Distributed Implementation of the Latent Dirichlet Allocation on Spark

Karim Sayadi
CHArt Laboratory EA 4004
EPHE, PSL Research University
Paris, France
karim.sayadi@ephe.sorbonne.fr

Quang Vu Bui
CHArt Laboratory EA 4004
EPHE, PSL Research University
Paris, France
Hue University of Science, Vietnam
quang-vu.bui@etu.ephe.fr

Marc Bui
CHArt Laboratory EA 4004
EPHE, PSL Research University
Paris, France
marc.bui@ephe.sorbonne.fr

ABSTRACT

The Latent Dirichlet Allocation (LDA) is one of the most used topic models to discover complex semantic structure. However, for massive corpora of text LDA can be very slow and can require days or even months. This problem created a particular interest in parallel solutions, like the Approximate Distributed LDA (AD-LDA), where clusters of computers are used to approximates the popular Gibbs sampling used by LDA. Nevertheless, this solution has two main issues : first, requiring local copies on each partition of the cluster (this can be inconvenient for large datasets). Second, it is common to have read/write memory conflicts. In this article, we propose a new implementation of the AD-LDA algorithm where we provide computation in memory and a good communication between the processors. The implementation was made possible with the syntax of Spark. We show empirically with a set of experimentations that our parallel implementation with Spark has the same predictive power as the sequential version and has a considerable speedup. We finally document an analysis of the scalability of our implementation and the super-linearity that we obtained. We provide an open source version of our Spark LDA.

CCS Concepts

•**Computing methodologies** → **MapReduce algorithms;** Latent variable models; •**Information systems** → *Document topic models;*

Keywords

Latent Dirichlet Allocation; Big Data; Distributed Systems

1. INTRODUCTION

A significant portion of unstructured data in textual format is being collected from e-mail, social media platforms and diverse documents type. This data is used to train different models that are built to process information retrieval queries from diverse systems like modern libraries or search engines.

In this context, probabilistic topic modeling has been actively pursued as a language model to get an analytic abstraction from the text. Topic modeling is a method for analyzing quantities of unlabeled data to extract the latent structure (i.e the topic structure). This latent structure is represented by latent variables that we need to infer. One of the most used topic models is the Latent Dirichlet Allocation [5] where a document is a mixture of topics and a topic is a mixture of words. Learning the different parameters of the latent mixtures is a problem of Bayesian inference. Two of the approximate inference algorithms are used to overcome this problem: The variational inference and Gibbs sampling. The former is faster and the latter is slower but more accurate.

In information retrieval systems we need both speed and accuracy. However, for massive corpora of text, the iterations of Gibbs sampling are extremely slow and can require days or even months of execution time [14]. Clusters of computers can resolve this problem. To this aim, we propose a distributed version of Gibbs sampling built on Spark. Spark is a cluster computing and data-parallel processing platform for applications that focus on data-intensive computations. The main idea of the proposed algorithm is to make local copies of the parameters across the processors and synchronize the global counts matrices that represent the coefficients of LDA mixtures. The main contributions of this article are:

- We propose a parallel version of the collapsed Gibbs sampling based on the AD-LDA. This version allows a fast data partition and an effective synchronization between the latent variable on different machines.
- Experiments show both the sequential Gibbs sampling on a single CPU and the Spark cluster version with the same predictive power.
- We demonstrate scaling to very large systems and provide an open source implementation[1] of the Spark LDA

[1]http://gitlab.humanum.ephe.fr:82/Humanum/Humanum/

SoICT '16, December 08-09, 2016, Ho Chi Minh City, Viet Nam

DOI: http://dx.doi.org/10.1145/3011077.3011136

version with the different experimentations.

2. BACKGROUND

2.1 The Latent Dirichlet Allocation

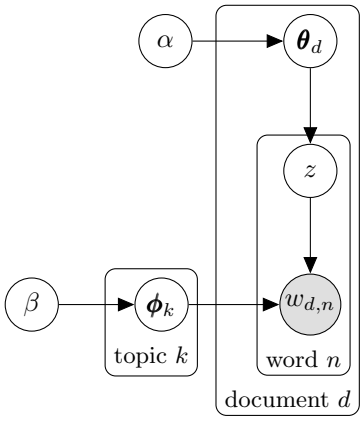

Figure 1: Bayesian network of the Latent Dirichlet Allocation.

