Distributed LDA on Harp

Ethan Li
School of Informatics and Computing
107 S. Indiana Avenue
Bloomington, Indiana
li526@indiana.edu

Rohit Patil
School of Informatics and Computing
107 S. Indiana Avenue
Bloomington, Indiana
rnpatil@umail.iu.edu

ABSTRACT

Harp LDA is a distributed variational bayes inference (VB) algorithm for LDA model which would be able to model a large and continuously expanding dataset using Harp collective communication library. We also study how variational bayes inference converges within Map-Collective jobs provided by Harp. The proposed algorithm is run against a corpus of Wikipedia Dataset to find if it can achieve better performance and shorter time and memory requirements. We also provide experimental procedures to evaluate execution accomplishment of our proposed algorithm in terms of scalability and log-likelihood metrics.

CCS Concepts

•Computer systems organization → Distributed systems;

Keywords

LDA; Harp; Distributed algorithm; Scalability

1. INTRODUCTION

Latent Dirichlet Allocation (LDA)[1], a topic modeling algorithm designed for text documents in unsupervised environments. It is especially useful for web content such as on-line news articles, blogs along with data from major domains like Google, Yahoo, and Amazon etc. Suppose you have a very large corpus and want to come up with some inference about the data, topic models is a mechanism to get a corpus-level view of major themes. LDA assumes that the documents in a corpus are generated from a set of K topics. Thus from an input corpus and number of topics K, LDA model maps words in documents to topics. Even though the sequential model is theoretically effective, a significant drawback evident while using LDA is the amount of time taken and memory requirements for inference while dealing with a very large scaled and dynamically expanding corpus. Moreover the huge data-sets wont fit on a single machine. Thus a distributed multiprocessor system framework is essential for solving topic modelling LDA inference for a large data corpus. If suppose we have a distributed framework with will N processor, for distributed computation each of N processors deals with only 1/N of the total data set and the inference results of each processors are aggregated to generate the final results.

Approximate inference for topic modelling using LDA can be carried out using a number of different approaches such as variational bayes methods, expectation maximization and Markov chain Monte Carlo methods. An approach discussed in Mr. LDA[4] devised a parallel LDA algorithm in the Apache Hadoop framework[3] using MapReduce programming framework. Mr. LDA approach also emphasized on positives of implementing variational inference over using Gibbs sampling (Markov chain Monte Carlo) in terms of extensibility, flexibility, and number of iterations and thus eventually affecting the scalability. Our proposed LDA model which is based on Harp[5] collective communication library is similar to MapReduce framework and uses Variational Bayes (VB) inference. Although, MapReduce framework can provide better communication between multiple processors and high reliability systems, to might suffer from I/O cost to large number of HDFS read/write operations. Harp library can be plugged into Hadoop runtime to enable efficient in-memory communication to avoid HDFS read/write thus saving excess I/O cost. Moreover, other major plausible improvements of Harp comprises hierarchal data abstraction, collective communication model, pool based memory management, BSP style computation parallelism etc. A better understanding of our algorithm can be developed in the sections proposed methods and experimental results.

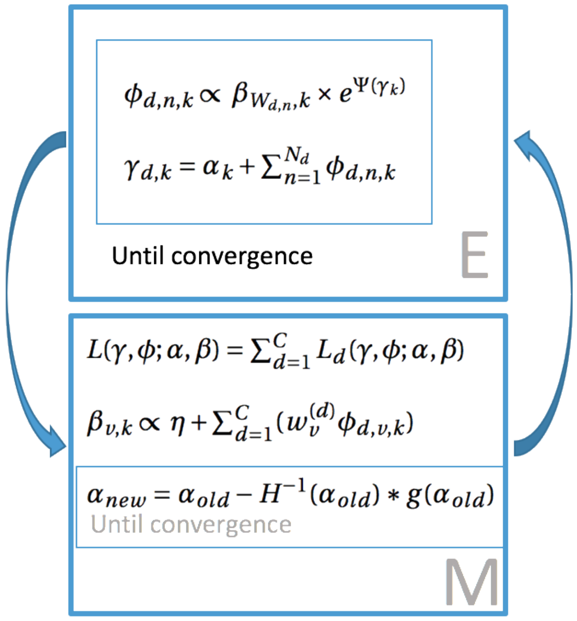
2. PROBLEM DEFINITION

LDA is a powerful topic modelling algorithm for clustering words into topics and documents into mixtures of topics. Even though the sequential LDA model is theoretically effective, a significant drawback evident while using LDA is the amount of time taken and memory requirements for inference while dealing with a very large scaled and dynamically expanding corpus. Moreover the huge data-sets won't fit on a single machine. Thus a distributed multiprocessor system framework is essential for solving topic modelling LDA inference for a large data corpus as an efficient way of distributing the computation across multiple machines. The primary aim of the project is to build a scalable topic modelling tool for a large corpus of textual data devised by implementing LDA model in a distributed environment. We

