Model Data-Centric Computation Abstractions in Machine Learning Applications

Bingjing Zhang
zhangbj@indiana.edu

Peng Bo
pengb@indiana.edu

Judy Qiu
xqiu@indiana.edu

School of Informatics and Computing
Indiana University
Bloomington, IN, USA

ABSTRACT

Each of the existing parallel machine learning applications provides one solution to improve model convergence speed during the training process. We further categorize them into four types of computation models, and based on these, we propose a new set of model data-centric computation abstractions which transform the parallel machine learning applications from training data-centric processing into model data-centric processing. The analysis uses LDA as an example, and the experiment results show that an efficient parallel model data update pipeline can achieve similar or higher model convergence speed compared with the related work.

Keywords

Machine Learning, Big Model Data, Model Computation

1. INTRODUCTION

Machine learning algorithms are a type of induction algorithm which can learn from training examples and make predictions through the derived model data [11]. During this period, the training data are repeatedly processed and the model data are iteratively updated. When applying machine learning algorithms on a large training dataset with big model data settings, the inherent challenge lies in that while training data are split among parallel workers, the model data which all local computations depend on remain extremely large and generate significant synchronization overhead.

In the past, when the model data were small enough to be held on one machine, classic collective communication methods were applied to synchronize model data. Though these operations are high-performance, the current big model data have exceeded their capabilities. In this situation many new solutions have been proposed to synchronize the tremendous model data efficiently. In this paper, we summarize these solutions into four computation models.

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Based on the computation models summarized, we propose a set of model data-centric computation abstractions to transform traditional training data-centric processing into model data-centric processing. Through adjusting the model data synchronization mechanisms and frequencies, we hope to answer the following four questions:

- **What** part of the model data needs to be updated?
- **When** should the model data update happen?
- **Where** should the model data update occur?
- **How** is the model data update performed?

With our new computation abstractions, we implement LDA and compare it with state-of-the-art implementations such as Yahoo! LDA [5] and Petuum LDA [3] on the "clueweb" dataset [1] with a total of 10 billion model parameters. We conduct performance experiments on a cluster of Intel Haswell architecture up to 100 nodes with a total of 4000 parallel threads. The results show that through an efficient parallel model data update pipeline, the new model data-centric computation abstractions can achieve similar or faster model convergence speed compared with other competitors.

The following sections describe: a survey of computation models (Section 2), model data-centric computation model (Section 3), experiments (Section 4), and conclusions (Section 5).

2. COMPUTATION MODEL SURVEY

In this section we use LDA application to demonstrate the difference between computation models. LDA can be viewed as a process of decomposing a word-document matrix into one word-topic matrix and another document-topic matrix. Collapsed Gibbs Sampling [16] is a Markov chain Monte Carlo (MCMC) type inference algorithm for LDA topic modeling and shows high scalability in parallelization [14, 19], In the "initialize" phase, each training point (a word token in training document dataset), is assigned to a random topic denoted as z_{ij}. It then begins to reassign topics to each token $x_{ij} = w$ by sampling from a multinomial distribution of a conditional probability of z_{ij}:

$$p\left(z_{ij} = k \mid z^{\neg ij}, x, \alpha, \beta\right) \propto \frac{N_{wk}^{\neg ij} + \beta}{\sum_w N_{wk}^{\neg ij} + V\beta} \left(M_{kj}^{\neg ij} + \alpha\right)$$

Here superscript $\neg ij$ means that the corresponding token is excluded. V is the vocabulary size. N_{wk} is the token count of word w assigned to topic k in K topics, and M_{kj} is the token count of topic k assigned in document j. The matrices

Z_{ij}, N_{wk} and M_{kj}, are the model data. Hyperparameters α and β control the topic density in the final model output. The model data gradually converge during the process of iterative sampling. This is the phase where the "burn-in" stage occurs and finally reaches the "stationary" stage. In real LDA trainers, SparseLDA [20] algorithm is often used as an optimized CGS implementation.

2.1 Computation Model Attributes

Worker Each parallel unit in a computation model is called a "worker". In a parallel implementation, there are usually two levels of parallelism: I. Distributed Environment and II. Multi-thread Environment. In Level I each worker is a process and the workers are synchronized through network communication. In Level II, the workers are threads and are coordinated through thread synchronization mechanisms.

Model Data LDA model data contains four parts: I. Z_{ij} - topic assignments on tokens, II. $\sum_w N_{wk}^{\neg ij}$ - topics' token counts, III. N_{wk} - words' token counts on topics, IV. M_{kj} - documents' token counts on topics. Part I is stored along with the training tokens. Part II is always shared between workers. When Part III is stored in the local, Part IV is shared between workers, and vice versa.