Latent Dirichlet Allocation (LDA) by Blei et al. [5] is a generative probabilistic model for collections of grouped discrete data. Each group is described as a random mixture over a set of latent topics where each topic is a discrete distribution over the collection's vocabulary.

The generative model of LDA [5] is described with the probabilistic graphical model [9] on the left of Fig. 1. In this LDA model, different documents d have different topic proportions θ_d. In each position in the document, a topic z is then selected from the topic proportion θ_d. Finally, a word is picked from all vocabularies based on their probabilities ϕ_k in that topic z. θ_d and ϕ_k are two Dirichlet distribution with α and β as hyperparameters. We assume symmetric Dirichlet priors with α and β having a single value.

The hyperparameters specify the nature of the priors on θ_d and ϕ_k. The hyperparameter α can be interpreted as a prior observation count of the number of times a topic z is sampled in document d [17]. The hyperparameter β can be interpreted as a prior observation count on the number of times words w are sampled from a topic z [17].

As a latent variable model the inference in LDA is computationally expensive. The inference in LDA is the process of estimating the posterior distribution, which is solved with the following equation :

$$p(\theta, \phi, z|w, \alpha, \beta) = \frac{p(\theta, \phi, z, w|\alpha, \beta)}{p(w|\alpha, \beta)} \tag{1}$$

As a consequence to this, approximate inference algorithms, e.g Variational Inference and Gibbs Sampling are widely used in the literature [17].

In the original LDA paper [5] Blei et. al used variational inference. The variational inference is faster than Gibbs sampling but, it can be biased. However, in Gibbs sampling as the number of iteration for running the Markov Chain increases the bias approaches 0. Gibbs sampling is then a more accurate algorithm for LDA even if it is slower. Recently, some efforts [16] were oriented to combine both methods to yield greater speeds and accuracy but they are not applied yet on LDA.

There are different versions of Gibbs sampling algorithm, we have chosen to work with the collapsed version [17]. In [13], Newmann et al. showed with empirical results that the non-collapsed and partially collapsed sampling algorithms converge slower than the fully collapsed. The non-collapsed exhibit slower convergence due to strong dependencies between parameters and latent variables.

The Gibbs sampler for LDA needs to compute the probability of a topic z being assigned to a word w_i, given all other topic assignments to all other words. Somewhat more formally, we are interested in computing the following posterior up to a constant:

$$p(z_i \mid z_{-i}, \alpha, \beta, w) \tag{2}$$

where z_{-i} means all topic allocations except for z_i.

$$P(z_i = j|z_{-i}, w) \propto \frac{n^{w_i}_{-i,j} + \beta}{n^{(\cdot)}_{-i,j} + V\beta} \frac{n^{d_i}_{-i,j} + \alpha}{n^{d_i}_{-i,\cdot} + K\alpha} \tag{3}$$

where $n^{w_i}_{-i,j}$ is the number of times word w_i was related to topic j. $n^{(\cdot)}_{-i,j}$ is the number of times all other words were related with topic j. $n^{d_i}_{-i,j}$ is the number of times topic j was related with document d_i. $n^{d_i}_{-i,\cdot}$ is the number of times all other topics were related with document d_i. V is the number of words in the vocabulary and K is the number of topics. Those notations were taken from the work of Thomas Griffiths and Mark Steyvers [6].

$$\hat{\phi}^{(w)}_j = \frac{n^{(w)}_j + \beta}{n^{(\cdot)}_j + V\beta} \tag{4}$$

$$\hat{\theta}^{(d)}_j = \frac{n^{(d)}_j + \alpha}{n^{(d)}_{\cdot} + K\alpha} \tag{5}$$

Equation (4) is the bayesian estimation of the distribution of the words in a topic. Equation (5) is the bayesian estimation of the distribution of topics in documents.

From the equation (3) the Gibbs sampling update the topic assignment z_i in each iteration for every word in every document. This process can become slower as the corpora of text become bigger. It may require days or months of CPU time to finish the iterations [14]. A natural way to solve this problem is to divide the collection of documents across P processor. Each processor updates the topic assignment locally on the portion of the collection and then sends the result to synchronize with the other processors. This is the idea behind Approximate Distributed LDA (AD-LDA).

