Distributed Gibbs Sampling and LDA Modelling for Large Scale Big Data Management on PySpark

Christos Karras*, Aristeidis Karras*, Dimitrios Tsolis†, Konstantinos C. Giotopoulos‡, Spyros Sioutas*
*Decentralized Systems Computing Group, Computer Engineering and Informatics Department, University of Patras, Greece
{c.karras, akarras, sioutas}@ceid.upatras.gr
†Department of History and Archaeology, University of Patras, Greece
dtsolis@upatras.gr
‡Department of Management Science and Technology, University of Patras, Greece
kgiotop@upatras.gr

2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM) | 979-8-3503-9858-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SEEDA-CECNSM57760.2022.9932990

***Abstract*—Big data management methods are paramount in the modern era as applications tend to create massive amounts of data that comes from various sources. Therefore, there is an urge to create adaptive, speedy and robust frameworks that can effectively handle massive datasets. Distributed environments such as Apache Spark are of note, as they can handle such data by creating clusters where a portion of the data is stored locally and then the results are returned with the use of Resilient Distributed Datasets (RDDs). In this paper a method for distributed marginal Gibbs sampling for widely used latent Dirichlet allocation (LDA) model is implemented on PySpark along with a Metropolis Hastings Random Walker. The Distributed LDA (DLDA) algorithm distributes a given dataset into P partitions and performs local LDA on each partition, for each document independently. Every n^{th} iteration, local LDA models, that were trained on distinct partitions, are combined to assure the model ability to converge. Experimental results are promising as the proposed system demonstrates comparable performance in the final model quality to the sequential LDA, and achieves significant speedup time-optimizations when utilized with massive datasets.**

***Index Terms*—Distributed Gibbs Sampling, Random Walker, Metropolis Hastings, LDA, Big Data Management, PySpark**

I. Introduction

In the big data era, the amount of the produced information is increasing every day, creating many opportunities for machine learning, but also sets serious barriers on how to handle and analyse extremely large datasets effectively in terms of memory constraints and processing time. The latent Dirichlet allocation model (LDA) [1]–[9], is among the most used methods to overcome the aforementioned matter for topic modelling. LDA is a hierarchical Bayesian model with three layers that is sought to investigate latent schemes in document corpora. Each document is represented as a random mixture of subjects, with each topic characterized by a distribution across words. Either variational inference or Markov Chain Monte Carlo (MCMC) techniques may be used to develop an LDA model, and both have benefits and trade-offs. The primary benefit of variational inference is a large reduction in processing time, but at the cost of potentially erroneous conclusion. Contrariwise, on MCMC approaches, precision is gained at the expense of computational complexity, rendering these methods inapplicable to massive datasets.

In this paper, a distributed implementation of an MCMC method is introduced along with a marginal Gibbs sampling technique, for creating a Distributed LDA on PySpark (DLDA). As the number of variables is far more than the number of cores utilised for parallelization [10]–[13], the technique adheres to the concept of weak dependence between variables, allowing data to be partitioned into P sets of partitions and LDA to be performed locally.

In particular, in the context of this paper the following contributions take place: i) A distributed Gibbs Sampling method over PySpark is implemented. ii) A Metropolis Hastings Random Walker over PySpark is introduced. iii) A marginal Gibbs sampling for LDA on Spark is implemented, namely as DLDA. iv) An evaluation of the effectiveness of the DLDA model for latent topic discovery task takes place. v) An evaluation of the influence of the parameters of DLDA on the model performance and execution time occurs.

The remaining of the paper is organized as follows. In Section II the preliminaries of this work are introduced followed by subsection II-A whereabouts a brief overview to plain LDA models and marginal Gibbs sampling techniques is introduced, while in Section III the proposed system is covered in both theory and application level. Specifically, in subsection III-B a detailed analysis on the distributed LDA implementation using PySpark is presented. In Section IV, an evaluation of the results of the distributed model is provided and a comparison to the sequential algorithm takes place. Finally, the conclusions and future directions of this work are presented in Section V.

