Model-Centric Computation Abstractions in Machine Learning Applications

Bingjing Zhang
zhangbj@indiana.edu

Bo Peng
pengb@indiana.edu

Judy Qiu
xqiu@indiana.edu

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

ABSTRACT

We categorize parallel machine learning applications into four types of computation models and propose a new set of model-centric computation abstractions. This work sets up parallel machine learning as a combination of training data-centric and model parameter-centric processing. The analysis uses Latent Dirichlet Allocation (LDA) as an example, and experimental results show that an efficient parallel model update pipeline can achieve similar or higher model convergence speed compared with other work.

CCS Concepts

•**Information systems** → *Data analytics;* •**Theory of computation** → *Parallel computing models;*

Keywords

Machine Learning, Big Model, Model Computation

1. INTRODUCTION

Machine learning algorithms typically learn from training examples and make predictions through derived model parameters[1]. During this period, the training data are repeatedly processed, and the model parameters are iteratively updated. When applying machine learning algorithms on a large training dataset with a large number of model parameters, the inherent challenge lies in that while training data are split among parallel workers, the number of model parameters, which all local computations depend on, remain extremely large. As a result, communicating these model parameters from their home storage can generate significant synchronization overhead.

In some cases, when the total number of model parameters is small enough that the full model can be held on one machine, classic collective communication methods are applied to synchronize model parameters. Though these operations are high-performance, some current big model problems have exceeded their capabilities. Therefore, new solutions have been proposed to synchronize big models more efficiently. In this paper, we categorize these solutions into four computation models.

Based on our categorization of computation models, we propose a set of abstractions to enhance traditional data-centric processing with sophisticated model-centric processing. By adjusting the model synchronization mechanisms and frequencies, we aim to answer the following four questions:

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

With our new computation abstractions, we implement Latent Dirichlet Allocation (LDA) [5] and compare it with state-of-the-art implementations such as Yahoo! LDA [3] and Petuum LDA [1] on the "clueweb" dataset[2] with a total of 10 billion model parameters. We conduct performance experiments on a cluster of Intel Haswell architecture with up to 100 nodes and a total of 4000 parallel threads. The results show that through an efficient parallel model update pipeline, the new model-centric computation abstractions can achieve similar or faster model convergence speed compared with other approaches.

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

2. COMPUTATION MODEL SURVEY

In this section, we use LDA to demonstrate the differences among 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 [13] is a Markov chain Monte Carlo type inference algorithm for LDA topic modeling and shows high scalability in parallelization [11, 16], In the "initialize" phase, each training token, 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

[1] http://ai.stanford.edu/~ronnyk/glossary.html

[2] 10% of ClueWeb09 (a collection of English web pages, http://lemurproject.org/clueweb09/)

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 $\neg ij$ refers to the corresponding token 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. Hyperparameters α and β control the topic density in the final model output. The model gradually converges during the process of iterative sampling. This is the phase where "burn-in" occurs and finally reaches the "stationary" stage. In real LDA trainers, the SparseLDA [17] algorithm is often used as an optimized CGS implementation.

2.1 Computation Model Attributes

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

Model The LDA model 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 locally, 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 Computation models may use either the latest values or stale values from the shared model. The "latest" means that the current model used in computation is up-to-date and not modified simultaneously by other workers, while the "stale" indicates the values in the model are old. Since the computation model using the latest model maintains model consistency, its model output contains less approximation and is close to the output of the sequential algorithm. In LDA, the consistency of Model Part III and Part IV are the most relevant. Model Part II is commonly stale because the sum of token counts per topic is very large, thus changes on these values are hardly noticeable.

2.2 Types of Computation Models

LDA implementations can be classified into four types of computation models, each of which uses a different means to handle the model and coordinate workers (see Fig. 1). The computation model description focuses on the distributed environment, in which Model Part III is chosen to be shared between workers. However, computation models can also be applied to a multi-thread environment. In a system with two forms of parallelism, model composition is commonly adapted, with one type of model at Form I and another at Form 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 parameters and prevents other workers from accessing them. When the sampling is performed and the related model parameters are updated, the worker unlocks the parameters. As long as workers compute and update on different model parameters, they can execute in parallel. Only one worker is allowed to access a word's model parameters at a time; therefore the model parameters used in local computation is always the latest. In practice, this computation model is seldom applied due to the high overhead of locking.

