Asynchronous Complex Analytics in a Distributed Dataflow Architecture (Extended Abstract)

Joseph E. Gonzalez, Peter Bailis†, Michael I. Jordan, Michael J. Franklin,
Joseph M. Hellerstein, Ali Ghodsi, Ion Stoica
UC Berkeley and †Stanford University

Abstract

Scalable distributed dataflow systems have recently experienced widespread adoption, with commodity dataflow engines such as Hadoop and Spark, and even commodity SQL engines routinely supporting increasingly sophisticated analytics tasks (e.g., support vector machines, logistic regression, collaborative filtering). However, these systems' synchronous (often Bulk Synchronous Parallel) dataflow execution model is at odds with an increasingly important trend in the machine learning community: the use of asynchrony via shared, mutable state (i.e., data races) in convex programming tasks, which has—in a single-node context—delivered noteworthy empirical performance gains and inspired new research into asynchronous algorithms. In this work, we attempt to bridge this gap by evaluating the use of lightweight, asynchronous state transfer within a commodity dataflow engine. Specifically, we investigate the use of asynchronous sideways information passing (ASIP) that presents single-stage parallel iterators with a Volcano-like intra-operator iterator that can be used for asynchronous information passing. We port two synchronous convex programming algorithms, stochastic gradient descent and the alternating direction method of multipliers (ADMM), to use ASIPs. We evaluate an implementation of ASIPs within on Apache Spark that exhibits considerable speedups as well as a rich set of performance trade-offs in the use of these asynchronous algorithms.

1 Introduction

The recent rise of large-scale distributed dataflow frameworks has enabled widespread adoption of increasingly sophisticated analytics tasks at scale [1, 24, 14, 4, 10]. The last decade has seen considerable research and industrial effort put towards understanding how to integrate complex analytics and learning tasks into programmer workflows [25, 12], existing system architectures [6, 8], and new cluster compute frameworks [22, 13].

Simultaneously, in the machine learning community, the statistical nature of many of these analytics tasks has led to increasing interest in exploiting *asynchrony* during computation. That is, a range of recent theoretical results has demonstrated that removing synchronization within an emerging class of problems can yield surprising improvements in performance. These problems can be solved via highly concurrent update mechanisms that expose, in effect, read-write race conditions [5, 16, 20]. As an example, Recht et al. have demonstrated that stochastic gradient descent—typically implemented via serializable locking (and only proven to converge under serial execution)—can be made robust against asynchronous processing over shared, mutable model state: in effect, when conflicts are rare (enough), (some) staleness and data races will not affect statistical correctness [19]. Empirically, on single-node systems, these asynchronous algorithms have yielded order-of-magnitude improvements in performance and are the subject of active research, even within the systems community [6, 26, 27].

Unfortunately, these two trends stand in opposition. Architecturally, commodity distributed dataflow systems such as Hadoop and Spark are optimized for coarse-grained (often bulk synchronous parallel [23]) data transformations and are not designed to natively provide the fine-grained communication required for efficient asynchronous analytics tasks. Consequently, evaluation of these new asynchronous algorithms have been largely confined to single-node, multi-processor (and NUMA) context [26, 19]: it is relatively unknown how the increased latency of a distributed environment impacts their performance and correctness guarantees. The technological trajectory outlined by recent research suggests a divide between widely-deployed dataflow-based cluster compute frameworks and specialized asynchronous optimization mechanisms, which largely rely on a shared memory abstraction [15, 9, 21].

In this work, we study this disconnect by addressing two key questions. First, can increasingly ubiquitous dataflow systems be easily adapted to support asynchronous analytics tasks? Second, in a distributed dataflow environment, what are the benefits (and costs) of these asynchronous algorithms compared with existing synchronous implementations? We present the design and evaluation of a simple dataflow operator that $i.)$ enables implementation of asynchronous complex statistical analytics (primarily, *convex programming* tasks, including Support Vector Machines and Logistic Regression [3]) yet $ii.)$ is implementable using a commodity dataflow engine (Apache Spark). We use this operator to study the implications of bringing distributed asynchrony to two classic convex programming procedures: stochastic gradient descent (SGD) [3] and alternating direction method of multipliers (ADMM) [2]. This juxtaposition of traditional BSP systems and algorithms with their incipient asynchronous counterparts yields an opportunity to study the differences between these paradigms.