2.2 Approximate Distributed LDA

In the Approximate Distributed LDA model (AD-LDA) first proposed by Newman et al. [14], a corpus is divided on P processors, with approximately $\frac{D}{P}$ documents on each processor. Then, LDA is implemented on each processor, and Gibbs sampling is simultaneously executed on each $\frac{D}{P}$ documents to approximate a new z_i from the equation (3) for every word i in every document j in the collection of documents.

tree/master/SparkLDA

In each iteration each processor has local copy of the counts matrix word by topic n_p^{wk} and the counts matrix document by topic n_p^{dk} in parallel. A global synchronization described by the equation (6) and (8) is executed to have global counts n^{wk} and n^{dk}.

$$n_{\text{new}}^{wk} = n_{old}^{wk} + \sum_p (n_{\text{p}}^{wk} - n_{old}^{wk}) \quad (6)$$

$$= \sum_p n_p^{wk} - (p-1) n_{old}^{wk} \quad (7)$$

$$n_{\text{new}}^{dk} = n_{old}^{dk} + \sum_p (n_{\text{p}}^{dk} - n_{old}^{dk}) \quad (8)$$

$$= \sum_p n_p^{dk} - (p-1) n_{old}^{dk} \quad (9)$$

There are two main issues with AD-LDA : first it needs to store P copies of the global counts for all the processors in parallel, this can be inconvenient for large datasets. Second, we can have read/write memory conflicts on the global counts n^{wk} and n^{dk} which can lower the prediction accuracy.

Spark provides computation in memory and therefore we don't need to store the global counts in parallel. We can avoid read/write memory conflicts with broadcasting temporary copies of the global counts n^{wk} and n^{dk}. In the next section, we explain why we chose to work with Spark instead of Hadoop/MapReduce and then present the algorithm of our implementation.

3. SPARK LDA

3.1 Spark

Spark [20] is a cluster computing and data-parallel processing platform for applications that focus on data-intensive computations. The main component and primary abstraction in Spark is the Resilient Distributed Dataset (RDD). An RDD is a distributed collection of elements in memory that is both fault-tolerant (i.e. Resilient) and efficient (i.e the operation performed on them are parallelized).

Spark automatically distributes the data contained in the different RDDs and applies in parallel different operations (i.e functions) defined within the so-called driver program. The driver program contains the application (e.g Spark LDA) main functions (e.g MCMC methods) and applies them on the cluster.

The prominent difference between MapReduce/Hadoop and Spark is that the former creates an acyclic data flow graph [4] and the latter a lineage graph [20]. As soon as the Hadoop implementation became popular, users wanted to implement more complex applications; iterative algorithms (e.g machine learning algorithms), interactive data mining tools (e.g Python, R) that can not be expressed efficiently as acyclic data flows.

In our work, we used the Gibbs sampling algorithm to approximate the inference. This Monte Carlo algorithm depends on randomness. Hadoop/MapReduce does not allow us to have this randomness because it considers each step of computation in the implemented application as the same no matter where or when it runs. However, this problem can be fixed by seeding a random number generator [12], it adds another layer of complexity to the implementation and can slow it down or affect the complete synchronization of the counts of different parameters distributed on the cluster.

We chose Spark, because it is faster than Hadoop (i.e. computation in memory), allows randomness, iterative jobs, general programming tasks (i.e Machine Learning algorithms are not usually built for Map Reduce tasks). We will use three simple data abstractions provided by the Spark syntax to program the clusters : the RDDs, broadcast variables for the global counts of the different parameters in the Gibbs sampling and the Map and Reduce implementation.