Synchronized/Asynchronous Algorithm Computation models can be divided into those with synchronized algorithms and others with asynchronous algorithms. In synchronized algorithms, the computation progress on one worker depends on the progress on other workers; asynchronous algorithms lack this dependency.

The Latest/Stale Model Data Computation models may use either the latest values or stale values from the shared model data. Here "latest" means the current model data used in the computation is up-to-date and not modified simultaneously by other workers, while "stale" means the values in the model data are old. Since the computation model using the latest model data maintains the model consistency, its model output contains less approximation and is close to the result of the sequential algorithm. In LDA, the consistency of Model Data Part III and Part IV are the most relevant. Model Data Part II is commonly stale because the sums of token counts on topics are very large and thus some alternation on these values is hardly noticeable.

2.2 The Types of Computation Models

LDA implementations are categories to four types of computation models, each of which uses a different means to handle the model data and coordinate workers (see Fig. 1). The computation model description focuses on the distributed environment in which Model Data Part III is chosen to be shared between workers. However computation models can be applied to a multi-thread environment. In a system with two levels of parallelism, model composition is commonly adapted, with one type of model at Level I and another at Level II.

Computation Model A This computation model uses a synchronized algorithm to coordinate parallel workers. In each iteration, once a worker samples a token, it locks a word's model data and excludes access from other workers. When the sampling is done and the related model data is updated, the worker unlocks it. As long as workers compute and update on different model data, they can execute in parallel. Because only one worker is allowed to access one word's model data each time, the model data used in the local computation is always the latest. In real implementations, due to the high overhead of locking, this computation model is seldom applied.

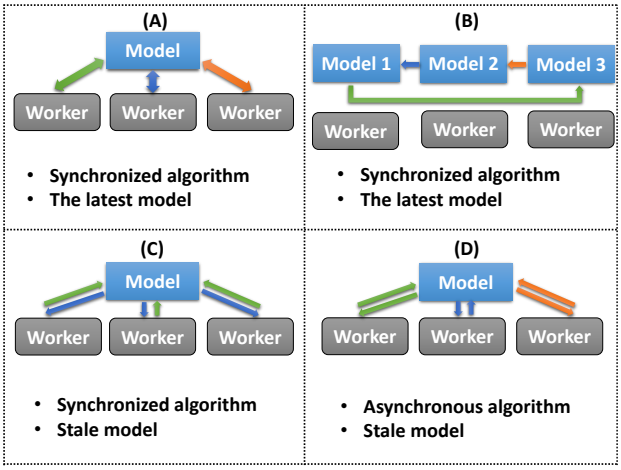

Figure 1: The Types of Computation Models

Computation Model B The next computation model also has a synchronized algorithm. Each worker first takes a partition of the shared model data and performs sampling. Afterwards the model is shifted between workers. When all the model partitions are accessed by all the workers, an iteration is complete. Through model rotation, each word's model data is computed and updated only by one worker at a time so that the consistency of the model data is maintained.

Computation Model C Computation model C uses a synchronized algorithm but with stale model data. In a single iteration, each worker firstly fetches all the model data required by the local computation. If the local model is too large to fit in memory, local computation can be split to stages where each time a part of the model data is fetched and computed. When the local computation is done, the modifications of local model data from all the workers are gathered to update the model data.

Computation Model D With this model, an asynchronous algorithm employs stale model data. Here each worker independently fetches the related model data to local, performs the local computation and returns the model data modifications. Unlike Computation Model A, during the period of local computation, other workers are allowed to fetch or update the same word's model data. As opposed to Computation Model B and C, there is no synchronization point in this computation model.

2.3 Discussions

Initial research shows many machine learning algorithms can be implemented in MapReduce system [9] and later model data communication is improved by the collective communication operations in iterative MapReduce [15, 21, 8]. However, this solution is only one special case in Computation Model C and it is not scalable as the model size grows larger than the capacity of the local memory. As modern model data may reach $10^{10} \sim 10^{12}$ parameters, later, Parameter Server type solutions [13, 5, 17, 6] store model data to a set of server machines and use Computation Model D to reduce communication overhead, but Petuum shows model synchronization is important to model convergence

Table 1: LDA CGS Implementations

Implementation Name	Algorithm	Computation Model
PLDA [18]	CGS	C
PowerGraph LDA [4]	CGS	C
Yahoo! LDA [5]	SparseLDA	D
Peacock [19]	SparseLDA	D & B
Parameter Server [13]	CGS, etc.	D
Petuum 0.93 [10]	SparseLDA	D
Petuum 1.1 [12]	SparseLDA	B & D

and a computation model mixed with C and D proved to have better performance [10]. Furthermore, Petuum also implements Computation Model B [3, 12] which shows higher model convergence speed with the latest model data. We show the computation models used in each LDA implementation in Table 1. Model composition also occurs in Peacock [19] and Petuum 1.1 [3], in which one computation model is used in the distributed environment and another is used in the multi-thread environment.