To address the first, architectural question, we codify and exploit a common pattern in asynchronous analytics tasks. We observe that, on a single machine, these tasks can be cast as single-stage parallel dataflow, with shared memory acting as a communication channel between operators. Therefore, to allow asynchronous data sharing during distributed operation, we introduce the Asynchronous Sideways Information Passing (ASIP) pattern, in which a set of shared-nothing, data-parallel operators are provided access to a special communication channel, called a *ASIP iterator*, that allows fine-grained communication across concurrent operator instances. The ASIP iterator abstracts the details of distribution and routing (similar to the exchange operator [7] and allows fine-grained communication across operators as in sideways information passing [11]. This enables our target convex programming routines to take advantage of asynchrony within more general purpose distributed dataflow systems. We present the design and implementation of a prototype ASIP ASIP iterator system in Apache Spark and discuss the challenges arising from fault tolerance and scheduling. Notably, in our implementation, the bulk of data transfer and computation occurs via the primary iterator interface, exploiting Apache Spark's strength of efficient parallel computation, while, in the convex optimization routines we study, the ASIP iterator acts as a "control plane" for facilitating fine-grained model synchronization.

To address the second, more algorithmic question, we evaluate the costs and benefits of distributed, asynchronous execution within two common analytics tasks. We first extend BSP SGD (as provided natively in Spark via MLlib [22]) to ASIP gradient descent, using the ASIP iterator to ship fine-grained delta-encoded model updates between operators (approximating a well-studied but—to our knowledge—seldom empirically evaluated algorithm known as dual averaging [5]). We also extend BSP ADMM to the ASIP setting, using the ASIP iterator to ship actual models between parallel operators and leveraging Escrow-like divergence control [18, 17] to bound drift imposed by asynchrony. Across a range of learning tasks, both ASIP algorithms demonstrate speedups of up to two orders of magnitude compared to their BSP counterparts. However, the two ASIP algorithms evince a careful trade-off between speed and safety: the fast delta updates of ASIP GD are remarkably efficient when data is well-behaved but can cause instability in pathological workloads. In contrast, ASIP ADMM behaves well across workloads but is generally slower. To the best of our knowledge, this evaluation is the first apples-to-apples comparison of these techniques at scale in a distributed setting and on real-world data.

In summary, we make the following contributions:

- We present a distributed dataflow operator providing intra-operator sideways information passing that is sufficient to implement asynchronous convex optimization routines within existing dataflow systems.

- We present the design and implementation of two asynchronous convex programming routines—gradient descent and ADMM—within the ASIP operator, drawing on the theoretical machine learning literature when possible.
- We evaluate the costs and benefits of asynchronous convex programming via ASIP within Apache Spark and demonstrate improvements in convergence rates via the use of asynchrony across a range of workloads.

Full text on arXiv; preprint at: `http://db.cs.berkeley.edu/asip/asip-arxiv-preview.pdf`

References

[1] V. Borkar, M. J. Carey, and C. Li. Inside big data management: ogres, onions, or parfaits? In *EBDT*, 2012.

[2] 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):1–122, 2011.

[3] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2009.

[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[5] J. C. Duchi, A. Agarwal, and M. J. Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. *Automatic Control, IEEE Transactions on*, 57(3), 2012.

[6] X. Feng, A. Kumar, B. Recht, and C. Ré. Towards a unified architecture for in-RDBMS analytics. In *SIGMOD*, 2012.

[7] G. Graefe. Encapsulation of parallelism in the volcano query processing system. In *SIGMOD*, 1990.

[8] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, et al. The madlib analytics library: or mad skills, the sql. In *VLDB*, 2012.

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

[10] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[11] Z. G. Ives and N. E. Taylor. Sideways information passing for push-style query processing. In *ICDE*, 2008.

[12] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, 2013.

[13] J. Langford, L. Li, and A. Strehl. Vowpal wabbit online learning project, 2007.

[14] M. Leich, J. Adamek, M. Schubotz, A. Heise, A. Rheinländer, and V. Markl. Applying stratosphere for big data analytics. In *BTW*, 2013.

[15] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.

[16] J. Liu, S. J. Wright, C. Ré, and V. Bittorf. An asynchronous parallel stochastic coordinate descent algorithm. In *ICML*, 2014.

[17] C. Olston. *Approximate Replication*. PhD thesis, Stanford University, 2003.

[18] P. E. O'Neil. The escrow transactional method. *ACM TODS*, 11(4):405–430, 1986.

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

[20] S. Shalev-Shwartz and T. Zhang. Accelerated mini-batch stochastic dual coordinate ascent. In *NIPS*, 2013.

[21] A. Smola and S. Narayanamurthy. An architecture for parallel topic models. In *VLDB*, 2010.

[22] E. R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, M. J. Franklin, M. I. Jordan, and T. Kraska. Mli: An api for distributed machine learning. In *ICDM*, 2013.

[23] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[24] 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 *NSDI*, 2012.

[25] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. SIGMOD, 2014.

[26] C. Zhang and C. Ré. DimmWitted: A study of main-memory statistical analytics. In *VLDB*, 2014.

[27] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin. A fast parallel sgd for matrix factorization in shared memory systems. In *RecSys*, 2013.