3.2 The Algorithm

Algorithm 1 Algorithm : for distributing the Latent Dirichlet Allocation

```
1: procedure SparkLDA
2:     RddData = sc.textFile("textexample.txt")
3:     If FirstIteration then
4:         initialize global counts at Random
5:     loop For each iteration
6:         Begin
7:             rdd = RddData.Map ()
8:             globalcounts = rdd.ReduceAndUpdate()
9:             rddGlobal = globalcounts.parallelize()
10:             rdd = rddGlobal.Map()
11:             φ, θ = rdd.ReduceAndUpdate()
12:         End
13:     return φ, θ
14: end procedure
```

Our implementation on Spark of the LDA with collapsed Gibbs sampling (CGS) can be considered as an extension of the AD-LDA algorithm where we aim to provide computation in memory and a good communication between the processors to avoid read/write memory conflicts when we synchronize the global counts n^{wk} and n^{dk}. Spark generalizes the MapReduce model, therefore before presenting the overall algorithm 1, we will present what is executed on each processor in the algorithm 2 and what is synchronized through all the processors in the algorithm 3.

Algorithm 2 Algorithm : Mapper

```
1: procedure Map
2:     D_1, ···, D_p = partition (Corpus(D))
3:     loop For each partition do in parallel
4:         Begin
5:             n_p^wk = n^wk
6:             n_p^dk = n^dk
7:             loop For each document j and for each word i in V
8:                 Begin
9:                     Sample z_ij from n_p^wk and n_p^dk using CGS.
10:                     Get n_pnew^wk and n_pnew^dk from z_ij.
11:                 End
12:             Get the broadcasted n^wk and n^dk and compute the φ
    and θ equation (4) and (5).
13:         End
14:
15: end procedure
```

The first procedure called *Map* described the algorithm 2, represents the instructions executed on P processors to sample topics locally. These instructions are the parts of the algorithm that benefit from the improvement of the system (i.e the increase of the number of processors or memory). First, we initialize the global counts with the synchronized versions then we sample z_{ij} and produce new global counts that we broadcast again to compute the distribution of top-

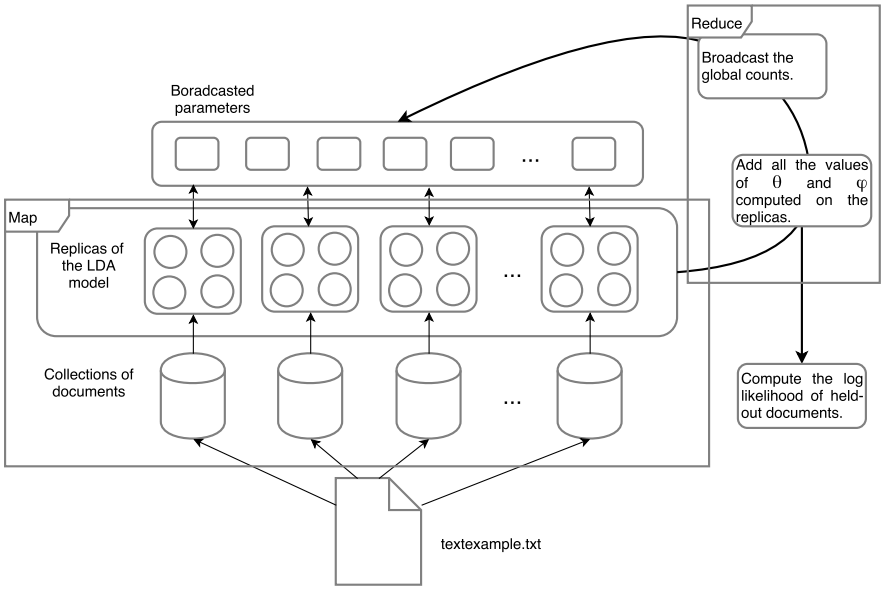

Figure 2: Workflow of the Spark implementation. Each iteration is executed within the Map and Reduce framework. After the reduce application the global counts parameters n^{wk} and n^{dk} are broadcasted to the different replicas of the LDA model. The log-likelihood is computed on the held-out documents after each iteration.

Algorithm 3 Algorithm : Reducer

1: **procedure** ReduceAndUpdate
2: n^{wk}_{new} = equation (6)
3: n^{dk}_{new} = equation (8)
4: broadcast(n^{wk}_{new},n^{dk}_{new})
5: θ = add($\theta_{1...p}$)
6: ϕ = add($\phi_{1...p}$)
7: **end procedure**

ics per document θ and the distribution of words per topic ϕ.