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

follow the Mr. LDA to implement distributed variational bayes inference LDA on Harp. Harp modules particularly of our interests are dynamic scheduler, all-reduce and push-pull communication models.

3. PROPOSED METHOD



$\phi_{d,n,k} \propto \beta_{W_{d,n},k} \times e^{\Psi(\gamma_k)}$
$\gamma_{d,k} = \alpha_k + \sum_{n=1}^{N_d} \phi_{d,n,k}$
Until convergence
E
$L(\gamma,\phi;\alpha,\beta) = \sum_{d=1}^{C} L_d(\gamma,\phi;\alpha,\beta)$
$\beta_{v,k} \propto \eta + \sum_{d=1}^{C} (w_v^{(d)} \phi_{d,v,k})$
$\alpha_{new} = \alpha_{old} - H^{-1}(\alpha_{old}) * g(\alpha_{old})$
Until convergence
M

Figure 1: algorithm

We propose a distributed LDA algorithm using Harp. First of all, the data will be preprocessed to be a D $\times$ V matrix M, where D is the number of documents in the corpus and V is the size of vocabulary. The element $M_{d,v}$ in the matrix is the number of tokens of the word v in the document d. Then the data will be partitioned to P partitions. Each partition contains D/P documents. P is the number of mappers. For each mapper, it will compute ϕ, γ, α, β using VB method. The collective communication methods like *allreduce* and *push-pull* will be used to synchronize α, β and likelihood. The main pseudocode is as follows.

```
Pseudocode LDAMapCollective
INPUT ($inputDir$, $metafile$, $outputDir$, <number of terms> $numOfTerms$, $numOfTopics$, $numOfDocs$, $numOfMapTasks$, $numOfIterations$)

*** $inputDir$ is the path to the data. In the $inputDir$, the data is splited into serval splits.
*** $metafile$ is the path to the metadata file. This file records the beginning doc index of each data splits.
*** $outputDir$ is the path to put the final results.
*** $numOfTerms$ is the size of the vocabulary.
*** $numOfTopics$ is the number of topics.
*** $numOfDocs$ is the number of documents in the dataset.
*** $numOfMapTasks$ is the number of mappers users want to launch.
*** $numOfIterations$ is the number of maximum running iterations.
parse Arguments
config a job
job.waitForCompletion()
return
```

```
Pseudocode LDAMapper
INPUT (KeyValReader $reader$)

*** $reader$ contains the paths to data files (splits)

*** Data strutures following are:
*** (Table) $dataTable$ is a numOfDocsInThisMapper $\times$ numOfTerms matrix
*** (Table) $alphaTable$ is a 1 $\times$ numOfTopics vector, represented by $\alpha$
*** (Table) $betaTable$ is a numOfTerms $\times$ numOfTopics matrix, represented by $\beta$
*** (Table) $gammaTable$ is a numOfDocs $\times$ numOfTopics matrix, represented by $\gamma$
*** (Table) $phiTable$ is a numOfTerms $\times$ numOfTopics matrix, represented by $\phi$
*** (Table) $globalPhiTable$ is a numOfTerms $\times$ numOfTopics matrix
*** (Table) $localPhiTable$ is a numOfTerms $\times$ numOfTopics matrix
*** (Table) $totalAlphaSufficientStatisticsTable$ is a 1 $\times$ numOfTopics vector

load data from $reader$ to (Table) $dataTable$
initialize (Table) $alphaTable$, $betaTable$
repeat
    for doc d in dataTable do
        compute loglikelihooodAlphaInOneDoc
        initialize $\gamma$
        *** E-step, train $\gamma$ and $\phi$
        *** also compute loglikelihoodPhiInOneDoc
        repeat
            for  v in vocabulary do
                for  k in topics do
                    $\phi_{v,k} = \beta_{v,k} \times \exp(\Psi(\gamma_{d,k}))$
                end for
                normalize row $\phi_{v,*}$
                *** $w_v$ is the value of dataTable(d, v), which is the count of the tokens of word v in doc d.
                update $\sigma = \sigma + w_v \phi_v$
            end for
            update $\gamma_{d,*} = \alpha + \sigma$
        until convergence
        compute loglikelihoodGammaInOneDoc
        LogLikelihood += likelihoodAlphaInOneDoc + likelihoodGammaInOneDoc + likelihoodPhiInOneDoc;
        update totalAlphaSufficientStatistics
        aggregate $\phi$ to $localPhiTable$
    end for

    *** M-step, update alpha and beta
    *** compute $\beta$
    push data from localPhiTable to globalPhiTable
    normalize data in globalPhiTable
    pull data from globalPhiTable to localPhiTable
    update new $\beta$ from localPhiTable

    *** compute $\alpha$
    allreduce totalAlphaSufficientStatistics
    compute $\alpha$ using Newton-Raphson method (need totalAlphaSufficientStatistics vector)


    *** compute loglikelihood
    allreduce loglikehood
until loglikehood converges or reach the maximum iterations
output results
return
```