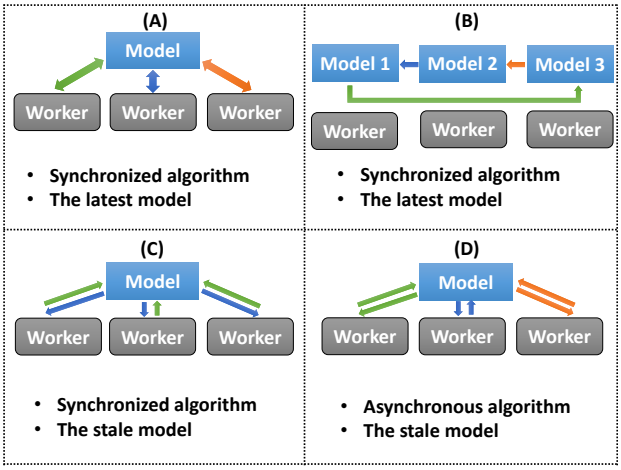

Figure 1: Types of Computation Models

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

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

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

2.3 Discussion

Previous work shows many machine learning algorithms can be implemented in the MapReduce paradigm [7]; later on, model communication is improved by collective communication operations in iterative MapReduce [12, 19, 6]. How-

Table 1: LDA CGS Implementations

Implementation	Algorithm	Computation Model
PLDA [15]	CGS	C
PowerGraph LDA [2]	CGS	C
Yahoo! LDA [3]	SparseLDA	D
Peacock [16]	SparseLDA	D & B
Parameter Server [10]	CGS, etc.	D
Petuum 0.93 [8]	SparseLDA	D
Petuum 1.1 [1]	SparseLDA	B & D

ever, the solution is a special case in Computation Model C and is not immediately scalable as the model size grows larger than the capacity of the local memory. As current models may reach $10^{10} \sim 10^{12}$ parameters, Parameter Server type solutions [10, 3, 14, 4] store the model on a set of server machines and use Computation Model D to reduce communication overhead. Petuum, however, shows model synchronization is important to model convergence, and a computation model mixed with C and D is proved to have better performance [8]. Furthermore, Petuum implements Computation Model B [1, 9], which shows higher model convergence speed with the latest model.

We show the computation models used in each LDA implementation in Table 1. Model composition also occurs in Peacock [16] and Petuum 1.1 [1], in which one computation model is used in the distributed environment and another in the multi-thread environment.

3. MODEL-CENTRIC ABSTRACTIONS

Although individual solutions utilize a particular computation model, the effectiveness of these solutions is not well studied. To improve model update rate and increase model convergence speed, parallelization of machine learning applications should concentrate more on model-centric processing. In order to build an efficient parallel model update pipeline, we derive a new set of model-centric computation abstractions from the computation models. These abstractions contain model abstractions and APIs for model synchronization.

We split model parameters into partitions and use the concept "table" to associate partitions on different workers and form a complete model. For a small model, traditional collective communication APIs such as "broadcast", "reduce", "allgather", and "allreduce" are used to efficiently synchronize model copies on all the workers.

For a large model which cannot be held in the memory of one machine, two types of model 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; 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. In addition, we define three new model synchronization APIs. The first two operations are paired. "syncGlobalWithLocal" reduces the model partitions from local tables to the global table, and "syncLocalWithGlobal" redistributes the model partition in the global table to local tables. Routing optimized broadcasting is used if some partitions are required by all the workers. Lastly, "rotateGlobal" 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 communicates to its neighbors, "rotateGlobal" can transmit global data in parallel without any network conflicts.

Here we briefly discuss the applicability of model-centric computation abstractions based on the computation dependency between parallel workers and the model. The computation dependency can be represented as a matrix, where each row signifies a worker, each column represents a model partition, and each element shows the requirements of the partition in 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 enable developers to handle complex model synchronization and program iterative machine learning algorithms more productively. Many algorithm kernels and applications can be supported in the new abstractions, including but not limited to:

- **Expectation-Maximization Type**
  - K-Means Clustering
  - Collapsed Variational Bayesian for topic modeling (e.g. LDA)
- **Gradient Optimization Type**
  - Stochastic Gradient Descent and Cyclic Coordinate Descent for classification (e.g. SVM and Logistic Regression), regression (e.g. LASSO), collaborative filtering (e.g. Matrix Factorization)
- **Markov Chain Monte Carlo Type**
  - Collapsed Gibbs Sampling for topic modeling (e.g. LDA)

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

4. EXPERIMENTS

The LDA experiments are run on the Juliet cluster[3] which contains 32 nodes each with two 18-core 36-thread Intel Xeon E5-2699 processors and 96 nodes each with two 12-core 24-thread Intel Xeon E5-2670 processors. In the experiments, 31 nodes with E5-2699 and 69 nodes with E5-2670 are used to form a cluster of 100 nodes, each with 40 threads and 128GB memory. All the tests are performed with Infiniband through IPoIB protocol.