3. MODEL DATA-CENTRIC COMPUTATION ABSTRACTIONS

Though each solution promotes a certain computation model, their effectiveness is not well studied. To improve the model update rate and increase the model converge speed, the parallelization of machine learning applications should be turned from training data-centric processing to model data-centric processing. To build an efficient parallel model data update pipeline, we derive a new set of model data-centric computation abstractions from the computation models. These abstractions contain model data abstractions and APIs for model synchronization.

We split model data into partitions and use the concept "table" to associate data partitions on different workers and form a complete model dataset. For small model data, traditional collective communication APIs such as "allgather" and "allreduce" are used to synchronize model data copies on all the workers efficiently. For large model datasets which cannot be held in the memory of one machine, two types of data abstractions are defined: the global table and the local table. In global tables, each partition has a unique ID and represents a part of the whole distributed model dataset; but in local tables, partitions on different workers can share the same partition ID. Each of these partitions sharing the same ID is considered a local version of a partition in the full distributed model dataset.

We define three model synchronization APIs. The first two are paired operations following Computation Model C. "syncGlobalWithLocal" reduces the model data partitions from local tables to the global table and "syncLocalWithGlobal" redistributes the model data partition in the global table to local tables. Routing optimized broadcasting is used if some partitions are required by all the workers. Lastly, "rotateGlobal" follows Computation Model B. It considers workers in a ring topology and shifts the partitions in the global table between neighbors. When the operation is complete, each worker will hold a different set of partitions. Since each worker only talks to its neighbors, "rotateGlobal" can transmit global data in parallel without any network conflicts.

Here we simply discuss the applicability of the model data computation abstractions based on the computation dependency between the parallel workers and the model data. The computation dependency can be represented as a matrix, where each row signifies a worker, each column represents a global model data partition and each element shows the requirements of the partition in the local computation. Based on the density of this matrix, we can choose proper operations in different applications. If the matrix is dense, we suggest using the "rotateGlobal" operation. If the matrix is sparse, using "syncGlobalWithLocal" and "syncLocalWithGlobal" is a superior solution.

The new abstractions allow developers to handle model data synchronization in a convenient way and program iterative machine learning algorithms productively. Many algorithm kernels and applications can be supported in the new abstractions, including:

- **Expectation-Maximization Type**
  - K-Means Clustering
  - CVB (Collapsed Variational Bayesian) for topic modeling (LDA)
- **Gradient Optimization Type**
  - SGD (Stochastic Gradient Descent) and CCD (Cyclic Coordinate Descent) for classification (SVM, Logistic Regression), regression (LASSO), Collaborative Filtering (Matrix Factorization)
- **Markov Chain Monte Carlo Type**
  - CGS (Collapsed Gibbs sampling) for topic modeling (LDA)

In the future, given access to new abstractions, we can provide initial answers to the four questions "what", "when", "where" and "how" in model data-centric computations. The new operation APIs allow us to adjust the mechanisms and the frequencies of the model synchronization. By exploiting the sparsity of computation dependency and optimizing routing topology, the new abstractions can benefit many machine learning applications.

4. EXPERIMENTS

LDA experiments are done on Juliet cluster [2] which contains 32 nodes each with two 18-core 36-thread Xeon E5-2699 processors and 96 nodes each with two 12-core 24-thread Xeon E5-2670 processors. In the experiments, 31 nodes with Xeon E5-2699 and 69 nodes with Xeon E5-2670 are used to form a cluster of 100 nodes, each with 40 threads and 128GB memory. All the tests are done with Infiniband through IPoIB support.

We use the "clueweb" dataset to test the effectiveness of the new model data-centric computation abstractions. "clueweb" contains 50.5 million documents and 12.48 billion tokens. Model Data Part III (word's model data) is shared between workers. It contains 1 million words, each with 10 thousand topics, 10 billion parameters in total. The initial model size is about 14.7GB. Hyperparameters α and β are both set to 0.01. With the new abstractions, two LDA applications are developed. One is "rtt", which follows Computation Model B in Parallelism Level I and Computation Model D in Level II. Another is "lgs", which follows Computation Model C in Level I and Computation Model D in

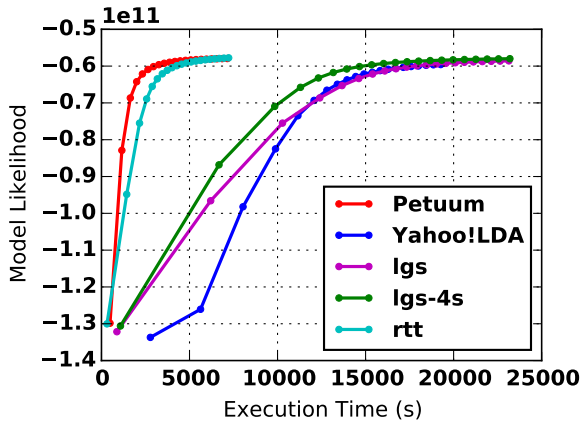

Figure 2: Model Convergence Speed on "clueweb"

Level II. Model Data Part II is simply synchronized with "allreduce" operation per iteration in two implementations.