II. Preliminaries

A. Latent Dirichlet Allocation

Before proceeding with the details of DLDA developed, the standard LDA model is briefly reviewed and shown in Figure 1 using plate notation. LDA represents each of D documents as a mixture of K latent topics, where each topic is a multinomial probability distribution over a vocabulary of V words. The process of generating a new document j is described as follows:

979-8-3503-9858-8/22/$31.00 ©2022 IEEE

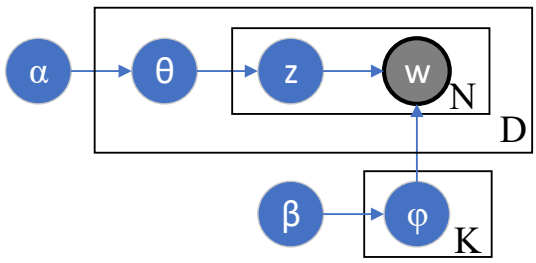

Fig. 1: Graphical model representation for LDA. Observed variable w is shaded with grey color.

- from a Dirichlet distribution with parameter α, draw a mixing proportion $\theta_{k|j}$.
- for the i^{th} word within the document, first draw a topic assignment z_{ij}, where topic k is chosen with probability of $\theta_{k|j}$, and then the value w of word x_{ij} is drawn from the z_{ij} topic with probability $\phi_{w|k}$, where $\phi_{w|z_{ij}}$ which is drown from a Dirichlet prior with parameter β.

The preceding description of the process of document generation is equivalent to:

$$\begin{array}{cc} \theta_{k|j} \sim \mathrm{Dir}(\alpha) & \phi_{w|k} \sim \mathrm{Dir}(\beta) \\ z_{ij} \sim \theta_{k|j} & x_{ij} \sim \phi_{w|z_{ij}} \end{array} \tag{1}$$

where α and β are fixed Dirichlet priors.

Let the observed words to be $\mathbf{x} = \{x_{ij}\}$. The aim is to compute the posterior distribution over the latent variables z, θ and ϕ. The two most-frequently used inference procedures are based on either using variational methods [1] [14] [15] or Markov Chain Monte Carlo (MCMC) methods [16]–[28]. This paper focuses on the latter, and more precisely, on a marginal Gibbs sampling inference procedure for LDA, initially proposed in [16]. The samples of the marginal Gibbs sampling with latent variable z with θ and ϕ are desegregated. In this scenario the conditional probability of z_{ij} is defined as:

$$p(z_{ij} = k | z^{\neg ij}, x, \alpha, \beta) = \frac{c_{k,m,\cdot} + \alpha}{N_j^{\neg i} + K\alpha} \frac{c_{k,\cdot,n} + \beta}{c_{k,\cdot,\cdot} + V\beta} \tag{2}$$

where $\neg ij$ denotes word i within a document j that is eliminated in the count values and $c_{k,m,n}$ represents the number of times topic k is assigned to a word n in the document m. Missing index value for $c_{k,m,n}$ (e.g. $c_{k,\cdot,n}$) denotes summing over that index (e.g. $c_{k,\cdot,n} = \sum_{m=1}^{M} c_{k,m,n}$).

The deployment of a sequential marginal Gibbs sampler for LDA is pretty straightforward as it operates on a set of count values, $c_{k,\cdot,m}$, $c_{k,m,\cdot}$ and $c_{k,\cdot,\cdot}$. The algorithm initiates by randomly assigning a topic to a word in a document, updating the count values accordingly, and performs a loop over the number of iterations to reassign a topic to a word according to the conditional probability. The outline of LDA marginal Gibbs sampling is presented in Algorithm 1.

