Thesis Proposal

Scaling Distributed Machine Learning with System and Algorithm Co-design

Mu Li

October 2016

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

For a lot of important machine learning problems, due to the rapid growth of data and the ever increasing model complexity, which often manifests itself in the large number of model parameters, no single machine can solve them fast enough. Therefore, distributed optimization and inference is becoming more and more inevitable for solving large scale machine learning problems in both academia and industry. Obtaining an efficient distributed implementation of an algorithm, however, is far from trivial. Both intensive computational workloads and the volume of data communication demand careful design of distributed computation systems and distributed machine learning algorithms.

In this thesis, we focus on the co-design of distributed computing systems and distributed optimization algorithms that are specialized for large machine learning problems. We propose two distributed computing frameworks: a parameter server framework which features efficient data communication, and MXNet, a multi-language library aiming to simplify the development of deep neural network algorithms. In less than two years, we have witnessed the wide adoption of the proposed systems. We believe that as we continue to develop these systems, they will enable more people to take advantage of the power of distributed computing to design efficient machine learning applications to solve large-scale computational problems. Leveraging the two computing platforms, we examine a number of distributed optimization problems in machine learning. We present new methods to accelerate the training process, such as data partitioning with better locality properties, communication friendly optimization methods, and more compact statistical models. We implement the new algorithms on the two systems and test on large scale real data sets. We successfully demonstrate that careful co-design of computing systems and learning algorithms can greatly accelerate large scale distributed machine learning.

1 Introduction

Among the recent machine learning technology advances, there are two prevalent trends: the training data is getting bigger and bigger [13], and the models are becoming deeper and deeper [7]. The scaling in both training data and model complexity provides both opportunities and challenges. Both "big data" and "deep learning" significantly increase the computational cost of machine learning applications. For example, companies often need to learn machine learning model from data with terabytes size [2], and a state-of-the-art image classifier employs hundreds of layers in convolutional networks [5], in which processing a single image requires billions of float-point operations. At such a large scale, although the computational power of modern hardware grows exponentially, no single machine can finish the training tasks at the speed that meets the industrial demands.

Distributed computing is a common approach to tackle the problem of large scale machine learning. The basic idea of distributed computing is to partition the computational workload and assign different parts to different machines, which coordinate to complete the overall workload. Recently, thanks to the increasingly convenient access to public cloud services, such as Amazon AWS [1], Google Cloud [4] and Microsoft Azure [14], using distributed computing to accelerate large scale machine learning has led to a surge of research interest in both academia and industry.

However, it is highly non-trivial to either design or implement efficient machine learning algorithms using multiple machines. In particular, both the iterative nature of a lot of learning algorithms and the sheer size of the models and the training data require a large amount of communication between different machines. Nevertheless, today even in the leading industrial data centers, both the network bandwidth and the latency for communication between multiple machines is 10 times worse than that within a single machine. The communication overhead is indeed the major bottleneck that prevents us from enjoying the benefit of distributed computing in large scale machine learning problems. There are also other challenges brought into the machine learning applications by distributed computing, such as fault tolerance when one or few of the machines break down, and frequent synchronization among different machines, which all need to be carefully addressed.

This thesis aims to address the problem of large scale machine learning using careful co-design of distributed computing systems and distributed learning algorithms. For a wide class of large scale machine learning applications, we propose new distributed computing frameworks, new machine learning models, and new optimization methods with theoretical guarantees to shows that distributed machine learning can be made simple, fast, and scalable. Next, we give a briefly overview of the three areas and highlight the corresponding challenges.

2 Background

2.1 Statistical Models

The realm of machine learning is mainly divided into supervised and unsupervised learning. In supervised learning, the training data consists of pairs of input objects and output values, and the learning goal is to infer a mapping from input objects to output values, which can be used for mapping new instances. While in unsupervised learning, the training data contains only the input objects with unknown output values, and the learning goal is to find "interesting patterns" from the data. In this thesis, we consider several representative applications in each category as motivating examples for our system and algorithm design. These applications range from simple linear model to complex neural network with hundreds of layers.

