Petuum: A Framework for Iterative-Convergent Distributed ML

Wei Dai, Jinliang Wei, Xun Zheng, Jin Kyu Kim
Seunghak Lee, Junming Yin, Qirong Ho and Eric P. Xing
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
wdai,jinlianw,xunzheng,jinkyuk,seunghak,junmingy,
qho,epxing@cs.cmu.edu

Abstract

A major bottleneck to applying advanced ML programs at industrial scales is the migration of an academic implementation, often specialized for a small, well-controlled computer platform such as desktop PCs and small lab-clusters, to a big, less predicable platform such as a corporate cluster or the cloud. This poses enormous challenges: how does one train huge models with billions of parameters on massive data, especially when substantial expertise is required to handle many low-level systems issues? We propose a new architecture of systems components that systematically addresses these challenges, thus providing a general-purpose distributed platform for Big Machine Learning. Our architecture specifically exploits the fact that many ML programs are fundamentally loss function minimization problems, and that their iterative-convergent nature presents many unique opportunities to minimize loss, such as via dynamic variable scheduling and error-bounded consistency models for synchronization. Thus, we treat data, parameter and variable blocks as computing units to be dynamically scheduled and updated in an error-bounded manner, with the goal of minimizing the loss function as quickly as possible.

1 Introduction

Machine learning is becoming the primary mechanism by which information is extracted from Big Data, and which artificial intelligence is built upon. However, despite the rapid and voluminous emergence in recent years of new models, algorithms [6, 18, 11, 21, 1, 4] and execution frameworks [7, 14, 13, 12, 2, 3, 20, 15] across a wide spectrum of applications, successful and effective adoption of ML technology based on truly advanced and large-scale probabilistic or optimization programs — i.e., programs that involve billions of variables, with massive amount of relational or sparsity constraints, processing petabyte-scale datasets, and operating perpetually and autonomously on large computer clusters or in the cloud — remain largely unseen. Most existing large-scale applications still seem to rely on classical approaches such as kNN and decision trees, which impose modest challenges on model and algorithm design, while scalable computational support for such methods can be straightforwardly handled by existing representation, programming, and hardware technology. Thus it is tempting to ask, where have all the latest exciting developments in ML research – such as nonparametric Bayesian models, advanced subspace models beyond topic models, multitask and nonlinear high-dimensional inference theory, and consistent graph learning algorithms, to name a few – gone to in the broader application domains? We conjecture that, from the scalable execution point of view, what prevents many state-of-the-art ML models and algorithms from being more widely adopted is the lack of satisfactory answers to the following needs: 1) a turnkey inference engine; 2) ways to scale on data; 3) ways to scale on model size; and 4) abstraction of hardware/system configuration. Our goal is to develop a distributed Big ML framework that aims at providing a generic yet effective interface to a broad spectrum of ML programs.

Our design philosophy is rooted in *iterative-convergent* solutions to loss function minimization. A great number of ML algorithms are formulated in this manner, which involves repeatedly executing update equations that decrease some error function. Examples of such algorithms include variational methods for graphical models [11], proximal optimization for structured sparsity problems [5], backpropagation on deep neural networks [6], MCMC for determining point estimates in latent variable models [8], among many others. Thus, the core goal of our framework is to execute these iterative updates in a manner that most quickly minimizes the loss function in a large-scale distributed environment. We do so by employing statistical insights such as error-bounded consistency schemes to decrease network communication, and rescheduling of updates to decrease correlation effects and optimizing load-balancing, which are executed by systems components such as a parameter server for global parameter synchronization and a dynamic scheduler to organize and distribute worker tasks. In summary, our system views the iterative-convergent nature of ML as the prime opportunity to be exploited in realizing scalable execution of generic Big ML problems.