Algorithm 1 Gibbs Sampling

Input: latent variable $z^{(0)} = \langle z_1^{(0)}, ..., z_k^{(0)} \rangle$

begin
for all $t = 1$ to T **do**
  for all $i = 1$ to k **do**
    $z_i^{(t+1)} \sim P(Z_i | z_1^{(t+1)}, ..., z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, ..., z_k^{(t)})$
  end for
end for
end

III. Methodology

A. Distributed Gibbs Sampling over Spark

At this point, think of a pretty basic latent variable model being a mixture of exponential distributions comprised by a finite number of components. Then, for each data-point y_i suppose that y_i is drawn among one of K distinct exponential-based distributions. Hence, each of the data-points will have a distinct time scale parameter as in (3).

$$\begin{aligned} y_i \sim w_1\theta_1 \exp(-\theta_1 y) + w_1\theta_2 \exp(-\theta_2 y) + \ldots \\ + w_K\theta_K \exp(-\theta_K y) \end{aligned} \tag{3}$$

Without sacrificing generality, assume that we only observe just a two-component model. This leads to the following model of generation as in (4).

$$y_i \sim w_1\theta_1 \exp(-\theta_1 y) + w_1\theta_2 \exp(-\theta_2 y) \tag{4}$$

Then, for every model component, the aim is to estimate a single parameter θ_k for each model component along with the relative weight for the corresponding component w_k. The preceding model although is a fundamental one, it has the same inner workings among many latent variable models. Therefore, given some data of form $\{y_1, y_2, ...\}$ denoted as y_N, the aim is to estimate the model hyperparameters $\{w_1, w_2, \theta_1, \theta_2\}$. Utilizing a Bayesian approach, the goal is to investigate the posterior distribution $p(\theta_1, \theta_2, w_1, w_2 | y_N)$.

Similarly with Bayesian mixture models, the next step is the data augmentation (only applied to the parameters and not on the data entirely). We perform so, by adding a latent indicator variable s_i for every data-point y_i. This added indicator mainly points to one of the components $s_i \in \{1, 2\}$ in order to identify which of the components is possibly generated for a specific data-point.

In particular, the posterior distribution is now of the form as shown in (5).

$$p(\theta_1, \theta_2, w_1, w_2 s_1, s_2, ..., s_N | y_N). \tag{5}$$

However, by using Gibbs sampling the s_i is separated and this leads to (6).

$$\begin{aligned} \int p(\theta_1, \theta_2, w_1, w_2, s_1, s_2, ..., s_N | y_N) ds_1 ... ds_N \\ = p(\theta_1, \theta_2, w_1, w_2 | y_N). \end{aligned} \tag{6}$$

Which results to the original posterior distribution. At this point that the latent indicators exist, let A_j denote a set of each and every i where $s_i = j$. Then, the set A_j collects the data-points all together based on the current indicator s_i.

Then, a Gibbs sampling technique can be derived where only three types of parameters are taken into consideration which are shown in (7).

$$\begin{aligned} p(\theta_1, \theta_2 | s_i, ..., s_N, w_1, w_2, y_N) &= Unknown, \\ p(w_1, w_2 | \mu_1, \mu_2, s_1, ...s_N, y_N) &= Unknown, \\ p(s_i | \mu_1, \mu_2, w_1, w_2, y_N) &= Unknown. \end{aligned} \tag{7}$$

Utilizing a Gamma prior on the $p(\theta_k) = Ga(a, b)$ and a flat prior on the s_i, we derive the following conditional posteriors as in (8).

$$\begin{aligned} \theta_k | s_i, ..., s_N, y_N \sim Ga\left(a + |A_k|, b + \sum_{i \in A_j} y_i\right), \\ w_1, w_2 | ... \sim Dir(|A_1|, |A_2|), \\ s_i | ... = Cat(p_1, p_2). \end{aligned} \tag{8}$$

The latent indicators s_i from (8) are drawn from a categorical distribution comprised by a parameter vector whose elements are the mainly the likelihood of observing y_i in each component. In particular, the very first component can have a likelihood of the form as shown in (9).

$$L_1 = \theta_1 w_1 exp(-\theta_1 y_i) \tag{9}$$

While the second parameter of (8) has a likelihood of the form as shown in (10).

$$L_2 = \theta_2 w_2 exp(-\theta_2 y_i) \tag{10}$$

These likelihoods are therefore normalized and produce elements of the probability vector of form (11).