The second procedure called *ReduceAndUpdate*, in the algorithm 3, is executed at the end of each Gibbs sampling iteration to synchronize the word-topic counts and to compute the words per topic count matrix ϕ and the topic per document count matrix θ from all the local ones in different partitions P. The Map and Reduce procedures are executed until convergence of the Gibbs algorithm.

Finally, the overall algorithm in Spark uses the defined procedures Map and Reduces, shown in the algorithms 2 and 3, to perform the computations on the different partitions where we divided our collections of documents (e.g. an input text file "textexample.txt"). The interconnections between the different procedures of the Spark implementation are depicted in 2. In the next section, we will evaluate the results of our implementation.

4. EXPERIMENTS AND RESULTS

In this section, we document the speed and the likelihood of held-out documents. The purpose of the experiments is to investigate how our distributed version of LDA on Spark performs when executed on small and big data.

We report three evaluations : first the perplexity of the sequential CGS of LDA and the distributed CGS on Spark. Second, after an investigation of the Spark job in the workload, we discuss the percentage of the execution time that benefits from the improvement of the resources and we compute the speedup. Finally, we discuss the scaling behavior of our proposition. The results are reported on three datasets retrieved from the UCI Machine Learning Repository[2].

4.1 Data Collection and Processing

Table 1: Description of the four datasets used in experiments

	Nips	**KOS**	**NYT**
D	1,500	3430	300,000
V	12,419	6906	102,660
N	2,166,058	467714	99,542,125

We used three datasets from UCI Machine Learning Repository. Table 1 summarizes the information about the dataset where D is the number of documents, V is the size of the vocabulary, and N is the number of tokens.

Nips and KOS are considered as small data, we used them to report the experimentation on the perplexity between the sequential CGS and the distributed CGS. The NYT dataset is considered as a big data set. We used the datasets to report the experimentation on the speedup and the scaling behavior.

The downloaded files from UCI are formatted as following : each line in the file has an ID that corresponds to a document, an ID for a word in the vocabulary V and the number of occurrence of this work in the document. We needed to transform this format to the following : each line in the input file contains the ID of the document, followed by the ID of the different words in the vocabulary.

4.2 The Environment

[2]https://archive.ics.uci.edu/ml/datasets/Bag+of+Words

We run all the experiments on a cluster with 10 nodes running Spark 1.5.2. Each node has an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz (4 cores/8threads), 4 of them have 32GB of RAM and the rest have 16GB of RAM.

4.3 Analyzing the Results

We empirically validate our results by analyzing the computed perplexity and the scaling behavior of our sequential and distributed version of LDA.

4.3.1 Evaluation with the Perplexity

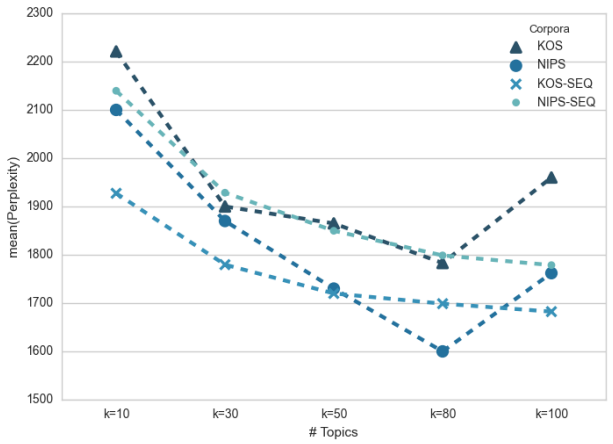

Figure 3: Perplexity values comparison between the Spark implementation and the sequential LDA version denoted by SEQ. The perplexity was computed with different number of topics for the KOS and NIPS datasets.

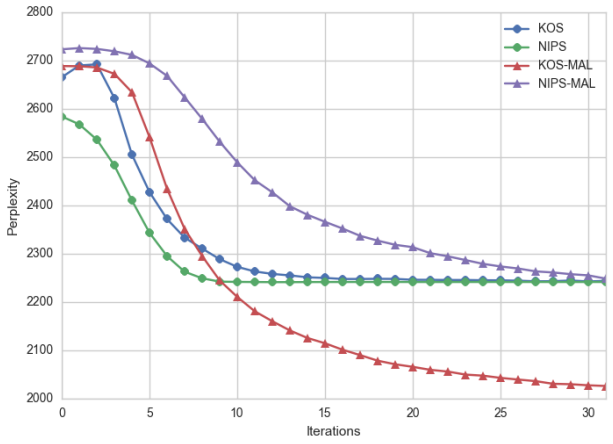

Figure 4: Convergence of the perplexity for the KOS and NIPS datasets compared to the convergence of the Mallet (MAL) multi-thread paralleling system.