2 System Components

We have develop a prototypic framework for Big ML called *Petuum*, which comprises several interrelated components, each focused on exploiting various specific properties of iterative-convergent behavior in ML. The components can be used individually, or combined to handle tasks that require their collective capabilities. In this workshop, we focus on two components:

- **Parameter Server for global parameters:** Our parameter server (Petuum-PS) is a distributed key-value store that enables an easy-to-use, distributed-shared-memory model for writing distributed ML programs over BIG DATA. Petuum-PS supports novel consistency models such as bounded staleness, which achieve provably good results on iterative-convergent ML algorithms [9]. Petuum-PS additionally offers several "tuning knobs" available to experts but otherwise hidden from regular users such as thread and process-level caching and read-my-write consistency. We also support out-of-core data streaming for datasets that are too large to fit in machine memory.
- **Variable Scheduler for local variables:** Our scheduler (STRADS) analyzes the variable structure of ML problems, in order to find parallelization opportunities over BIG MODEL while avoiding error due to strong dependencies. STRADS then dispatches parallel variable updates across a distributed cluster, while prioritizing them for maximum objective function progress. Throughout this, STRADS maintains load balance by dispatching new variable updates as soon as worker machines finish existing ones.

2.1 Parameter Server

Big Machine Learning is challenging because the global model parameters can be massive in size (billions to trillions), while the use of terabyte-scale Big Data places high algorithmic complexity demands on inference. Such memory and computational needs necessitate the use of many machines, thus we develop a general-purpose parameter server (PS) for iterative-convergent ML programming called Petuum-PS. Petuum-PS is a distributed key-value store that provides client machines shared-memory access to global parameters sharded on the server machines. Unlike conventional key-value stores that offer consistency models such as the eventual consistency and the strong consistency, Petuum-PS features a continuum of bounded-staleness consistency guarantees across all clients, which has been shown to achieve provably good results on iterative-convergent ML programs while largely reducing communication overhead [9].

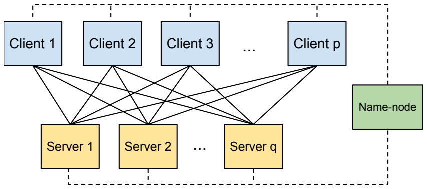

Figure 1: Petuum parameter server topology. Servers and clients interact via a bipartite topology, while a name-node machine handles bookkeeping and assignment of keys to servers.

Petuum-PS offers a table-based user interface: users create global tables in which each entry of the table can be accessed globally by a row-column ID pair, providing a general-purpose, easy-to-program interface [16]. Because the parameter server abstracts away the communication and synchronization, the distributed global table appears to be local to the user program executed on the client machines (distributed shared memory). This allows Petuum-PS to retro-fit eixisting single-machine parallel implementations with minimal modification. In the following sections we highlight several tunable system features: bounded consistency, thread and process-level caching, and the use of out-of-core (disk-based) storage.

2.1.1 Bounded Consistency

For Big Data+Model tasks, many variable blocks must access just as many parameter and data blocks, all spread over 100s -1000s of machines. To reduce network costs and eliminate global barriers/locking, we exploit the error-resistant iterative-convergence nature of ML programs — in other words, their robustness against minor inconsistency in their model state. Petuum-PS provides theoretically-guaranteed consistency schemes that reduce inter-machine synchronization and network communication, such as:

- **Stale Synchronous Parallel (SSP) Consistency:** SSP is based on the concept of *iteration-bounded staleness*: using parameters from a few iterations ago still preserves convergence guarantees [9]. Our PS exploit this by serving locally-cached versions of parameter blocks that are $\leq s$ iterations old, thus eliminating network traffic. Extensions to the SSP model include *heterogenous staleness*, in which different parameter blocks are read with different staleness, and *adaptive staleness*, in which the staleness values are automatically tuned for different phases of the ML algorithm.
- **Value-Bounded Consistency:** In value-bounded consistency, clients and servers are synchronized only when their parameter versions deviate by more than some threshold δ. For global parameter blocks that change infrequently, this strategy requires even less network synchronization than iteration-bounded staleness. Our proposed system will dynamically adjust δ to minimize user intervention.

2.1.2 Process-Level and Thread-Level Caching

Each PS client stores commonly used rows in local memory (process-level "caches") to reduce synchronization and network costs. To reduce the lock-contention between threads on process-level cache, every worker thread on a client machine has its own thread "cache", which is memory exclusive to the thread. A frequently accessed row in table could potentially be cached at both the process-level and thread-level cache. In order to keep memory usage at a reasonable level, intelligent caching and eviction strategies are necessary. Our system offers multiple strategies such as Least-Recently Used (LRU), Two-List LRU, and Priority LRU, in order to handle different scenarios.

