x) = \operatorname*{argmin}_{y \in \mathcal{X}} \; h(y) + \frac{1}{2\gamma} \|x - y\|_U^2 . \tag{2}$$

Here $\|x\|_U^2 := \langle x, Ux \rangle$ is the Mahalanobis norm with respect to a positive semi-definite matrix $U \succ 0$. Typically proximal algorithms choose $U = \mathbf{1}$. However, we will take advantage of nontrivial block structure in our analysis.

To minimize the composite objective function $f(x) + h(x)$, proximal gradient methods update x in two steps: a forward step performing steepest gradient descent on f and a backward step, carrying out projection using h. For a given learning rate $\gamma_t > 0$ at iteration t these two steps can be written as

$$x(t+1) = \text{Prox}_{\gamma_t}^{U(t)} \left[x(t) - \gamma_t \nabla f(x(t)) \right] \quad \text{for } t \in \mathbb{N} \tag{3}$$

As stated in the introduction, touching all data at once is costly. This can be addressed by two complementary strategies: online learning algorithms use only a group of observations each time [5]; alternatively we may update only a block of parameters at a time. This has been used in Gauss-Seidel algorithms [4] and block coordinate descent [24].

In the context of proximal algorithms we define Gauss-Seidel iterations as follows: assume that the coordinates $\{1, \ldots, p\}$ are divided into B disjoint blocks. At each iteration some block b will be updated. Denote by $x_b \in \mathbb{R}^p$ the vector by setting the entries of x which are not in block b to 0. Moreover, denote by $x_{-b} = x - x_b$ its complement. Finally, we denote by $U(t) \in \mathbb{R}^{p \times p}$ the scaling matrix and $s(t) \in \mathbb{R}^p$ the (inexact) gradients, which have non-zeros entries only in block b and will be formally defined later. Then we have the following updating rules:

$$\delta(t) = \text{Prox}_{\gamma_t}^{U(t)} \left[x_b(t) - \gamma_t U(t)^{-1} s(t) \right] - x_b(t) \quad \text{and} \quad x(t+1) = x(t) + \delta(t). \tag{4}$$

Since h is block separable, the proximal operator is separable too. Consequently $\delta_b(t) = \delta(t)$ and only parameters in block b are changed.

3.2 Distributed Implementation by the Parameter Server Framework

Even though only a portion of data will be touched in each iteration, it may be still too costly to fit into a single machine. We use the parameter server framework [2, 10, 13] to describe the distributed implementation. A parameter server consists of a distributed server array and several clients. The server array maintains globally shared variables by a group of machines. Each client has a partition of observations and parameters and communicate with the servers to synchronize the shared variables.

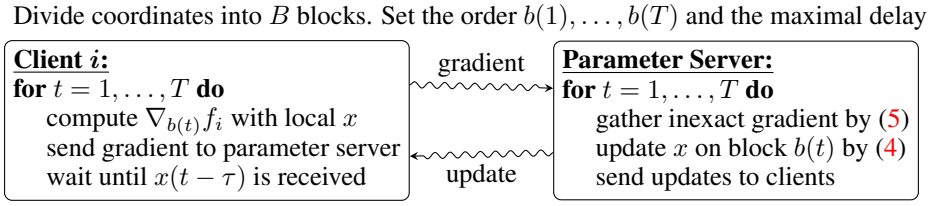

Figure 1: D2P, Distributed Delayed Proximal Gradient Methods. Both clients and the parameter server span several machines. All data sending and receiving are non-blocking.

We give a brief example when x is the shared variable, which is used by ℓ_1-regularized logistic regression. Given m clients, we rewrite f and its gradients on block b into a summation of m terms: $f(x) = \sum_{i=1}^m f_i(x)$ and $\nabla_b f(x) = \sum_{i=1}^m \nabla_b f_i(x)$. Then let client i maintain the observations and parameters needed to compute $f_i(x)$ and its gradients. On every iteration, all clients compute the local gradient $\nabla_b f_i(x)$ on this block simultaneously and then send it to the server, namely several machines in the server array. Subsequently they expect to receive the according updates for x. Unfortunately such a (synchronous) design is highly sensitive to delays of individual machines.