To evaluate the prediction quality of the trained Spark LDA we measure the log-likelihood of a held-out test set given an already trained model [18]. This evaluation is called the test set perplexity and it is defined as

$$exp(-\frac{1}{N_{test}} \log p(x^{test})) \tag{10}$$

For LDA, the test set is a set of unseen document W_d and the trained LDA is represented by the distribution of words ϕ. We compute the likelihood $p(x^{test})$ using $S = 10$ samples with 1000 iterations.

$$p(x^{test}) = \prod_{ij} \log \frac{1}{S} \sum_s \sum_k \hat{\theta}^s_{jk} \hat{\phi}^s_{x_{ijk}} \tag{11}$$

$$\hat{\theta}^s_{jk} = \frac{\alpha + n^s_{jk}}{K\alpha + \sum_k n^s_{jk}} \quad \hat{\phi}^s_{x_{ijk}} = \frac{\beta + n^s_{wk}}{W\beta + n^s_k}$$

Where $\alpha = 50/K$ and $\beta = 0.1$.

We used two small datasets from UCI (i.e. KOS and NIPS) to show that the Spark cluster version of CGS has the same predictive power as the sequential version. We computed the test perplexity with different initializations of the parameters.

In this work, we set aside 25% of the documents in each corpus as a test set and train on the remaining 75% of documents. We then compute predictive rank and predictive likelihood. In Figure 3, we compare the perplexity values of the sequential LDA version and our distributed LDA version on Spark. For this plot, we set different numbers of topics. We observe that our implementation has the same predictive power as the sequential version. It has even better predictive power for the bigger dataset, in the case the NIPS dataset. For example, when $K = 80$ the perplexity is equal to 1600, compared to 1800 for the sequential version.

For a fixed number of K, we observe in Figure 4 that our implementation converges to models having the same predictive power as standard LDA. In this figure, we compare the accuracy of our implementation to the most used framework for topic modeling, called Mallet. Mallet [11] uses multi-thread to distributed the computation of the Gibbs sampling. For this experimentation, we used for our implementation 12 cores and 12 threads for the Mallet instance and K = 10.

We note in Figure 4, that our implementation represented by the circle points converges before the Mallet instance represented by the triangle points. And this, for the two datasets NIPS and KOS. Whereas for the smallest dataset, i.e. KOS, the Mallet has a better accuracy, for the NIPS our implementation performs better with high convergence rate.

4.3.2 Speed Up of the Algorithm

We report in this part the speedup of the collapsed Gibbs sampling (CGS) with a metric which compares the improvement in speed of execution of a task on two similar architectures with different resources. We compute the speedup S based on Amdahl law [3] presented in the equation below

$$S = \frac{1}{1 - p + \frac{p}{s}} \tag{12}$$

where p is the percentage or the portion of the overall task that benefits from the changes in the resources of the architecture and s is the number of resources. Here s is equal to the number of processors. Our speed up experiments is conducted on the NIPS, KOS, and the large NYT dataset.

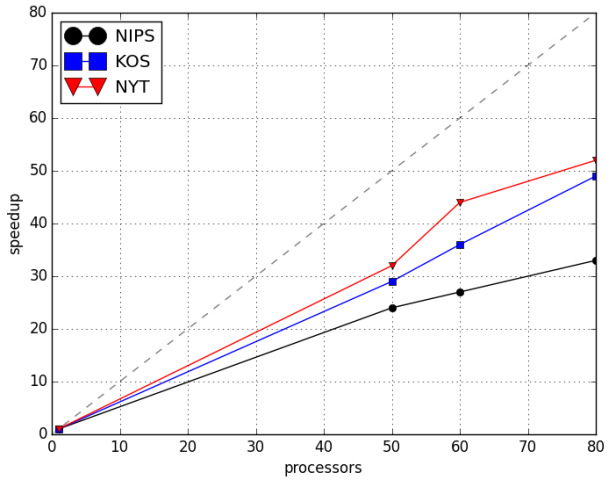

Figure 5: Speed up of the spark implementation compared to the number of processors.