We use the "clueweb" dataset to test the effectiveness of the new model-centric computation abstractions. "clueweb" contains 50.5 million documents and 12.48 billion tokens. Model Part III (words' model parameters) is shared between workers. It contains 1 million words, each with 10 thousand topics, resulting in a total of 10 billion parameters. The

[3] https://portal.futuresystems.org

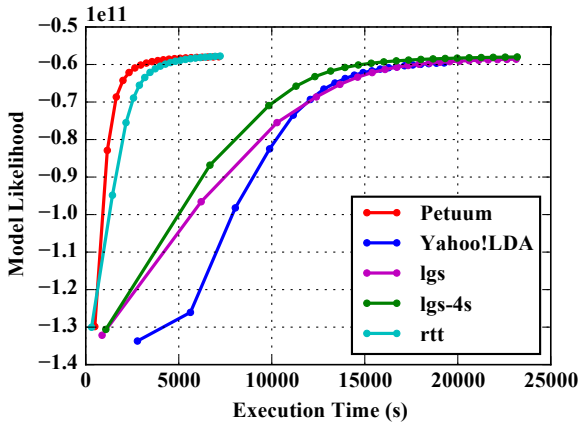

Figure 2: Model Convergence Speed on "clueweb"

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 on top of Harp [20]. One is "rtt", which follows Computation Model B (uses "rotateGlobal") in Parallelism Form I and Computation Model D in Form II. Another is "lgs", which follows Computation Model C (uses "syncGlobalWithLocal" and "syncLocalWithGlobal") in Form I and Computation Model D in Form II. Model Part II is simply synchronized with the "allreduce" operation per iteration in two implementations. Additional details of the experiments are reported elsewhere [18].

The performance results are presented 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 other implementations. "lgs" proves to be faster than Yahoo! LDA at the beginning, but at later stages, their model convergence speed tends 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

This paper investigates existing parallel solutions of machine learning applications and illustrates four computation models. This provides a new set of abstractions which enhance model-centric processing for parallelization of machine learning applications. Experimental results show that the two LDA applications developed under model-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 expect to build an efficient general parallel model update pipeline with high 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] Petuum LDA. https://github.com/petuum/bosen/wiki/Latent-Dirichlet-Allocation.

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

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

[4] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. Smola. Scalable Inference in Latent Variable Models. In *WSDM*, 2012.

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

[6] 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.

[7] 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*, 2007.

[8] 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 *NIPS*, 2013.

[9] 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*, 2014.

[10] 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*, 2014.

[11] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed Algorithms for Topic Models. *The Journal of Machine Learning Research*, 10:1801–1828, 2009.

[12] J. Qiu and B. Zhang. Mammoth Data in the Cloud: Clustering Social Images. *Cloud Computing and Big Data*, 23:231, 2013.

[13] P. Resnik and E. Hardist. Gibbs Sampling for the Uninitiated. Technical report, University of Maryland, 2010.

[14] A. Smola and S. Narayanamurthy. An Architecture for Parallel Topic Models. *VLDB*, 3(1-2):703–710, 2010.

[15] Y. Wang, H. Bai, M. Stanton, W.-Y. Chen, and E. Chang. PLDA: Parallel Latent Dirichlet Allocation for Large-Scale Applications. In *AAIM*, pages 301–314. Springer, 2009.

[16] 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 TIST*, 6(4):47, 2015.

[17] L. Yao, D. Mimno, and A. McCallum. Efficient Methods for Topic Model Inference on Streaming Document Collections. In *KDD*, 2009.

[18] B. Zhang, B. Peng, and J. Qiu. High Performance LDA through Collective Model Communication Optimization. In *ICCS*, 2016.

[19] B. Zhang and J. Qiu. High Performance Clustering of Social Images in a Map-Collective Programming Model. In *SoCC*, 2013.

[20] B. Zhang, Y. Ruan, and J. Qiu. Harp: Collective Communication on Hadoop. In *IC2E*, 2015.