3.3 Asynchronous with Delayed Parameters

As discussed earlier, the BSP model ensures that all clients have the newest shared variables at the beginning of each iteration. This may be not efficient. We relax this constraint by allowing clients to start computing the gradients immediately by using an out-of-date version of the shared variable. More precisely, assume client i only has $x(t - \tau(t, i))$ with delay $\tau(t, i) \geq 0$ at iteration t. It means that the local copy of x at client i only contains updates of x, i.e. δ in (4), until iteration $t - \tau(t, i)$. More recent updates are not received by client i yet. Then the inexact gradients are computed by

$$s(t) = \sum_{i=1}^{m} \nabla_b f_i(x(t - \tau(t, i))). \tag{5}$$

If the allowed delay is sufficient large, all clients keep on computing gradients without waiting the updates from the server. Usually, the changes of $x(t)$ will be small when approaching a stationary point. Hence $x(t - \tau(t, i))$ will be similar to $x(t)$ and therefore $s(t)$ is a good approximation to the exact gradient. We name this algorithm D2P, and show the overview in Figure 1.

3.4 Convergence Analysis

To obtain convergence guarantees we need to assume that the rate of change in the gradients of f is bounded, when changing blocks of parameters.

Assumption 1 (Block Lipschitz Continuity) *There exists positive constants L_i such that for any block b and $x, y \in \mathcal{X}$ with $x_{-b} = y_{-b}$, $\|\nabla_{b'} f_i(x) - \nabla_{b'} f_i(y)\| \leq L_i \|x - y\|$ holds for all i and b'.*

We define $L = \sum_{i=1}^m L_i$ as the bound on the aggregate changes. We obtain that if the delay $\tau(t)$ is bounded then the algorithm will converge with a suitable learning rate.

Theorem 2 *Assume Assumption 1 and h is block separable with $0 \in \partial h(x)$. Let τ be the maximal delay, namely $\tau(t, i) \leq \tau$ for any t and i. Choose $U(t)$ such that $M_t I \preceq U(t)$. Let $\epsilon > 0$, then D2P converges to a stationary point if the learning rate γ_t satisfies $\gamma_t \leq \frac{M_t}{(1+\tau)L+\epsilon}$ for all $t \geq 0$.*

Note that our results do not assume convexity, hence the convergence results may be modest. Since many problems in machine learning are nonconvex, the need for such guarantees is substantial.

4 Experiments

We now show how the general framework discussed above can be used to solve two rather challenging problems — ℓ_1-regularized logistic regression (ℓ_1-LR) with sparse training data:

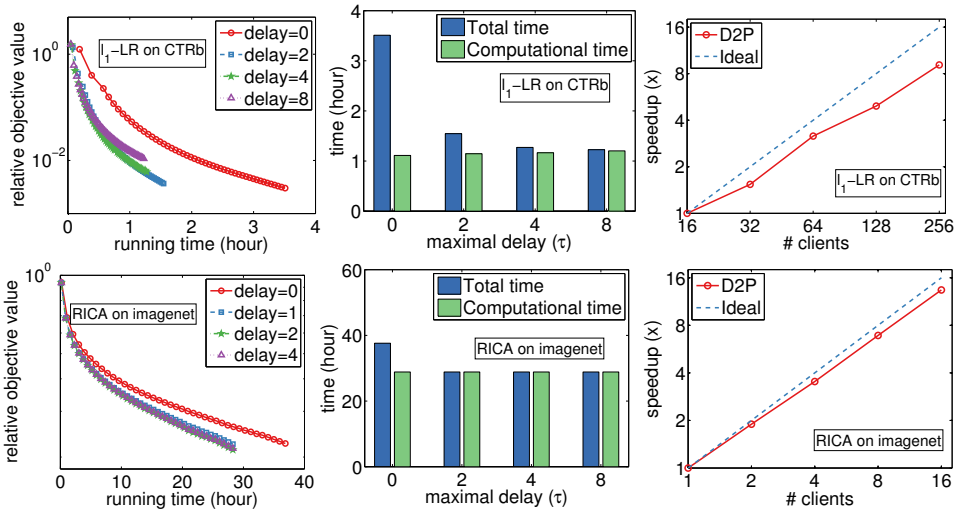

Figure 2: Left: varying delays. Allowing for modest delays yields considerable speedup due to improved resource usage. Middle: decomposition of running times. Delay significantly reduces the communication cost. Right: Scalability. Note the essentially linear dependence.

$\min_{x\in\mathbb{R}^p} \sum_{i=1}^n \log(1+\exp(-l_i \langle d_i, x\rangle)) + \lambda\|x\|_1$ and reconstruction ICA (RICA) with dense training data: $\min_{X\in\mathbb{R}^{\ell\times p}} \sum_{i=1}^n \frac{1}{2}\left\|XX^\top d_i - d_i\right\|_2^2 + \lambda\left\|Xd_i\right\|_1$. While the former is convex, the latter is highly nonconvex. The generalized proximal operator has close-form solution for ℓ_1-LR, i.e. soft-shrinkage operator. It is not true for RICA. We solve it by ADMM[6] instead. We use diagonal scaling matrices U for both of them: an upper bound of the diagonal Hession matrix[26] for ℓ_1-LR, and an aggregation of past gradients[10] for RICA. We select learning rate $\gamma \in [10, 0.1]$[1].