Petuum-PS allows user to specify the number of rows to cache at each cache level, which controls the memory footprint on the client machine. These parameters provide interesting performance trade-offs: increasing the thread-level cache size provides faster data access but data in thread cache, duplicated for each thread, could crowd out the process-level cache storage. Petuum-PS provides reasonable default values but we expect expert users have much to gain by experimenting with these parameters.

2.1.3 Out-of-Core Storage Support

In many cases, running ML algorithms on Big Data requires a cost-prohibitive amount of memory (e.g., $10^9 \sim 10^{10}$ documents in topic model). By discovering sequential access patterns in the algorithm (e.g. sequential read of documents), we can utilize the out-of-core (disk-based) storage by efficiently streaming data from hard disks or SSDs. Petuum-PS uses queue-based read-buffer and write-buffer to perform asynchronous disk I/O and hide latency. One can thus iterate over a large dataset with a small sliding window that fits in memory. While such out-of-core execution may result in speed penalty, it nonetheless enables ML algorithms on otherwise-intractable datasets.

2.2 STRADS Scheduler

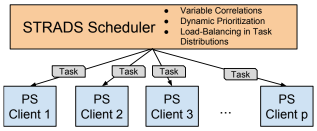

Figure 2: STRADS architecture. The worker machines can be Petuum-PS clients (as shown in the diagram) or nodes without PS support.

Big Models contain millions (e.g., whole-genome regression), if not billions (e.g., Google brain DeepNet) of parameters, which require clusters with 100s - 10,000s of processors running inference algorithms in parallel. Examples of big models include ultra-high dimensional regression, DNN, and complex latent space models, which allow for rich data analysis far beyond simple classification or clustering. The complexity of these models raises two major issues: 1) a strategy is needed to divide and exploit model structure, in a way that ensures iterative-convergent consistency and statistical guarantees; 2) different model variables may have non-uniform importance to overall convergence, which must be taken into account. We develop a structure-aware dynamic scheduler (STRADS) to address these issues in distributed inference/learning on large models.

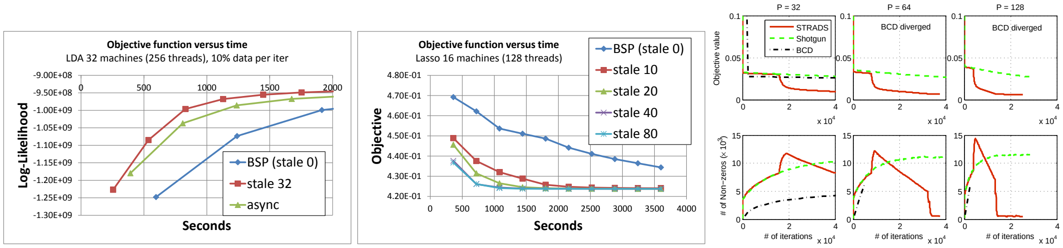

Figure 3: **Left, Center:** Parameter server performance under SSP consistency, versus BSP and asynchronous consistency, on LDA (NYtimes dataset, 256 cores) and Lasso regression (synthetic data, 128 cores). Both graphs plot the objective function versus time (higher is better for LDA, lower is better for Lasso). **Right:** STRADS scheduler performance on Lasso regression, versus the Shotgun and BCD algorithms (for 32, 64 and 128 cores). The top row compares objective value versus iteration number (lower is better), while the bottom row compares solution quality, in terms of sparsity (lower is better).

2.2.1 Dynamic Scheduling and Adaptive Load Balancing

For distributed model variables updates to be efficient, a scheduler is needed to partition the variables in a manner that minimizes parallelization error. Our STRADS scheduler accounts for the relationships between variables and the relative importance of each variable to the objective function; thus, STRADS can dynamically prioritize the most important variables for distributed updates, while minimizing parallelization error due to intereference between variables. In this manner, we use information about the ML problem to guarantee convergence at the fastest possible rate. As an example, the Lasso parallel coordinate descent algorithm is known to converge slowly when highly correlated variables are updated simultaneously [4]. STRADS avoids scheduling correlated variables together, thus minimizing any loss in convergence rate.