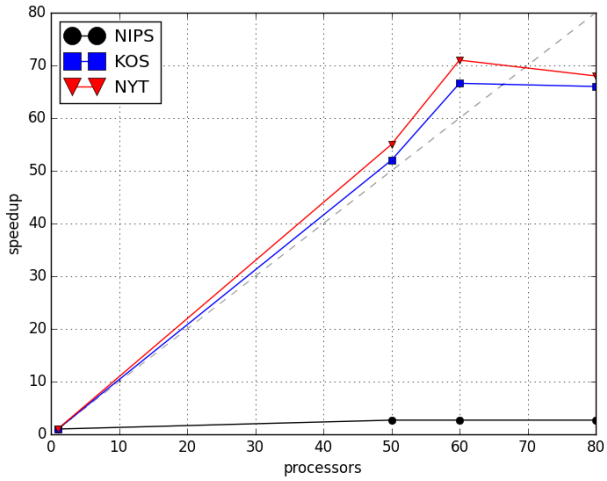

Figure 6: Scaling behavior of the Spark implementation computed on NIPS KOS and NYT datasets.

In our algorithm we have split the Gibbs sampling part into three consecutive parts as shown in the Algorithm 3 : We broadcast the reduce global counts n^{wk} and n^{dk} then we compute the distribution for θ and ϕ. Finally, we compute the overall speedup by using this equation $s = \frac{1}{\frac{p_1}{s_1} + \frac{p_2}{s_2} + \frac{p_3}{s_3}}$ from Amdahl law.

Figure 5 shows a strong correlation between the speedup and the size of the dataset. The speedup approaches the linear case with the NIPS and NYT datasets. For the KOS dataset, we observe a stable speed up from 60 processors. This is due to the small number of words in each partition that have no more effect on the time of the sampling.

4.3.3 *Scaling Behavior of the Spark Implementation*

Once a program is developed with the Spark syntax and worked on a small number of cluster nodes, it can be scaled to an arbitrary number of nodes with no additional development effort. The scalability here is the ability to speed up a task with improving the resources of an architecture of a particular system.

The main point of this part of experimentation is to quantify the scalability of the overall distributed implementation on the Spark framework. To this end, we will use the Universal Scalability Law introduced by Dr. Gunther [7], to analyze the configuration of the system that matches a speedup result. The speed up analyzed in section 4.3.2 was about the sampling time and not the scalability of the overall implementation. First, we redefine the speed up factor, in equation 13.

$$S_p = \frac{T_1}{T_p} \tag{13}$$

Where T_1 is the measured time on 1 processor and T_p is the measured time on p processors. The equation 13 is a generalization of the Amdahl law and is obtained from the performance model represented in equation 14.

$$S_p = \frac{p}{1 + \sigma(p-1) + \kappa p(p-1)} \tag{14}$$

where σ represents the degree of contention between different parts of the system and κ represents the delay needed to keep the system coherent. A delay caused by the distribution of the data on the different partitions.

Figure 6 that describes the scaling behavior of the algorithm, we observe that we have a sublinear speedup for the KOS data. For the NIPS and NYT datasets, we observe a super linear speed up and then from 65 processors the scalability curve cross the linear bound and enter what it is called a payback region. A detailed explanation of this phenomenon was studied on a Hadoop cluster and can be found in this paper [8].

5. RELATED WORKS AND DISCUSSION

To the best of our knowledge, there exist two works similar to our implementation. The first [15] showed good results with the same datasets that we used but the authors did not give any accounts about the global synchronization. For example, in their implementation, the authors did not synchronize C^{DK} alias the document-topic global counts matrix which important to decide when to stop the sampling algorithm (i.e. if the C^{DK} does not change through the iterations). Moreover, the authors did not report the scaling behavior of their implementation, we found that the paper lacked details of implementation (e.g. the broadcasted counts, the instructions in the map and reduce method), and we could not find their code.