$$\begin{aligned} p_1 = \frac{L_1}{L_1 + L_2}, \\ p_1 = \frac{L_2}{L_1 + L_2} \end{aligned} \tag{11}$$

The parameters w_k and s_i are inextricably linked due to the conjugacy of the Dirichlet distribution and the Categorical distribution; hence, we shall disregard the w_k for the sake of simplicity. Thus, a simpler two-part marginal Gibbs sampler is obtained which is shown in (12).

$$\theta_k \sim Ga\left(a + |A_k|, b + \sum_{i \in A_k} y_i\right), s_i \sim Cat(p_1, p_2). \tag{12}$$

Consequently, a Gibbs sampler for our latent variable model may be derived in two simple steps: first, we resample the component parameters θ_k given the current data labels, and then we resample the data labels s_i given the current model parameters. This approach will also hold true for models with far more complicated latent variables. Consequently, our whole Gibbs sampler may be implemented using the following steps.

```
For each loop:
resample the parameters given the labels
resample the labels given the parameters
```

Moreover, if we examine the necessities for each component of this for-loop, we can observe that each component can be computed rather simply in a distributed computing environment such as PySpark.

Specifically, one can first examine the resampling for the s_i data labels as in (13).

$$s_i \sim Cat(p_1, p_2), \tag{13}$$

where p_1 is the relative likelihood of detecting y_i from component 1, and p_2 is the relative likelihood of witnessing y_i from component 2. The present estimates of 1 and 2 may be used here to calculate these quantities; the only information required are the values of 1, 2, and y_i. We may then resample s_i for y_i as this demonstrates that the resampling of s_i is independent of all data-points other than y_i. This leads to the outcome that this step of the Gibbs sampler is extremely parallel and can be calculated utilizing a single $map()$ call.

Then, lets examine the resampling of the model parameters θ_k which is given in (14).

$$\theta_k \sim Ga\left(a + |A_k|, b + \sum_{i \in A_k} y_i\right). \tag{14}$$

Observe that, rather than using certain normalising constants, this posterior relies on a sum over a subset of the data as in (15).

$$\sum_{i \in A_k} y_i \tag{15}$$

To estimate θ_k for each k, the required information is just from the y_i where $i \in A_k$. While from this set of y_i, one can simply extract the necessary statistics to perform calculation; while in this example, merely their total and the size of the set is used. In the Spark application of the aforementioned method, we may observe two things. Initially, a $groupBy()$ function may be used to divide the data into groups based on their label s_i. Secondly, inside each group, adequate statistics may be obtained using a simple sum or count, both of which constitute a commutative monoid and are hence simply parallelizable and implementable with a $reduce()$ function. The $reduceByKey()$ function of Spark can efficiently perform both of these operations. Then, given adequate data for each group, it is easy to generate a new posterior sample and revise our estimates of the parameters w_k and θk for each group.

This type of Bayesian latent variable model can now be applied to large data sets with billions of observations using the $map()$, $reduceByKey()$, and other functions of Spark. At this point that everything required to conduct proper Gibbs sampling in Apache Spark is set, we may now proceed. This preceding example although is pretty rudimentary, it illustrates the fundamental structure of latent variable models that enables distributed parameter sampling. This Exponential Mixture Model example concludes with a PySpark application that computes what we have previously discussed.

B. Distributed inference for LDA

Although LDA and marginal Gibbs sampling have been extensively used, this approach is inapplicable for processing big document corpora because to its high computing cost. In addition, enhancing the speedup process via parallelization is not simple, since marginal Gibbs sampling is a strictly sequential procedure that samples a new topic assignment z_{ij} based on the present state of one variable. This section presents an approximate marginal Gibbs sampling approach for LDA, when data is dispersed over various PySpark processors and clusters.

1) Distributed LDA with PySpark: In order to distribute the computations, we follow the hypothesis proposed in [10] and assume that, since the number of words in a document is typically greater than the number of processors, the dependency between topic assignment z_{ij} and $z_{i'j'}$ is weak; consequently, the sequential sampling requirement can be relaxed, and the LDA model can be trained and tested in a distributed environment.

