Adaptive Consistency Management for Distributed Machine Learning

by

Christoph Alt

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

TECHNISCHE UNIVERSITÄT BERLIN

March 2017

Author . . . .
Department of Electrical Engineering and Computer Science
March 15, 2017

Certified by . . . .
Prof. Dr. Odej Kao
Associate Professor
Thesis Supervisor

Accepted by . . . .
Prof. Dr. Voker Markl

Adaptive Consistency Management for Distributed Machine Learning

by

Christoph Alt

Submitted to the Department of Electrical Engineering and Computer Science
on March 15, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

In recent years, machine learning emerged as an essential part of many successful applications and businesses. The ever growing amount of data requires these algorithms, many of them sequential in nature, to be executed in a distributed manner on a large scale. This is commonly achieved by exploiting the algorithms stochastic nature to allow for parallel execution at the expense of lowered consistency among the distributed data structures. Even though the quality of the result is not affected, an improper level of consistency can severely affect algorithm performance, resulting in a non optimal convergence rate. Furthermore the level of consistency required to achieve the best performance can change during different periods of algorithm execution. System architecture, topology, algorithm, hardware and data properties influence performance as well. Despite its widespread use, the implications of these aspects of distributed systems on algorithm performance are not yet well understood. This thesis aims to answer the question of how the level of consistency affects the overall performance of distributed machine learning algorithms and also aims to identify strategies for consistency management that can help to mitigate the negative effect on performance. Based on the identified strategies, a protocol will be developed that is able to manage the consistency level of distributed state. The protocol will be designed in a way that allows for dynamic changes in consistency. This enables the possibility of adaptive consistency management in distributed machine learning environments based on heuristics and algorithm progress.

Thesis Supervisor: Prof. Dr. Odej Kao
Title: Associate Professor

Acknowledgments

This is the acknowledgements section. You should replace this with your own acknowledgements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the most challenging tasks in computer science and engineering resolves around improving algorithm performance. In general this has been done by making hardware faster and inventing new strategies and algorithms to parallelize work more efficiently. Since it is clear that Moore's Law will not hold anymore, a lot of effort has been spent to horizontally scale algorithm computation across multiple machines. Machine learning is no exception and efficient parallelization is a key aspect towards more intelligent systems. By now, many general purpose frameworks for large scale data processing have been developed and published. Many of those are used for running more complex machine learning algorithms at scale as well. Unfortunatelly, the performance often is not satisfying due to the architecture and programming model not reflecting the underlying structure of most commonly used machine learning algorithms. Common data processing tasks can be represented as an extract-transform-load (ETL) pipeline, which is often easily parallelizable. This does not hold for machine learning, where algorithms are mostly sequential in nature and the only way of enabling parallel computation is by exploiting their inherent stochastic properties. This allows to break the sequential execution in favor of parallel learning on subsets, which then needs to be combined in order to obtain a global solution to the task. While this can lead to a great speedup in terms of the amount of data processed, it can have a negative affect on the learning progress. Therefore a key part of horizontally scaling machine learning algorithms is to ensure all participating learners have a consistent view on each others progress while at the same time maintaining a tradeoff between communicating progress and spending time on their own learning task.

1.1 MapReduce and Beyond

Many of todays successful businesses throughout fields like finance, e-commerce and healthcare rely heavily on the ability to process vast amounts of user or sensorical data, collected to make services smarter and user experience better. In order to learn from the collected data, discover patterns and ultimately gain new insights, it needs to be processed by an algorithm. It is not uncommon that the input ranges somewhere between hundred gigabytes and tenth of petabytes. In the past, processing this much data required either a supercomputer, which was only available to large institutions or government entities or some proprietary compute cluster. All this changed when Google introduced MapReduce [1] in 2004. The MapReduce framework made it possible to process data in a distributed and fault tolerant way with the help of a compute cluster formed by hundredth or thousandth of machines. Instead of using a single, expensive, special hardware supercomputer the framework provides the foundation to assemble commodity hardware machines into a compute cluster. The framework takes care of all necessary aspects to ensure a fault tolerant and parallel execution of a task submitted to the cluster. The advantage compared to previous approachs is that the framework can be run entirely on top of machines using commodity hardware, which does not require special hardware and therefore equals low cost.