The second [1] gave more details about the code but the author did not show any evaluation of the results and the implementation does not work directly on any text file.

We mention also the work of the Spark community on the LDA. In their implementation, they use online variational inference as a technique for learning the LDA models. The community announced that they are currently working on a Gibbs sampling version to improve the accuracy of the LDA package that they propose.

We cite in the following two of the works that did not implement their proposition in Spark but offered interesting solutions for the communication and memory management.

Wang et al. [19] implemented LDA using Gibbs sampling. To overcome the issue of consistent counts the authors used message passing between the different partitions on the cluster. This IO/Communication dominates the cost in time of their parallel algorithm which affects the performance of the implementation. In our work, we don't have this problem since we work in memory and we don't have any persistence on the disk.

Ahmed et al. [2] tackled the problem of synchronizing the latent variable of the modeling between different machines. The authors used a distributed memory system to achieve consistent counts and the dual decomposition methods to make local copies of the global variables to obtain consistency in the final counts. The ideas in this work are very close to the Spark framework.

6. CONCLUSION AND FUTURE WORK

We proposed in this article a distributed version of the Latent Dirichlet Allocation that was implemented in Spark. We reduced the I/O communication and the memory conception by tackling the synchronization of the latent variables of the model. The next step of our work is to improve our implementation to handle the Medline dataset. We also intend to implement a streaming distributed version of LDA where the documents will be processed as they are crawled from the internet in general or social media in particular (e.g Twitter).

7. ACKNOWLEDGMENTS

The authors would like to thank the Jhon von Neuman Institute (VNUHCM), Ho Chi Minh City, Vietnam, for providing us high-performance computing resources along with an entire ecosystem of services.

8. REFERENCES

[1] mertterzihan/pymc, https://github.com/mertterzihan/pymc, 2015-07-02.

[2] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola. Scalable inference in latent variable models. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 123–132. ACM, 2012.

[3] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

[4] Arvind and D. E. Culler. Dataflow Architectures. *Annual Review of Computer Science*, 1(1):225–253, 1986.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.

[6] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences of the United States of America*, 101(Suppl 1):5228–5235, 2004.

[7] N. J. Gunther. A General Theory of Computational Scalability Based on Rational Functions. *arXiv:0808.1431 [cs]*, Aug. 2008. arXiv: 0808.1431.

[8] N. J. Gunther, P. Puglia, and K. Tomasette. Hadoop superlinear scalability. *Communications of the ACM*, 58(4):46–55, Mar. 2015.

[9] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[10] Z. Liu, Y. Zhang, E. Y. Chang, and M. Sun. Plda+: Parallel latent dirichlet allocation with data placement and pipeline processing. *ACM Transactions on Intelligent Systems and Technology, special issue on Large Scale Machine Learning*, 2011. Software available at https://github.com/openbigdatagroup/plda.

[11] A. K. McCallum. Mallet: A machine learning for language toolkit, 2002.

[12] Z. Ming, C. Luo, W. Gao, R. Han, Q. Yang, L. Wang, and J. Zhan. Bdgs: A scalable big data generator suite in big data benchmarking. In *Advancing Big Data Benchmarks*, pages 138–154. Springer, 2014.

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

[14] D. Newman, P. Smyth, M. Welling, and A. U. Asuncion. Distributed inference for latent dirichlet allocation. In *Advances in neural information processing systems*, pages 1081–1088, 2007.

[15] Z. Qiu, B. Wu, B. Wang, and L. Yu. Gibbs Collapsed Sampling for Latent Dirichlet Allocation on Spark. In *Journal of Machine Learning Research*, pages 17–28, 2014.

[16] T. Salimans, D. P. Kingma, and M. Welling. Markov Chain Monte Carlo and Variational Inference: Bridging the Gap. *arXiv:1410.6460 [stat]*, Oct. 2014. arXiv: 1410.6460.

[17] M. Steyvers and T. Griffiths. Probabilistic topic models. *Handbook of latent semantic analysis*, 427(7):424–440, 2007.

[18] H. M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno. Evaluation methods for topic models. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1105–1112. ACM, 2009.

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

[20] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[21] 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.