METHOD: A General Optimization Framework for Fast, Scalable Machine Learning

Abhimanu Kumar*
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
abhimank@cs.cmu.edu

Alex Beutel*
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
abeutel@cs.cmu.edu

Qirong Ho
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
qho@cs.cmu.edu

Eric P. Xing
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
epxing@cs.cmu.edu

ABSTRACT

How can Facebook scalably model their billion-node social network? How can Twitter efficiently run a toipc model over billions of tweets? "Big data" today does not just mean more data to mine but also more things to understand - more users, more documents, and more images. Thus, modern machine learning and data mining faces the challenge of scaling our classic models to both use all of the data available as well as build larger and more complex models of our world.

In this paper we describe METHOD, a general framework for performing fast machine learning on huge models and big data. Our system uses a combination of recent insights in stochastic learning theory to enable our system innovations. In particular, our system offers three contributions: **(1) Scalability:** Our system distributes both the data and model across the cluster of machines, allowing the system to scale to terrabytes of data as well as models with billions of parameters, beyond the size that can fit in memory for each machine. **(2) Versatility:** Our system supports synchronized parameter updates, such as normalizing across the cluster. As a result, we can fit a variety of machine learning models, including topic models, dictionary learning, and mixed membership stochastic block models. **(3) Speed:** We use a modification of stochastic gradient descent, called Always-On SGD, to continously improve our model fit, even if their are slow workers that would otherwise obstruct model synchronization and slow computation. Finally, we demonstrate that with our new approah, METHOD can fit ML models at an unprecedented scale and faster than prior work.

1. INTRODUCTION

alex-comment: **NOTE: The paper outline is still *very* rough, and I expect to re-order many sections, largely in attempt to keep the focus on the system.**

2. MODEL AND DATA ABSTRACTION

In designing any large scale machine learning system, one of the most important design decisions is the abstraction for the data and problem model. Recent distributed ML systems focus primarily on the abstraction for the data, such as GraphLab's [?] view of each data point as a node. Here we take a different approach - partitioning our data and the model we are building of the data simultaneously. As we will go on to demonstrate, focusing on both is advantageous for both scalability and speed.

2.1 Block structure in relational data

For many classic machine learning problems, we have relational data between two sets of items, and we are asking questions about these items. To be a bit more concrete, consider the classic topic modeling question: given a large set of documents, with what probability is each document in a given set of topics, and with what probability does a word represent a given topic? In this case, a data point is the number of times a given word was used in a given document. Because of the intrinsic relational nature of the data (here between words and documents), partitioning our data intelligently results in a clean partitioning of our model of topics for the words and documents.

More specifically, for any given subset of the documents and subset of the words, there is a unique set of data points that are applicable to items from both sets. If we view our data as a very sparse matrix, partitioning the documents and the words results in the data matrix also being partitioned into blocks, as seen in Figure **??**. Under this partitioning, all data points in block $\mathcal{B}_{i,j}$ describe a relationship between

[*] The first two authors contributed equally to this work.

a word from set $\mathcal{S}_i^{(1)}$ and document from set $\mathcal{S}_j^{(2)}$. We focus our computation around these blocks of data.

An interesting property of these blocks is that for some blocks, such as $\mathcal{B}_{i,j}$ and $\mathcal{B}_{i',j'}$ where $i \neq i'$ and $j \neq j'$, we see that the blocks are *independent*. That is, a data point from $\mathcal{B}_{i,j}$ does not describe any relations to words in $\mathcal{S}_{i'}^{(1)}$ or documents in $\mathcal{S}_{j'}^{(2)}$ and vice versa for data in $\mathcal{B}_{i',j'}$. As has been shown in previous stochastic learning literature [**?**] and used in simpler data mining problems [**?**, **?**], this independence property allows for improved scalability and parallel processing. We will discuss our system and the strengths of this abstraction primarily in terms of topic modeling, but note the system generalizes to many machine learning problems as we will demonstrate later.

2.2 Distributing our data

As mentioned above, distributing your data over a cluster is useful, but as data grows partitioning both the data and your model of that data is increasingly valuable. For example, if we would like to know the topic distribution for 10 million documents over 1000 topics, this would require over 37 gigabytes just to hold all of the answers to the query. Therefore, it is crucial that as our data grows we intelligently distribute both the data and our solution. This significantly improve memory efficiency and makes it possible to scale to unprecedented sizes. Luckily, our blocking abstraction makes this easy.

In processing a given block $\mathcal{B}_{i,j}$, we only need the data from that block, and the current information about the words $\mathcal{S}_i^{(1)}$ and documents $\mathcal{S}_j^{(2)}$. Therefore, in distributing the problem over a cluster of machines, we can have each worker only store and process one block and its corresponding object model at a time. Additionally, because our sets of documents and sets of words are each disjoint, we can process our blocks in such a way that each document and each word is only being worked on by one worker at a time. As a result, we do not need to store *any* duplicate data (about the topic model or blocks) and thus we are perfectly memory efficient.