Today, the key challenge in a lot of applications is the rapidly increasing size of training data and the model complexity. For instance, a large Internet company wants to use one year's ads impression log to train an ad click predictor [6]. The training data thus consists of trillions of examples, each is typically represented by a high-dimensional feature vector [2]. Figure 1 shows that the size of training data for ad click estimation in a leading Internet company doubled every year form 2010 to 2014. A lot of widely used machine learning algorithms process the training data in an iterative way, requiring a large amount of computing and communication resources when the data is at such a large scale. Meanwhile, larger data sets also enable and encourage the developers to use more complex machine learning models to discover finer structures in the data. Take deep learning as an example, the model size in terms of the depth of the neural network has been consistently increasing since the 1980s. In 1989, LeNet, the most widely used convolutional neural network back then, only had 5 convolutional layers; while the recent ImageNet challenge winners [5, 17] employed hundreds of convolutional layers. More complex models are often

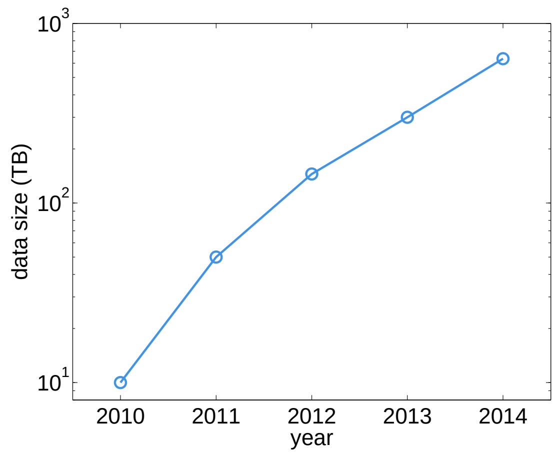

Figure 1: The size of training data for ad click estimation in a leading Internet company from year 2010 to 2014.

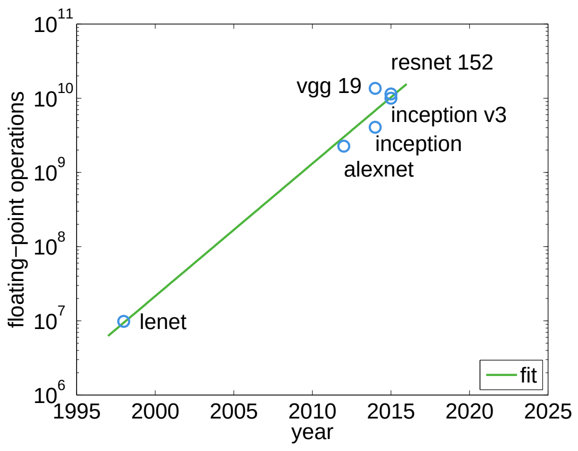

Figure 2: The number of floating-point operations required for processing a single image using LeNet and several recent ImageNet challenge winners.

associated with higher computational cost. Figure 2 shows that the number of floating-point operations to process a single example in the training data has increased from 10 million in the late 1980s to over 10 billion today. Besides the total computational cost, deeper neural networks also bring in more complex computational patterns—even just evaluating a single example involves hundreds of tensor operations.

With larger amount of training data and more complex machine learning models, the rapidly increasing computation workload calls for new solutions from the system design and algorithm implementation.

2.2 Distributed Computing

A distributed computing system consists of multiple nodes with computing power, which are connected through a network. We focus on cluster computing in which actual computers are connected via local networks. Examples of cluster computing include campus cluster machines, or public cloud service such as Amazon AWS. In this setting, the structure of the system and the network topology are often known in advance.

In a distributed computing system, each computing machine has its own private memory, which is difficult to be directly accessed by another machine. Information exchange between machines is conducted via exchanging messages over the communication network. Also, the network that connecting the machines has limited bandwidth, which makes communication one of the scarcest resources in the system. Moreover, a machine may fail at any time, so that an running job can be preempted. Such unreliability gets worse when the number of machines and the workload increase. These features of distributed systems indeed impose great challenges on developing efficient and reliable applications using distributed computing.

Next, we introduce three desired properties of distributed computing systems:

Efficiency Even though the computing cost may be shared among multiple machines, distributed computing also incurs the cost of machine synchronization and communication. Compared to accessing data within a single machine, both the communication latency and the communication bandwidth in distributed computing are significantly worse. In addition to the communication cost, since the machines may be running heterogeneous tasks, and thus some machines may finish their tasks faster or slower. In this case, if machine synchronization is required in the algorithm implementation, the computing power of all the machines may not be fully used at any time.