MapReduce esentially led the path to a convenient and widespread use of big data processing, which found its open source implementation in the Apache Hadoop project [2]. The project quickly gained traction and has spawned many business grade platforms, which quickly gained widespread adoption and by now provides a whole ecosystem around big data processing. The software stack includes a fault tolerant distributed filesystem (HDFS) a MapReduce framework and a cluster resource manager (YARN) [3]. On the other hand, MapReduce suffers from some practical limitations that lead to the development of new, more sophisticated and specialised big data frameworks. With the most widely used frameworks being Apache Spark [4], Apache Flink [5] and GraphLab [6]. The first two frameworks use at it's core a dataflow pipeline based architecture, whileas the latter uses a graph abstraction to model particular algorithms. All this works well for algorithms that can be expressed as an extract-transform-load (ETL) pipeline and are often embarrassingly parallel in nature. On the other hand, machine learning algorithms often rely heavily on many, computationally light, iterations to iteratively update a shared state (model) such as logistic regression or latent dirichlet allocation (LDA). These so called iterative-convergent algorithms required a change in how systems for distributed machine learning operate at its core.

1.2 Distributed Machine Learning

This limitation essentially sparked the development of specialized frameworks for distributed execution of iterative-convergent algorithms used in common machine learning tasks. The most widely recognized systems are Petuum [7], ParameterServer [8] and MALT [9]. Different from the MapReduce paradigm, these frameworks, instead of using a pipeline to transform an immutable dataset into another immutable dataset, operate on a fixed but mutable state, which is held by a single machine or distributed over multiple machines. This state can then be updated by workers that have computed an update locally and send it to these state keepers. Which act as surrogates, taking state updates and incorporating these into the state by some user defined function. While these systems can increase the performance on machine learning algorithms by an order of magnitude (c) compared to dataflow systems, most systems come with either limited usability, which makes it difficult to implement additional algorithms, are tied to a specific algorithm or are very low level frameworks. Efficiently distributing machine learning algoritms while at the same time provide the ability to conscisely express machine learning algorithms remains an extremely challenging problem. A system targeting the execution of those algorithms at scale must therefore provide the ability to conscisely express the underlying algebraic structure and at the same time be flexible enough to allow experimentation and fine tuning. Where consistency management is an essential part to ensure that algorithms are executed both fast and also learning performance is well enought.

1.3 Thesis Outline

In this thesis we introduce a novel framework for large scale distributed machine learning. It improves upon currently available systems by providing a powerful programming abstraction that can conscisely express state of the art machine learning algorithms and at the same time minimizes the effort necessary to move from a single machine to a cluster. The framework design is optimized for efficient parallel asynchronous execution of iterative-convegent algorithms in a cluster and ensures the required consistency is enforced among parallel learners, depending on the algorithm properties. By allowing the user to decide how to maintaining the best tradeoff between algorithm execution and progress communication the performance is improved as well. All of this can be easily customized for quick prototyping and finetuning, which makes the system suited for developers as well as researchers. The goal of this thesis is to implement the state centric programming model and show its performance in comparison with Apache Spark on an example implementation of the CoCoA (cite) framework. I start off by providing a background on the architecture and inner workings of state of the art frameworks for big data processing and distributed machine learning in Chapter 2. Furthermore the most commonly used algorithms and optimization techniques are introduced. The majority of those algorithms can be classified into the group of so called iterative-convergent algorithms for which I am going to provide a more formal description. I introduce the common algorithm parallelization strategies in distributed machine learning. I conclude the chapter by providing an overview over the challenges and issues that need to be addressed and considered when developing a distributed machine learning system and how this is achieved by current frameworks. This will give rise to understanding why a different framework architecture and abstraction is necessary to improve the performance and expressability of large scale distributed machine learning applications. Chapter 3 therefore introduces the state centric programming model, which treats the state as a mutable first class citizen, which can be distributed and altered by updates that result from distributed computation. This allows the system to reason about the most optimal distribution of state in the cluster which is then scheduled with computation that can update the state. I further describe the architecture of the system and how the essential components are implemented. When updating a shared state from multiple different locations, consistency must be maintained in order to ensure the algorithm progresses as expected. The system is responsible for managing a state's consistency among all of its instances across the cluster. Chapter 4 therefore describes several schemes for ensuring consistency and at the same time optimizing for bandwidth and computational resource usage. In order to show the system and its consistency management in action, Chapter 5 compares the system against Apache Spark by running the CoCoA framework on two datasets with linear regression as the choosen algorithm. Chapter 6 summarize the experiments with the lessons learned and provides suggestions on how to further improve systems for large scale distributed machine learning.