The primary function of the DLDA algorithm is shown in Algorithm 2. After random initialization, each document d is mapped to a record holding the document identifier, a sparse bag-of-words representation of the document's content, and the $c_{k,d,\cdot}$ count value corresponding to the document d. The collection of processed D documents is then partitioned using the usual Spark partitioning strategy and hashing function across P partitions. Finally, the usual LDA method with marginal Gibbs sampling is executed in parallel across all partitions, separately for each document, in order to update topic assignments **z**. Count values $c_{k,m,\cdot}$ and $c_{k,\cdot,\cdot}$ cannot be partitioned since they represent the overall state of the model. Consequently, $c_{k,\cdot,n}$ and $c_{k,\cdot,\cdot}$ are communicated to all nodes, and their local duplicates are used throughout the distributed sampling operation. Every n^{th} iteration, the count values $c_{k,\cdot,n}$ and $c_{k,\cdot,\cdot}$ are computed using the **z** reduction process and communicated back to the nodes. Iteration interval between the update of the global count values, n, is one of the the most fundamental parameters of the model.

C. Metropolis Hastings Random Walker

In this subsection a modified Metropolis Hastings-based random walker is introduced for unbiased sampling over Facebook network for text investigation of users' posts. Initially, consider each node seed u to be sampled. When the Metropolis Hastings algorithm draws a node v from $N(u)$ with probability $\frac{1}{d_u}$, the algorithm accepts v as a sample with probability min $(1, \frac{d_u}{d_v})$ and rejects it with probability 1-min$(1, \frac{d+u}{d_v})$ where $N(u)$ is a set of u neighbors and d_i is the degree of node i.

For the implementation, a specific percentage of seed nodes is initially selected and then random walker begins for each and every one. Because of this structure where it visits each node independently, the parallelism and distributedness of the algorithm occurs through PySpark. Ultimately, all the unique nodes in every iteration are inserted to the pool. The Algorithm for this implementation is given in Algorithm 3.

Algorithm 2 Distributed LDA with marginal Gibbs Sampling

```
begin
documents ← map(d: id, d.bow, c_{k,m,·}[d])
Initialize z at random and increase the count values
Update global counts c_{k,·,n}, c_{k,·,·}
Divide D documents into p partitions
for all i = 0 → iterations − 1 do
    Multi-cast global counts c_{k,·,n}, c_{k,·,·}
    for all p partitions distributed in parallel do
        Copy global counts c_{k,·,n}, c_{k,·,·}
        Perform LDA(D_p, z)
    end for
    z ← collect()
    Update global c_{k,·,n} ← z.reduce()
    Update global c_{k,·,·} ← z.reduce()
end for
end
```

Algorithm 3 Metropolis Hastings Random Walker

```
Let u ← seed node;
while Stop Condition is not satisfied do
    Pick a node v uniformly at random from u neighbors
    Construct a random value q ∈ [0, 1] uniformly
    if q ≤ (d_u/d_v)^α then
        u ← v;
    else
        Stay at u
    end if
end while
```

IV. Experimental Results

In this section, the results for the evaluation of the DLDA model effectiveness are presented and analyzed. The section initiates with comparing the performance of distributed and sequential LDA models, followed by investigating the influence of the *interval between updates* parameter on the model effectiveness.

Experiments were performed using three datasets containing texts available on Kaggle: CVPR 2019[1], NIPS papers[2] and A Million News Headlines[3]. Due to time and resource limitation, the experiments were conducted using a subset of each dataset. In the case of CVPR dataset, the whole dataset was used, while from NIPS papers we used all not missing abstracts. Note that for one of the experiments, full ABC dataset was employed. After tokenization and stopwords removal, each dataset was filtered for n most frequent words. Note that the value of n was determined experimentally, to assure that the runtime was reasonable taking into account the time restrictions. The characteristics of the datasets are presented in Table I.