2.3 Parallel processing

The last piece of the general system design is understanding the order in which we process our data. Our goal is to reach the optimal memory efficiency described above and also keep our computation fast and accurate. To do this, we must choose a *stratum*, a group of blocks, to process in parallel. In order to be memory efficient and keep our computation accurate, each stratum must only contain blocks that are independent of each other. From the block structure, we can create multiple strata such that each block is in exactly one stratum. We iterate over the strata, in each case processing each block in the stratum in parallel on the cluster. We call processing one stratum a *subepoch* and processing all of the strata an *epoch*. A small example can be seen in Figure **??**.

In practice, each machine in the cluster holds one of the blocks being currently processed as well as the topic model for the corresponding words and documents. As explained, doing this results in the topic models and blocks being stored exactly once and thus being memory efficient. When a subepoch completes, we load the blocks for the next subepoch and transfer the necessary pieces of the topic model to the appropriate machines.

alex-comment: maybe give classic systems graphic of data moving between machines in cluster?

3. LEARNING

The model and data abstraction defined in section 2 is a convenient view for distributed computing of a large scale machine learning problem. This abstraction can be separated logically from the way the learning algortihm of the problem. The partition abstraction is used to distribute the learning phase over different computig units. For example in the LDA problem discussed in the abstraction earlier, we can use various learning algorithms to reach the solution. In particular we can use either of the two general learning schemes: 1) Sampling based learning, 2) Gradient based learning. These two learning schemes differ in the way they treat the given problem.

3.1 Sampling based learning scheme

Sampling based approach in general takes a probabilistic view of the problem. It learns the parameters of the model by approximating the probability distribution of the true data. This scheme samples iteratively from a changing distribution that is affected by the samples drawn. The distribution eventually reaches an equilibrium. The parameters of this stationary distribution are our solutions (parameters of the model) and are used further down the pipeline for prediction. In case of LDA such a scheme progresses iteratively. We start with some distribution of abhi-comment: explain the gibbs sampling steps for LDA here. For that we need to discribe the objective function for LDA in section 2.

3.2 Gradient based learning

In gradient based learning an objective function is optimized to find the parameters of the model. The objective is formulated in a way to abstract away the probabilistic components of the model if any present. A simple example of such a learning scheme is, in case of LDA abhi-comment: again define

3.2.1 Stochastic gradient descent

4. ANSWERING COMPLEX QUESTIONS

5. RUNNING OVER BARRIERS

6. IMPLEMENTATION

How do we implement this in Hadoop

7. APPLICATIONS

7.1 Latent dirichlet allocation (LDA)

7.2 Dictionary learning (DL)

7.3 Mixed membership stochastic block models (MMSB)

8. EXPERIMENTAL SETUP

To compare our METHOD to PSGD (data partition), DSGD+ (sync barrier) and GraphLab over speed of convergence and convergence quality we run them over a collection of large real world dataset. To further demonstrate the scalability of the approach we replicate and stack up real dataset to artificially create datasets of terabytes scale.

8.1 Dataset

We use four public datasets, two for LDA, NyTimes and PubMed [1], and one each for DL and MMSB, ImageNet[2] and TGraph[3] respectively.

8.1.0.1 NyTimes .

It is a collection of 300,000 Ny Times news articles that contain 102,660 distinct words and 100,000,000 tokens (word occurrences) in the.

8.1.0.2 PubMed .

This set contains 8,200,000 PubMed abstracts, that have in total 730,000,000 word occurrences and 141,043 unique words.

8.1.0.3 ImageNet .

This dataset was originally used for large scale visual recognition challenge in 2010 [1]. The set contains 1,261,406 images each with 1,000 features and has in total 389,080,708 non-zero pixels.

8.1.0.4 TGraph .

This is a follower network from twitter that stores directed edges from followers to followee [2]. It consists of 1,468,365,182 edges distributed among 41,652,230 vertices (users).

8.1.0.5 Scalable NyTimes (NyTimesN).

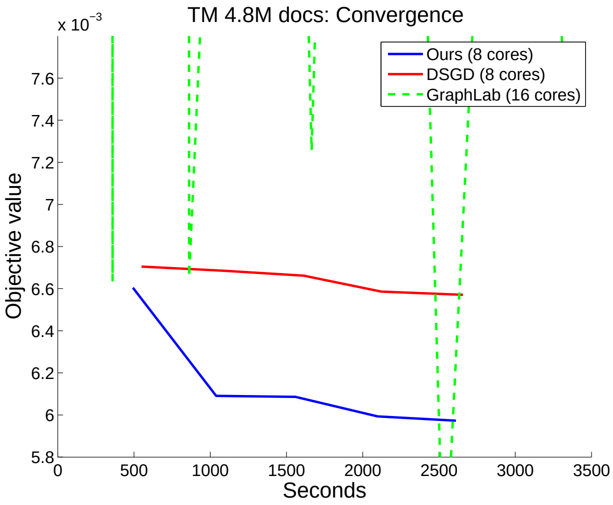
We replicate the documents in NyTimes to create a LDA datasets that are in scales of hundreds of gigabytes. For example NyTimes4 is a datset that has each news article in NyTimes dataset replicated 4 times. We create NyTimes4, NyTimes16, NyTimes64 and NyTimes256 that have 1,200,000, 4,800,000, 19,200,000 and 76,800,000 documents with data sizes 6.08, 25.12, 103.4, and 421.42 gbs respectively. Table 1 shows the concise statistics of the dataset used in all the experiments.