Chapter 2

Background

Current state of the art distributed data processing systems like Apache Spark and Apache Flink are based on the dataflow concept which is introduced in Section 2.2. The section introduces the concept behind dataflow and how it is utilized to provide distributed and fault tolerant execution of data processing tasks in a cluster of commodity hardware machines. Section **??** provides a formal introduction to iterative-convergent algorithms, a group of algorithms most of the commonly used algorithms in distributed machine learning belong to. The section provides the foundation for a more in depth discussion of commonly used algorithms and optimization techniques in Section 2.3. The section also provides an overview over the current state of the art distributed machine learning frameworks and introduces the reader to the challenges that come with parallelizing machine learning algorithms across a cluster of parallel learners.

2.1 Algorithms and Optimization

2.1.1 Iterative Convergent Algorithms

Consider a supervised learning setup with a dataset $D = \{z_1, \ldots, z_n\}$ with each example z_i being represented by a pair (x_i, y_i) consisting of an input x_i and a scalar output y_i. Consider also a loss function $\ell(\hat{y}, y)$ quantifying the cost of predicting $\hat{y}$

when the true output is y. As a model, a family F of functions $f_w(x)$ parameterized by a weight vector w is chosen. The goal is to find a function $f \in F$ that minimizes the loss $Q = \frac{1}{n}\sum_{i=0}^{n} \ell(f_w(x_i), y_i)$. In order to find an optimal solution many algorithms used in large scale machine learning such as regression, topic models, matrix factorization or neural networks employ either gradient based methods or markov chain monte carlo methods. In order to find the optimal solution those algorithms try to iteratively update the weight vector w. At each iteration t an updated weight vector w^t is computed based on the vector of the previous iteration $w^{(t-1)}$ and the data D. The resulting model f_{w^t} is again a better summary of the data D under the objective Q. Eq. 2.1 shows the process of refining the model, with Δ being an arbitrary update function.

$$w^t = w^{(t-1)} + \Delta(w^{(t-1)}, D) \tag{2.1}$$

The update function depends on the algorithm employed by the analytic and can be seen as a procedure of obtaining a step towards a better model. At each iteration an update Δw is computed and applied to the previous weight vector until a stopping condition is satisfied. The distance to the optimal solution or the objective difference between two iterations is monitored. When the difference is below a certain threshold the computation stops and the algorithm is said to be converged.

2.1.2 Optimization

In order to arrive at an optimal solution represented by a function $f_{w^*} \in F$, various optimization techniques can be employed. The focus throughout this thesis lies on algorithms that solve linear loss optimization problems of the following objective form

$$Q = \ell(w) + r(w) \tag{2.2}$$

where ℓ and r are both convex functions and ℓ is smooth. Commonly the term ℓ is an emperical loss over the data of the form $\sum_i \ell_i(w)$ and the term r is a regularizer, e.g. $r(w) = \lambda\|w\|_p$. Many of the beforementioned algorithms in machine learning can be expressed in this form, such as logistic and linear regression, lasso and sparse logistic regression and support vector machines. Depending on the structure of the problem, the most widely used techniques are stochastic gradient descent (SGD) and stochastic coordinate descent (SCD). Stochastic gradient descent

2.1.3 CoCoA

- start from minibatch and argue that increased minibatch size degrades performance

2.2 Dataflow Systems

2.3 Distributed Machine Learning

Distiributed machine learning posses a number of unique challenges resulting from the fact that most iterative convergent algorithms are sequential in nature. Parallelizing such an algorithm can be done by exploiting either the stochastic nature inherent to these algorithms or by exploiting decomposability of model parameters. Doing so leaves the developer with numerous possibilities and responsibilities regarding the distribution of data accross the workers, the formation of workers and the communication and synchronization scheme to be employed. Various frameworks have been published which all tackle these problems in a specific way. Often focusing on a fixed master/worker architecture with specific algorithsm.