The workflow can be found in figure 2. Multiple techniques are used to gain good scalability and speed.

- **Data preprocessing and compression.**Data preprocessing involved filtering common English stop words, stemming words to origin and removing non-alphanumeric words. We used Apache Lucene library[2] to devise the data preprocessing. Since the data is sparse, we adopt sparse matrix format to compress the dataset. It only stores (word_id:numOfOccurrence) pairs. For example, a dataset is as in table 1. After removing the stop words and stemming words to origin, we can get word-id pairs in table 2. The general option is to map this dataset to matrix, like in table 3. There are many zeros in table 3. To save memory space, our sparse matrix format for this dataset is in table 4.

doc_id	text
doc_0	I love programming.
doc_2	Smith is working.
doc_2	Harp is a collective library.

Table 1: a sample dataset

word	id	word	id
I	0	work	4
love	1	harp	5
program	2	collective	6
smith	3	library	7

Table 2: word-id pairs

1	1	1	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	1	1	1

Table 3: matrix representation

0:1	1:1	2:1
3:1	4:1	
5:1	6:1	7:1

Table 4: sparse matrix representation

- **Load balancing.**
  Load balance is important in distributed environment because it will reduce the waiting time. For example, if the algorithm is running on two machines. Machine1 takes 10% of the data and machine2 takes 90% of the data, machine1 will have to keep waiting until machine2 finishes it's work. The best case is that all machines finishing their computation at the same time and no waiting happens. We use two nodes in our experiments. So we partition the data into two partitions. Each partition contains the same number of documents. Besides, in the partition procedure, we keep the number of (word_id: numOfOccurrence) pairs similar in these two partitions. So each node will do similar amount of computation.

- **Dynamic scheduling.** We adopt harp's dynamic scheduling scheme to further divide the computation on thread level. For example, we launch 16 threads in each machine. Then every thread will take one document every time and do similar computation. Then after all documents are processed, the results from these threads will be aggregated to get the full model data.

- **Collective communication.** The computations are divided into multiple nodes. So it's necessary to do synchronization after computations. In our case, we will need to synchronize ϕ and α sufficient statistics (α' in figure 2). α' is a 1-dimensional vector and will be needed in every node. So allreduce communication scheme provided by harp is applied. ϕ is instead a matrix of words vs topics. It's a numOfTerms by numOfTopics matrix. It's not scalable to use allreduce for synchronizing this matrix because every node will store the whole matrix, which will be a bottleneck if the dataset is large. Also the communication overhead is very high. Notice that some words are not actually in some particular partitions. So the ϕ matrix in specific nodes is a numOfTermsInThisPartition by numOfTopics matrix. numOfTermsInThisPartition can be much smaller than numOfTerms. So push-pull communication scheme is more scalable for synchronizing this matrix.