8.2 GraphLab based solver

We modify GraphLab's collaborative filtering toolkit to add the constraints defined in equation **??** abhi-comment: put in the constraints equation, section 7. We modify SGD based learner of the toolkit as it is easily pralleizable in the data spaceabhi-comment: Do we need to justify why we use SGD based solver for graphlab?. We use its public APIs (*transform_vertices(), periodic aggregator* and *map_reduce_vertices()*) to put normalization constraints. We take the simplest and most efficient way of normalizing accross vertices. A periodic aggregator is called after every fixed interval to compute the normalization factor using *map_reduce_vertices()* after which we apply the computed factor to each vertex using *transform_vertices()*.

Dataset	Dimensions	Nonzeros	Size(GB)
NyTimes	$0.3 * 10^6 \times 102{,}660$	$0.1 * 10^9$	1.49
PubMed	$8.2 * 10^6 \times 141{,}043$	$0.73 * 10^9$	11.19
ImageNet	$1.26 * 10^6 \times 1{,}000$	$0.39 * 10^9$	5.06
TGraph	$41.6 * 10^6 \times 41.6 * 10^6$	$1.5 * 10^9$	23.99
NyTimes4	$1, 2 * 10^6 \times 102{,}660$	$0.4 * 10^9$	6.08
NyTimes16	$4.8 * 10^6 \times 102{,}660$	$1.6 * 10^9$	25.12
NyTimes64	$19.2 * 10^6 \times 102{,}660$	$6.4 * 10^9$	103.4
NyTimes256	$76.8 * 10^6 \times 102{,}660$	$25.6 * 10^9$	421.42

Table 1: Dimension, size and nonzero statistics for different datasets. The exact figures are rounded off for simplicity. Size is the file size in gigabytes. The biggest dataset (NyTimes256) is of size approximately 0.5 terabytes.



[1] http://archive.ics.uci.edu/ml/datasets/Bag+of+Words

[2] http://www.image-net.org/challenges/LSVRC/2010/download-public

[3] http://konect.uni-koblenz.de/networks/twitter

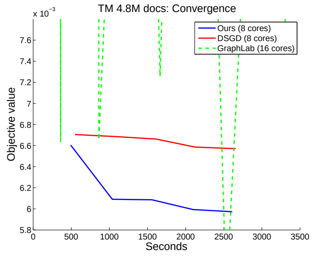
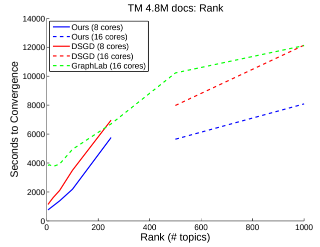
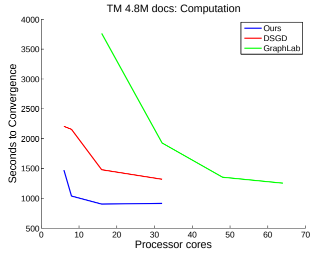
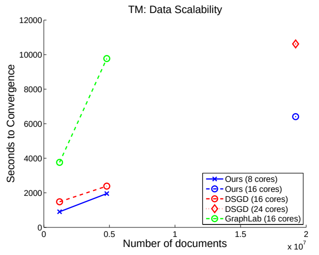
Topic Modeling			
Convergence Plots	# of Topics	# of Processors	# of Docs
			
Dictionary Learning			
Convergence Plots	# of Dictionary Bases	# of Processors	# of Images
TODO			
Mixed Membership Network Decomposition			
Convergence Plots	# of Network Roles	# of Processors	# of Network Nodes
TODO			

Figure 1: Convergence (Left) and scalability (in rank, processor cores and data size) of all methods, on topic modeling, dictionary learning and mixed-membership network decomposition. The convergence plot reveals the solution trajectory of each method, revealing pathological behavior such as oscillation. The scalability plots show how each method fares as the problem rank, number of processor cores, and data size is increased.

9. EVALUATION

9.1 Scalability

9.1.0.6 Dataset size.

data dimension

9.1.0.7 Model parameters.

rank

9.1.0.8 Processors.

machines, METHOD plateaus

9.2 Convergence speed

9.3 Convergence quality

GraphLab oscillation reasons and patterns, DSGD+ and PSGD converges to a poor quality,

9.4 Why METHOD wins

Put a plot of waiting times in different concstraints case and argue why it helps here.

10. RELATED WORK

11. CONCLUSION

12. REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[2] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *Proc. Int. World Wide Web Conf.*, pages 591–600, 2010.