A good distributed computing system should incur only a small communication cost and should take full advantage of the available computing power.

Fault Tolerance As we mentioned, in distributed computing, a single machine may fail at any time, and an running job can be preempted due to such machine breakdown. The number of failures increases with the number of machines in the systems and with the running time of the tasks. An important feature of a good distributed computing system is the robustness to such failures, namely fault tolerance.

To illustrate the importance of fault tolerance, we collect the job logs from a computing cluster running batch machine learning tasks for production in a large Internet company, over a three month period. Here, task failures are mostly due to being preempted or losing machines without necessary fault tolerance mechanisms. Table 1 shows the statistics of failure rate for different tasks at various scale. The failure rate can be as high as 25% for large scale problem needs over 10 thousand machine hours.

Easy to use It is desired that the programming interface of a distributed system strikes a balance between simplicity and flexibility. On one hand, the interface should hide as much as possible implementation details of using multiple machines, such as task allocation, data communication,

$\approx$ #machine $\times$ time	# of jobs	failure rate
100 hours	13,187	7.8%
$1,000$ hours	1,366	13.7%
$10,000$ hours	77	24.7%

Table 1: Failure rate for machine learning jobs in a data center over a three month period.

synchronization and machine failure, from the developers. On the other hand, the interface should also be flexible enough so that developers can conveniently implement a wide range of algorithms in the system.

2.3 Optimization Methods

In machine learning, a lot of model parameter learning problems can be formulated into the following minimization problem [18]:

$$\underset{w\in\Omega}{\text{minimize}}\ f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w), \tag{1}$$

where w denotes the parameter of the model and Ω denotes the parameter space. The function f_i is the objective function evaluated with the i-th example, describing how well the model w fits the particular example in the training data.

For general objective function f_i, there is usually no explicit solution to the optimization problem. A common way to get a numerical solution is the iterative gradient method, which refines the model w through multiple iterations of gradient flow operations. The basic idea is to start with an arbitrary point $w_0 \in \Omega$ at time 0, and then update the model as below for $t = 1, 2, \ldots$.

$$w_{t+1} = w_t - \eta_t \sum_{i=1}^{n} \partial f_i(w_t), \tag{2}$$

where ∂f_i is the partial gradient of f_i with respect to the model parameter w, and the scalar η_t is the learning rate specifying the step size. The iterations terminate if a stopping criteria is reached.

The computational bottleneck of iterative gradient methods is often the cost of calculating the gradients $\partial f_i(w)$ in each iteration. It is possible to partition and share this computational workload among multiple machines. Algorithm 1 sketches the approach called data parallelism, where in each iteration, the training data is partitioned and shared among all machines, and each machine only calculates the gradients of the objective function evaluated on the part of data assigned to it.

One important measure of the performance of an iterative optimization algorithm is its convergence rate, namely the amount of computing time required in order to achieve desirable accuracy. There is a rich body of research results on accelerating the vanilla gradient descent method. For example, stochastic gradient descent (SGD) [15] only samples a small subset of examples to compute the gradients so that each iteration takes much less time. Given the same amount of total time, this approach can run more number of iterations to update the model parameter, and can hopefully get faster convergence.

Algorithm 1 Distributed Gradient-based Optimization

```
1: Initialize $w_0$ at every node
2: Partition data into $m$ parts, $\{1, \ldots, n\} = \bigcup_{k=1}^m I_k$
3: for $t = 0, \ldots$ do
4:     for $k = 1, \ldots, m$ do in parallel
5:         Compute $g_t^{(k)} \leftarrow \sum_{i \in I_k} \partial f_i(w_t)$ on node $k$
6:     end for
7:     Aggregate $g_t \leftarrow \sum_{k=1}^m g_t^{(k)}$ on node 0
8:     Update $w_{t+1} \leftarrow w_t - \eta_t g_t$ on node 0
9:     Broadcast $w_{t+1}$ from node 0 to all nodes
10: end for
```