The updates on model variables may also change in computational complexity as the ML algorithm progresses, leading to uneven network and CPU workloads. STRADS actively monitors the contribution of each variable update to the objective function, and weighs it against the actual time taken for the update. It then uses this information to re-prioritize variables as the algorithm progresses, thus shifting computation to parts of the model that can benefit more. For example, in a structured-input-output regression algorithm, each worker will estimate new coefficients for every input and output group, via an expensive proximal operator. However, for groups with all-zeros, a cheap thresholding operator can be used instead, and furthermore, the emergence of such groups can be predicted a few iterations in advance. STRADS uses such knowledge of computational requirements to schedule jobs in a load-balanced manner.

3 Preliminary Results and Proposed Demonstration

We have published results for parameter server performance under the SSP consistency model for a variety of algorithms (LDA, MF, Lasso) [9], and we have preliminary results for the STRADS scheduler on Lasso regression, versus the Shotgun [4] and Blocked Coordinate Descent (BCD) [17] algorithms. Some of the highlights can be found in Figure 3. During the workshop, we plan to demo both Petuum-PS and STRADS running on large-scale ML applications, such as:

- Network role analysis on 100-million node networks, using the Mixed Membership Triangular Model (MMTM) [10, 19].
- Lasso regression on ultra-high dimensional genomic data, with ≥10-million dimensions.
- Latent Dirichlet Allocation on millions of documents, using 10,000s of topics.
- Matrix Factorization on matrices with ≥100-million nonzero entries.

References

[1] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 5451–5452. IEEE, 2012.

[2] Apache. Apache mahout: Scalable machine learning and data mining. http://mahout.apache.org/, October 2013.

[3] Vinayak Borkar, Michael Carey, Raman Grover, Nicola Onose, and Rares Vernica. Hyracks: A flexible and extensible foundation for data-intensive computing. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1151–1162. IEEE, 2011.

[4] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization. *ICML*, 2011.

[5] Xi Chen, Qihang Lin, Seyoung Kim, Jaime Carbonell, and Eric Xing. Smoothing proximal gradient method for general structured sparse learning. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, 2011.

[6] 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*, 2012.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101(Suppl 1):5228–5235, 2004.

[9] Q. Ho, J. Cipar, H. Cui, J.-K. Kim, S. Lee, P. B. Gibbons, G. Gibson, G. R. Ganger, and E. P. Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems 26*, 2013.

[10] Qirong Ho, Junming Yin, and Eric Xing. On triangular versus edge representations—towards scalable modeling of networks. In *Advances in Neural Information Processing Systems 25*, pages 2141–2149, 2012.

[11] Matt Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *arXiv preprint arXiv:1206.7051*, 2012.

[12] Tim Kraska, Ameet Talwalkar, John Duchi, Rean Griffith, Michael J Franklin, and Michael Jordan. Mlbase: A distributed machine-learning system. In *In Conference on Innovative Data Systems Research*, 2013.

[13] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *PVLDB*, 2012.

[14] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[15] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[16] Russell Power and Jinyang Li. Piccolo: building fast, distributed programs with partitioned tables. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 1–14. USENIX Association, 2010.

[17] Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David Haglin. Feature clustering for accelerating parallel coordinate descent. *NIPS*, 2012.

[18] Sinead A. Williamson, Avinava Dubey, and Eric P. Xing. Parallel markov chain monte carlo for nonparametric mixture models. In *International Conference on Machine Learning*, 2013.

[19] Junming Yin, Qirong Ho, and Eric P Xing. A scalable approach to probabilistic latent space inference of large-scale networks. *Advances in neural information processing systems*, 2013.

[20] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.

[21] Martin Zinkevich, John Langford, and Alex J Smola. Slow learners are fast. In *Advances in Neural Information Processing Systems*, pages 2331–2339, 2009.