We run ℓ_1-LR and RICA on clusters with 256 AMD CPU cores and 16 Tesla K20 graphical cards, respectively. A range of sparse and dense datasets have been evaluated. CTRb and imagenet are the largest two. The former is a sparse Ads click-through dataset from an anonymous search company. It contains 0.34B observations, 2.2B unique features, and 31B nonzero entries. The latter is downloaded from image-net.org with 100K images resized into 100x100 pixels.

Due to the space constraint, we only report results on these two datasets. The number of blocks is fixed by $B = 128$, to which the results are insensitive. For RICA we use 1 Billion parameters $X \in \mathbb{R}^{10^6\times 10^4}$. The results are shown in Figure 2. It is evident that asynchronous updating ($\tau > 0$) offer a clear improvement over synchronous optimization ($\tau = 0$). The improvement mainly comes from hiding the communication cost. A nine-fold speedup is observed when increasing the clients by 16 times for ℓ_1-LR, while we see a 13.5 fold acceleration for RICA.

5 Conclusion

Multicore and multi-machine inference is here to stay due to increasing amounts of data and an ongoing paradigm shift to more and relatively smaller processors for data analysis. These two changes imply that delays in updates due to network constraints, the sheer size of the parameter space, and natural variability in systems of computers, are quite fundamental effects that need addressing.

The current paper adds to the body of work outlining asynchronous computation strategies for machine learning. Unlike much systems work which attempts to guarantee perfect serializability, that is, correctness with regard to a sequential execution plan, we embrace asynchrony and design algorithms that are capable of obtaining valid results nonetheless. This approach is somewhat more laborious for the machine learner. However, We believe it to be a more rewarding strategy since it liberates the systems designer from having to provide execution guarantees that can be very expensive to deliver. In other words, Algorithms such as D2P work even in the adversity of delays.

[1]The C++ codes are available at http://parameterserver.org

References

[1] A. Agarwal and J. C. Duchi. Distributed delayed stochastic optimization. In *IEEE Conference on Decision and Control*, 5451–5452. 2012.

[2] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A.J. Smola. Scalable inference in latent variable models. In *WSDM*, 2012.

[3] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *WWW*, 2013.

[4] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.

[5] L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–123, 2010.

[7] J.K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for L1-regularized loss minimization. In *ICML*, 2011.

[8] K. Canini. Sibyl: A system for large scale supervised machine learning. *Technical Talk*, 2012.

[9] P. L. Combettes and J. C. Pesquet. Proximal splitting methods in signal processing. In *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, Springer, 2011.

[10] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng. Large scale distributed deep networks. In *NIPS*, 2012.

[11] R.-E. Fan, J.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. *SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[13] Q. Ho, J. Cipar, H. Cui, S. Lee, J. Kim, P. Gibbons, G. Gibson, G. Ganger, and E. Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *NIPS*, 2013.

[14] J. Langford, A. J. Smola, and M. Zinkevich. Slow learners are fast. In *NIPS*, 2009.

[15] Q.V. Le, A. Karpenko, J. Ngiam, and A.Y. Ng. ICA with reconstruction cost for efficient overcomplete feature learning. In *NIPS*, 2011.

[16] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 2013.

[17] K. B. Petersen and M. S. Pedersen. The matrix cookbook, 2008.

[18] B. Recht, C. Re, S.J. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS 24*, 2011.

[19] S. Sra. Scalable nonconvex inexact proximal splitting. In *NIPS*, 2012.

[20] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, 2011.

[21] C. Teflioudi, F. Makari, and R. Gemulla. Distributed matrix completion. In *ICDM*, 2012.

[22] C. H. Teo, S. V. N. Vishwanthan, A. J. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *JMLR*, 11:311–365, 2010.

[23] Martin Takac and Peter Richtarik. Distributed coordinate descent method for learning with big data. Technical report, 2013.

[24] P. Tseng and S. Yun. A block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. *Journal of Optimization Theory and Applications*, 2009.

[25] G. X. Yuan, K. W. Chang, C. J. Hsieh, and C. J. Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *JMLR*, pages 3183–3234, 2010.

[26] T. Zhang and F. J. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4:5–31, 2001.

[27] M. Zinkevich, A. J. Smola, M. Weimer, and L. Li. Parallelized stochastic gradient descent. In *NIPS*, 2010.