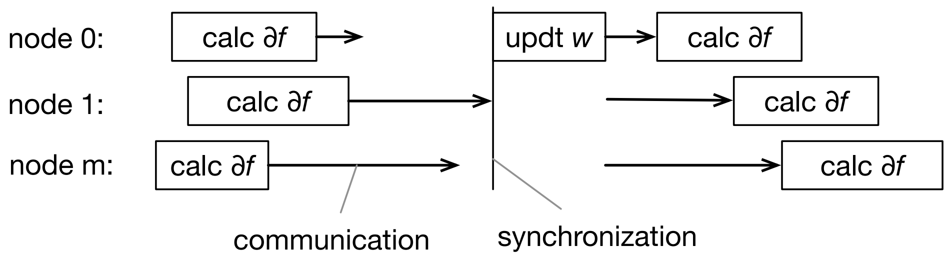

Figure 3: The communication and synchronization overhead of Algorithm 1.

Another critical measure of the performance is the communication cost, including the amount of communication required between the machines when they coordinated to complete the task, and the amount of time that the machines stay idle due to communication latency or for the purpose of system synchronization. For example, Figure 3 illustrates a possible distributed implementation of Algorithm 1. The problem with this implementation is that if the data is partitioned unevenly or one machine is significantly slower than other machines, all other machines would waste a large amount of time in the synchronization.

An efficient distributed optimization method should aim to achieve fast convergence and small communication cost simultaneously.

3 Thesis Statement

This thesis seeks to address the multifaceted challenges arising in distributed computing, statistic modeling, and numerical optimization methods to make large-scale distributed machine learning more accessible. These three areas are not independent technological blocks. They influence each other, which is illustrated in Figure 4, and thus must be handled simultaneously.

In particular, this thesis provides evidence to support the following statement:

> **Thesis Statement:** *With appropriate frameworks and algorithms, distributed machine learning can be made simple, fast, and scalable, both in theory and in practice.*

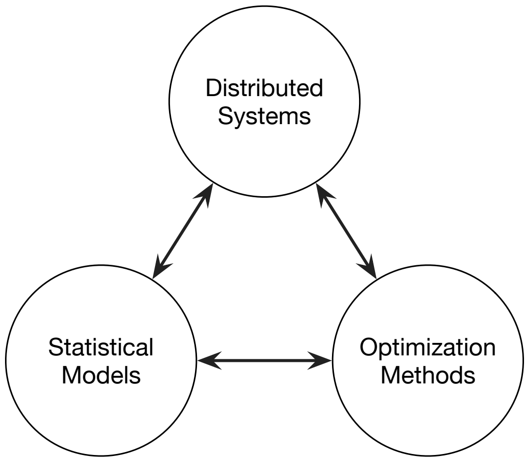

Figure 4: Three aspects towards efficient distributed machine learning.

We believe that the frameworks and algorithmic ideas developed in this thesis will enable more people to take advantage of the power of distributed computing to develop efficient machine learning applications to solve large-scale computational problems.

All the codes developed when completing this thesis are made publicly available at `https://github.com/dmlc/` under Apache 2.0 license.

4 Tentative Outline of the Thesis

Next, we briefly describe the tentative structure of the thesis. The thesis is mainly divided into two parts: distributed computing system (Section 2 to 4), distributed optimization methods (Section 5 to 10). Figure 5 visualizes the connections between different sections.

Part 1 In the first part of the thesis, we introduce two computing frameworks designed for large scale distributed computing: a new generation parameter server and MXNet. The key features of these two systems are summarized below.

Parameter server is a general purpose distributed machine learning framework. Compared to existing frameworks, it has the following prominent features: *flexible data consistency model*, which lowers the communication and synchronization cost; *elastic scalability*, which enables adding new machines to the system without restarting the running framework; *continuous fault tolerance*, which prevents non-catastrophic machine failures to interrupt the overall computation process; and *an efficient implementation of vector clocks*, which ensures well-defined behavior after network failure.

This is joint work with David Andersen, Alex Smola, and Junwoo Park from CMU, together with Amr Ahmed, Vanja Josifovski, James Long, Eugene Shekita and Bor-Yiing Sufrom Google. Part of the work has been published in OSDI'14 [8].