List the challenges of distributed ML (synchronization/consistency, distributing state and model, choosing the best formation of workers (maybe different jobs (computing updates, merging updates))), choosing the best communcation model, show picture

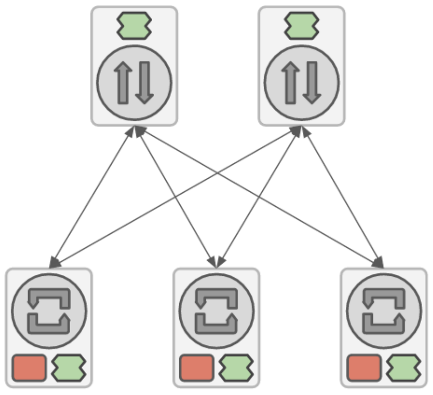

Figure 2-1: Parameter Server

2.3.1 Parameter Server

2.3.2 Consistency

The most important part of any distributed system is the synchronization strategy used to ensure consistency among multiple nodes concurrently accessing and updating the parameters stored on the parameter server. There are multiple schemes to synchronize nodes during the iterative parameter refinement. *Bulk synchronous parallelization (BSP)* leads to the best algorithm throughput (e.g. convergence achieved over the number of data points processed). Essentially each worker must finish its iteration and push all updates to the parameter server. The server then computes a refined model according to Eq. 2.3 and each node retrieves the updated parameters before beginning the next iteration. This synchronization scheme guarantees consistency among all nodes at all times.

$$W^t = W^{(t-1)} + \frac{1}{K}\sum_{k=1}^{K} \Delta(W_k^{(t-1)}, D_k) \tag{2.3}$$

While this synchronization strategy essentially recovers the sequential algorithm for a single machine and has the same convergence properties and guarantees, it suffers from a severe limitation when used in a distributed setup [?]. Imagine one of the workers is for some reason a lot slower than the others. Due to the synchronization strategy, the other workers have to wait for this particular worker to complete its iteration.

This is well known as the straggler problem [?] and can seriously affect performance in a distributed environment, because the progress is limited by the slowest node in the cluster. The second strategy is *total asynchronous parallelization (TAP).* Similar to BSP, all nodes push their parameter updates to the server after each iteration but in this case, the changes are applied to the model immediately. No waiting for other workers is required, resulting in a very high data throughput. The straggler problem can be mitigated by this synchronization scheme as well, depicted in Figure **??**. Even though worker 3 is a straggler, the remaining workers can continue with their next iterations without waiting for slower workers. Although this consistency scheme seems to work quite well in practice [?], it lacks formal convergence guarantees and can even diverge [?]. The reason is that no bound exists for a situation where the divergence in iterations between the slowest and the fastest worker is unbound. A middle ground between bulk synchronous parallelization and total asynchronous parallelization is *stale synchronous parallel (SSP)* [?] or *bounded staleness (BS).* As shown in Figure **??**, BSP introduces a maximum delay, or staleness threshold, of Δ_{max} between the slowest and fastest node. This overcomes the limitation of the TAP approach by introducing a bound on the number of iterations. Formal convergence guarantees can be restored while still maintaining the flexibility of asynchronous parallelization and limiting the straggler problem [?].

Chapter 3

State Centric Programming Model

Chapter 4

Consistency Management

Chapter 5

Experiments

Chapter 6

Conclusion

Bibliography

[1] J. Dean and S. Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters," *Proceedings of 6th Symposium on Operating Systems Design and Implementation*, pp. 137–149, 2004.

[2] A. Hadoop, "Hadoop," 2009.

[3] V. Kumar Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O 'malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," *SOCC '13 Proceedings of the 4th annual Symposium on Cloud Computing*, vol. 13, pp. 1–3, 2013.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark : Cluster Computing with Working Sets," *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, p. 10, 2010.

[5] A. Alexandrov, R. Bergmann, S. Ewen, J. C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke, "The Stratosphere platform for big data analytics," *VLDB Journal*, vol. 23, no. 6, pp. 939–964, 2014.

[6] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, and C. Guestrin, "GraphLab: A Distributed Framework for Machine Learning in the Cloud," *The 38th International Conference on Very Large Data Bases*, vol. 5, no. 8, pp. 716–727, 2012.

[7] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, and X. Zheng, "Petuum : A New Platform for Distributed Machine Learning on Big Data," 2015.

[8] M. Li, D. G. Andersen, J. W. Park, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, B.-y. Su, M. Li, D. G. Andersen, J. W. Park, A. J. Smola, and A. Ahmed, "Scaling Distributed Machine Learning with the Parameter Server," *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014.

[9] H. Li, A. Kadav, E. Kruus, and C. Ungureanu, "MALT : Distributed Data-Parallelism for Existing ML Applications," 2015.