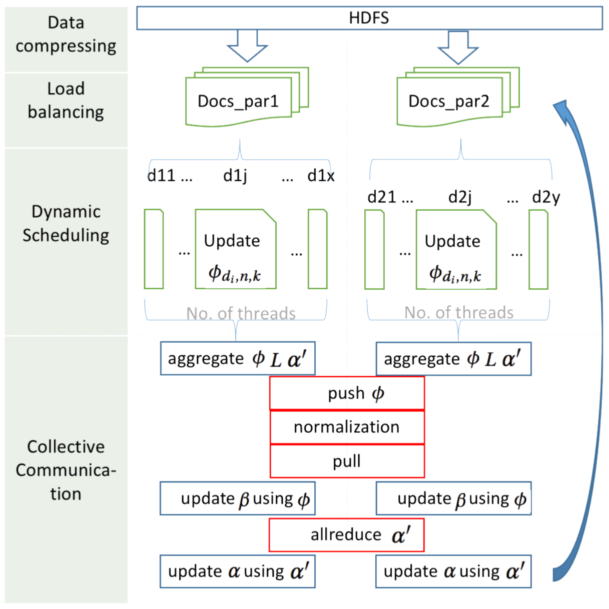


Figure 2: workflow

4. EXPERIMENTS

We worked on a Wikipedia dataset dump file. The total size of file was 12 GB. For our experimental procedures we samples the corpus into 3 sub-samples from the Wikipedia dataset. The details of the datasets are shown in table 5. We ran our experiments on Juliet Cluster with 2 allocated nodes and a maximum thread count of 16 threads. The number of topics in these experiments is fixed to be 100.

4.1 Evaluation of convergence

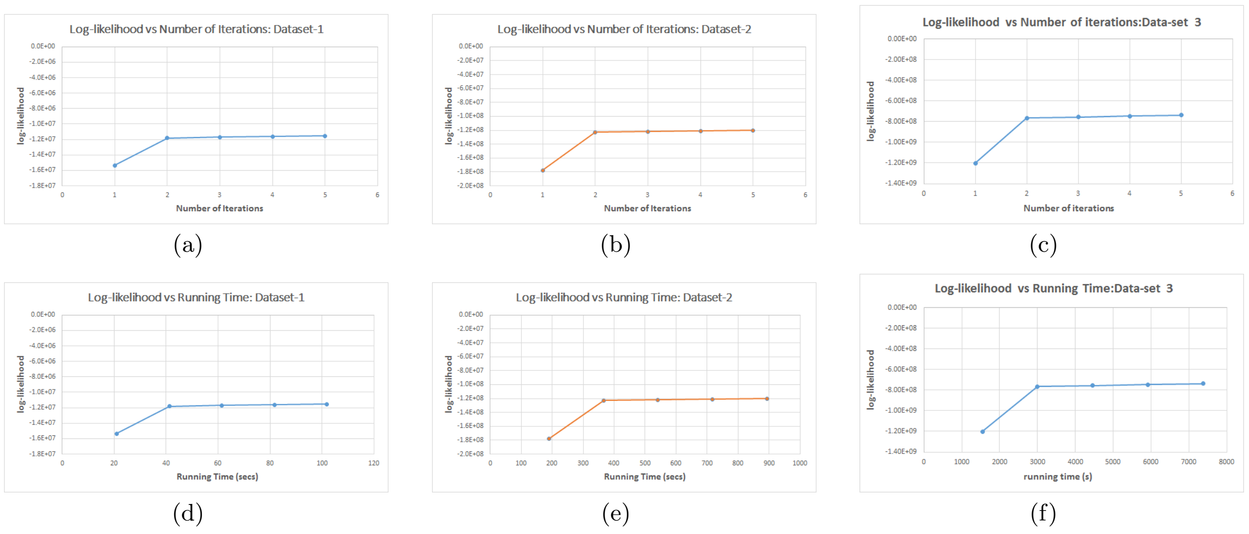


Figure 3: Evaluation of convergence

Name	Dataset-1	Dataset-2	Dataset-3
docsize	744	7698	80986
numOfWords	89907	435840	1454666
numOfTokens	535156	5308848	36096582

Table 5: datasets

An important metric for evaluation of our LDA model is log-likelihood. We conducted a series of experiments on increasing the size of the input data-sets (1k, 10k and 100k). For each experiment, 16 threads per node are launched.

The results in figure 3 provide an evidence depicting the log-likelihood results of Harp-LDA against the incrementing number of iterations and the noted running time. A clear indication of fast convergence of likelihood after a few number of iterations is observed.