MXNet is a multi-language library aiming to simplify the development of large scale deep neural network algorithms. It is developed based on Parameter Server. It outperforms many

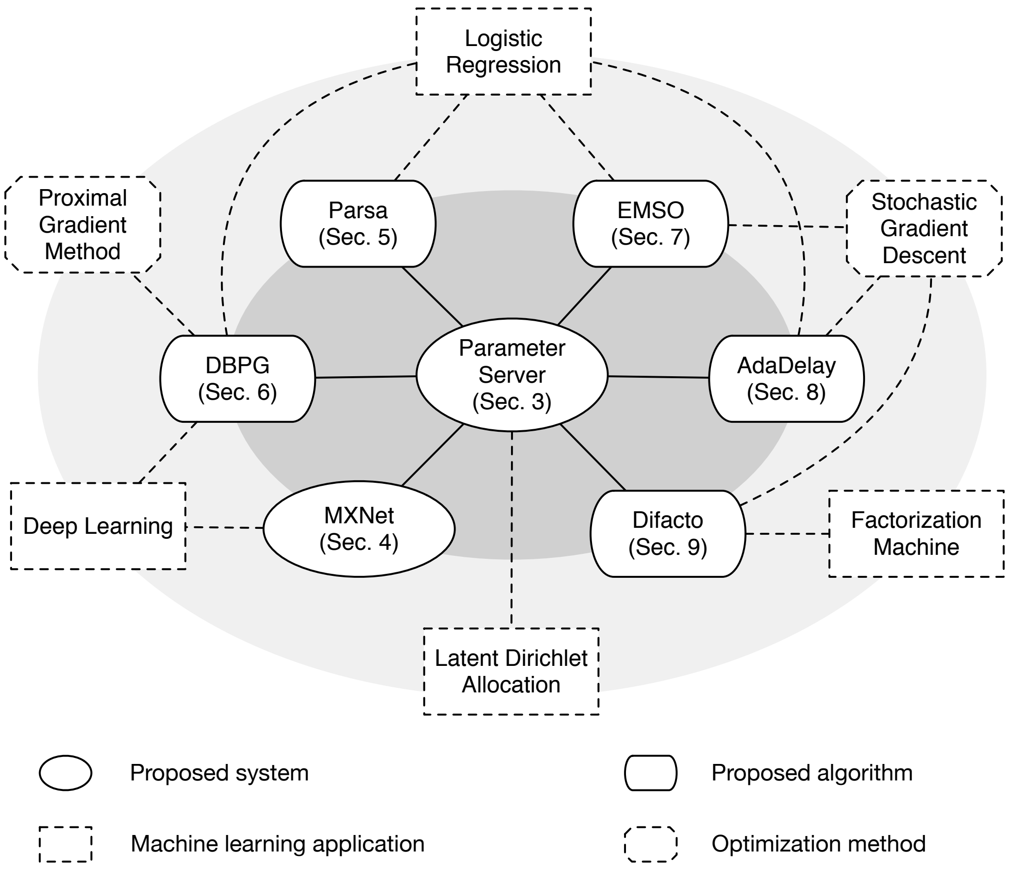

Figure 5: Connections of the sections in the thesis

existing platforms by exploiting the following features: *a mixed interface* with imperative programming and symbolic programming to achieve both flexibility and efficiency; a more general *asynchronous execution engine* that extends parameter server's asynchronous communication model to alleviate the data dependencies in deep neural networks; and more flexible ways for distributed deep learning for *hetogenous computing*.

This is joint work with a large number of collaborators from difference university and companies, including Tianqi Chen (U. Washington), Yutian Li (Standford), Min Li (NUS), Naiyan Wang (TuSimple), Minjie Wang (NYC), Tianjun Xiao (Microsoft), Bing Xu (Apple), Chiyuan Zhang (MIT), and Zheng Zhang (NYU Shanghai). Part of the work has been published in the learning system workshop on NIPS'16 [3].

Part 2 In the second part of the thesis, we present new distributed optimization algorithms for a couple of important machine learning problems. We implement the algorithms on either PS or MXNet and demonstrate that careful co-design of computing systems and optimization algorithms can greatly accelerate large scale distributed machine learning.

We first study the problem of using data partitioning to reduce the communication and synchronization cost. We formulate data placement as a submodular load-balancing problem and we propose a parallel partition algorithm named **Parsa** to solve it approximately. We show that with high probability the objective function is at least $n/\log(n)$ of the that with the best partition. The runtime of the algorithm is in the order of $\mathcal{O}(k|E|)$, where k is the number of partitions and $|E|$ is the number of edges in the graph.