The performance results are in Fig. 2. Both "rtt" and Petuum use the same set of computation models and achieve similar model convergence speed. They are remarkably faster than the rest. "lgs" proves faster than Yahoo! LDA at the beginning, but later their model convergence speed lines tend to overlap. Through adjusting the number of model synchronization frequencies to 4 per iteration, "lgs-4s" exceeds Yahoo! LDA from start to finish.

5. CONCLUSIONS

The paper summarizes existing parallel solutions to machine learning applications into four computation models, and then provides a new set of model data-centric computation abstractions which turn the view of machine learning application parallelization from training data-centric processing to model data-centric processing. The initial experiments show that the two LDA applications developed under the model data-centric computation abstractions can achieve similar or even faster model convergence speed compared with state-of-the-art implementations. In the future, by adjusting model update mechanisms and frequencies, we aim to build an efficient parallel model update pipeline with high performance model convergence speed.

6. ACKNOWLEDGMENTS

We gratefully acknowledge support from Intel Parallel Computing Center (IPCC) Grant, NSF 1443054 CIF21 DIBBs 1443054 Grant, and NSF OCI 1149432 CAREER Grant. We appreciate the system support offered by FutureSystems.

7. REFERENCES

[1] clueweb. http://lemurproject.org/clueweb09/.

[2] FutureSystems. https://portal.futuresystems.org.

[3] Petuum LDA. https://github.com/petuum/bosen/wiki/Latent-Dirichlet-Allocation.

[4] PowerGraph LDA. https://github.com/dato-code/PowerGraph/blob/master/toolkits/topic_modeling/topic_modeling.dox.

[5] Yahoo! LDA. https://github.com/sudar/Yahoo_LDA.

[6] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. Smola. Scalable inference in latent variable models. In *WSDM*, pages 123–132, 2012.

[7] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–102, 2003.

[8] M. Chowdhury, M. Zaharia, J. Ma, M. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. *ACM SIGCOMM Computer Communication Review*, 41(4):98–109, 2011.

[9] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, volume 19, page 281, 2007.

[10] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. Gibbons, G. Gibson, G. Ganger, and E. Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*, pages 1223–1231, 2013.

[11] R. Kohavi and F. Provost. Glossary of terms. http://ai.stanford.edu/~ronnyk/glossary.html.

[12] S. Lee, J. K. Kim, X. Zheng, Q. Ho, G. Gibson, and E. Xing. On model parallelization and scheduling strategies for distributed machine learning. In *NIPS*, pages 2834–2842, 2014.

[13] M. Li, D. Andersen, J. W. Park, A. Smola, A. Ahmed, V. Josifovski, J. Long, E. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *OSDI*, pages 583–598, 2014.

[14] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed algorithms for topic models. *The Journal of Machine Learning Research*, 10:1801–1828, 2009.

[15] J. Qiu and B. Zhang. Mammoth data in the cloud: clustering social images. In *Clouds, Grids and Big Data*, Advances in Parallel Computing. IOS Press, 2013.

[16] P. Resnik and E. Hardist. Gibbs sampling for the uninitiated. Technical report, University of Maryland, 2010.

[17] A. Smola and S. Narayanamurthy. An architecture for parallel topic models. In *VLDB*, volume 3, pages 703–710, 2010.

[18] Y. Wang, H. Bai, M. Stanton, W.-Y. Chen, and E. Y. Chang. PLDA: parallel latent dirichlet allocation for large-scale applications. *Algorithmic Aspects in Information and Management*, pages 301–314, 2009.

[19] Y. Wang, X. Zhao, Z. Sun, H. Yan, L. Wang, Z. Jin, L. Wang, Y. Gao, C. Law, and J. Zeng. Peacock: learning long-tail topic features for industrial applications. *ACM Transactions on Intelligent Systems and Technology*, 6(4), 2015.

[20] L. Yao, D. Mimno, and A. McCallum. Efficient methods for topic model inference on streaming document collections. In *KDD*, pages 937–946, 2009.

[21] B. Zhang and J. Qiu. High performance clustering of social images in a map-collective programming model. In *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013.