4.2 Evaluation of scalability

The primary motivation for developing distributed algorithms for LDA is to have high scalable model in terms of memory and running time. Memory requirements depend on both memory for data and memory for model parameters. The variational LDA requires a few number of iterations to converge. Fewer iterations means less frequency of communication. The experiments for the evaluation of scalability of the proposed LDA model ensured by two measures viz. increasing the size of data-set and increasing the number of parallel threads running. We conducted a series of experiments on increasing the size of the input data-sets (1k, 10k and 100k). For each data set we gradually increased the number of parallel threads over a range of (1, 2, 4, 8 and 16). The results are shown in figure 4 and figure 5.

Two primary observations for these experiments were:

- **Scalability by data volume.** For each data-set increasing number of threads show a nearly linear growth in the speed up for the proposed LDA model. This depicts high scalability.
- **Scale up.** By increasing the number of thread per node, the nearly linear decrease of running time is observed.

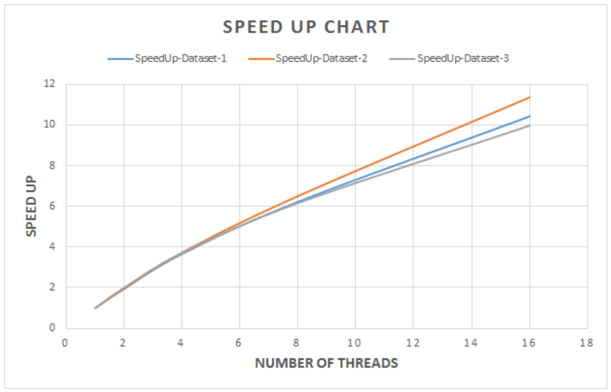
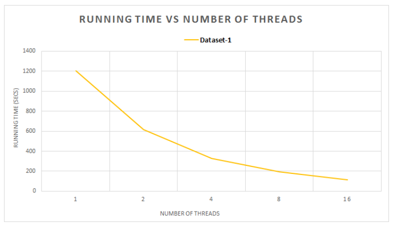


Figure 4: speedup

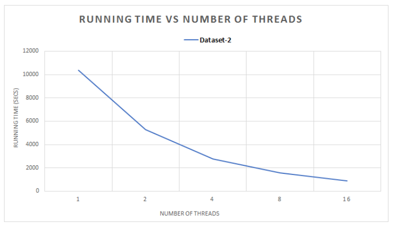
5. CONCLUSIONS

We proposed distributed LDA algorithm based on Harp. It represents a scalable mechanisms for inference of topic models. Our proposed LDA model based on Harp collective communication library enables efficient in-memory communication to avoid any excess HDFS read/write thus saving I/O cost. We also adopt specific data format, load balancing scheme, dynamic scheduling to achieve good performance and less memory requirements. The results from the experimental procedure on Wikipedia Dataset suggests clear scope of fast convergence and high scalability.

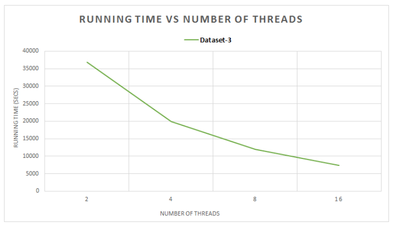
The limited time and resources of the project obstructs us from testing our algorithm more thoroughly. In the future, we will apply our algorithm to larger datasets and use more machines to verify the performance in real big data environment. And we will also find counterparts to do comparison in terms of scalability, convergence speed and accuracy measures.



(a)



(b)



(c)

Figure 5: Evaluation of scalability

6. ACKNOWLEDGMENTS

We thank Professor Judy Qiu, Bingjing Zhang, Po Peng and AI's help.

7. ADDITIONAL AUTHORS

8. REFERENCES

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[2] D. Cutting, M. Busch, D. Cohen, O. Gospodnetic, E. Hatcher, C. Hostetter, G. Ingersoll, M. McCandless, B. Messer, D. Naber, et al. Apache lucene, 2008.

[3] T. White. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[4] K. Zhai, J. Boyd-Graber, N. Asadi, and M. L. Alkhouja. Mr. lda: A flexible large scale topic modeling package using variational inference in mapreduce. In *Proceedings of the 21st international conference on World Wide Web*, pages 879–888. ACM, 2012.

[5] B. Zhang, Y. Ruan, and J. Qiu. Harp: Collective communication on hadoop. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pages 228–233. IEEE, 2015.