This is joint work with David Andersen and Alexander Smola, and the results appeared in arXiv [11].

Then we study the problem of speeding up coordinate descent and stochastic gradient descent (SGD), which are widely used optimization methods in distributed computing environments.

To speed up coordinate descent We propose a new algorithm named **DBPG** based on the proximal gradient method to solve non-convex and non-smooth problems. It updates parameters in the blockwise style and allows delays between blocks to reduce the synchronization cost.

This is joint work with David Andersen, Alexander Smola and Kai Yu from Baidu. The results were published in NIPS'14 [9]

To speed up SGD in parallel computing, minibatch training has been used to reduce the communication cost, at the cost of a slower convergence rate. We propose a new minibatch training algorithm called **EMSO**. Instead of just running gradient descent with each minibatch, for each minibatch the algorithm solves an optimization with a conservatively regularized objective function. We show that this more efficient use of minibatches speeds up the convergence while maintaining low communication cost.

This is joint work with Alexander Smola together with Tong Zhang and Yuqiang Chen from Baidu. The results were published in KDD'14 [10]

To speed up asynchronous SGD, we propose a new algorithm **AdaDelay**, which allows the parameter updates to be sensitive to the actual delays experienced, rather than to worst-case bounds on the maximum delay. We show that this delay sensitive update rule leads to larger stepsizes, that can help gain rapid initial convergence without having to wait too long for slower machines, while maintaining the same asymptotic complexity.

This is joint work with Suvrit Sra, Adams Yu and Alex Smola, and the results were published in AISTATS'16 [16].

Finally we study a promising application for recommendation systems and polynomial generalized linear models—Factorization Machines. To make it scale to large amounts of data and large numbers of features, we propose a new algorithm **DiFacto**, which uses a refined Factorization Machine model with sparse memory adaptive constraints and frequency adaptive regularization.

This is joint work with Ziqi Liu, Alexander Smola, and Yu-Xiang Wang. The results were published in WSDM'16 [12].

5 Timeline to Completion

Oct 14th: Thesis committee meeting for proposal review

Oct 25th: Submit thesis draft to committee members for feedback

Nov 25th: Thesis Defense

Nov 30th: Final Thesis submission

References

[1] Amazon. Amazon web services. https://aws.amazon.com/. 1

[2] K. Canini. Sibyl: A system for large scale supervised machine learning. *Technical Talk*, 2012. URL `http://users.soe.ucsc.edu/~niejiazhong/slides/chandra.pdf`. 1, 2.1

[3] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015. 4

[4] Google. Google cloud. https://cloud.google.com/. 1

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`. 1, 2.1

[6] Larry Kim. How many ads does google serve in a day?, 2012. URL `http://goo.gl/oIidXO`. `http://goo.gl/oIidXO`. 2.1

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015. 1

[8] M. Li, D. G. Andersen, J. Park, A. J. Smola, A. Amhed, V. Josifovski, J. Long, E. Shekita, and B. Y. Su. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014. 4

[9] M. Li, D. G. Andersen, A. J. Smola, and K. Yu. Communication efficient distributed machine learning with the parameter server. In *Neural Information Processing Systems*, 2014. 4

[10] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014. 4

[11] Mu Li, Dave G Andersen, and Alexander J Smola. Graph partitioning via parallel submodular approximation to accelerate distributed machine learning. *arXiv preprint arXiv:1505.04636*, 2015. 4

[12] Mu Li, Ziqi Liu, Alexander J Smola, and Yu-Xiang Wang. Difacto: Distributed factorization machines. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 377–386. ACM, 2016. 4

[13] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela H Byers. Big data: The next frontier for innovation, competition, and productivity. 2011. 1

[14] Microsoft. Microsfot azure. https://azure.microsoft.com/. 1

[15] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951. 2.3

[16] Suvrit Sra, Adams Wei Yu, Mu Li, and Alexander J Smola. Adadelay: Delay adaptive distributed stochastic convex optimization. *arXiv preprint arXiv:1508.05003*, 2015. 4

[17] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL `http://arxiv.org/abs/1409.4842`. 2.1

[18] V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, New York, 1998. 2.3