The performance of the DLDA and sequential LDA models is assessed based on two criteria: the quality of the trained

[1]URL: kaggle.com/datasets/paultimothymooney/cvpr-2019-papers
[2]URL: kaggle.com/datasets/benhamner/nips-papers
[3]URL: kaggle.com/datasets/therohk/million-headlines

TABLE I: Characteristics of Datasets

Dataset	CVPR	NIPS	ABC
Number of documents	1294	9719	1213004
Number of distinct tokens	277	312	428
Total number of words	2070	15542	1997528

model, as measured by the perplexity metric [1], and the execution time. Initially, we generated perplexity values during training for both models using the ABC News headlines S and NIPS abstracts datasets for a variety of topic counts. Second, we assessed the execution time of model training and estimated the acceleration using the two aforementioned datasets and ABC News headlines. Experiments with sequential LDA were conducted on a local computer with an Intel Core i9-10850k 3.90 GHz processor and 32 GB of memory, while DLDA research was conducted on a server with 56 cores. To decrease server resource usage, 50 partitions were utilised in the experiments.

In order to determine the effect of the *interval between updates* parameter on DLDA performance, the model was trained with variable values for this parameter and a changing number of topics. The evaluative criteria are identical to those used in the first portion of the experimental results section.

A. Perplexity

The perplexity findings for ABC News headlines and NIPS abstracts which are illustrated in Figures 2,3, demonstrate that DLDA method often converges more slowly than the sequential LDA method, which is to be anticipated given that it relaxes the sequential sampling constraint. In the majority of instances, DLDA is able to attain a final perplexity value that is very similar to or the same as the standard LDA model, although it needs more iterations to do so. Observe that when issue dimensions increase (e.g., the amount of documents/topics/words), the distributed model requires more iterations to reach equivalent model quality to the sequential LDA, as shown by the divergent convergence rates of the three examined datasets.

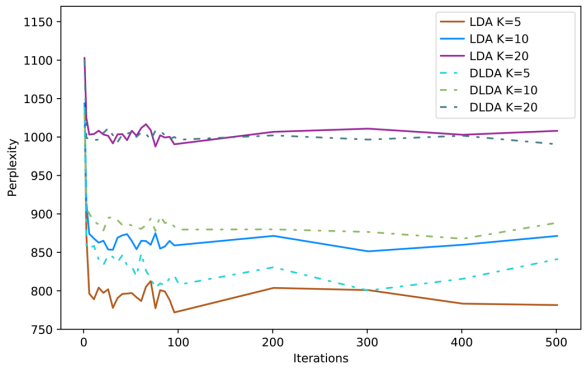

Fig. 2: Perplexity during training for different number of topics in ABC dataset.

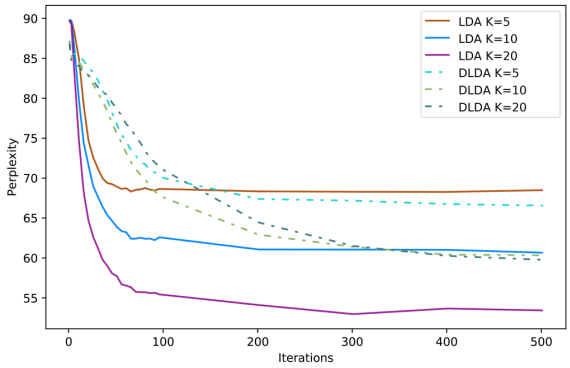

Fig. 3: Perplexity during training for different number of topics in NIPS dataset.

B. Execution time

TABLE II: Execution time in seconds and speedup achieved by DLDA with respect to different number of topics in CVPR.

Dataset	CVPR		
Topics	K = 5	K = 10	K = 20
LDA	47.80	51.59	57.41
DLDA	117.27	126.82	143.06
Speedup	0.408	0.407	0.401

TABLE III: Execution time in seconds and speedup achieved by DLDA with respect to different number of topics in NIPS.

Dataset	NIPS		
Topics	K = 5	K = 10	K = 20
LDA	1729.06	1779.65	2022.13
DLDA	128.03	159.32	207.01
Speedup	13.505	11.170	9.672

TABLE IV: Execution time in seconds and speedup achieved by DLDA with respect to different number of topics in ABC.

Dataset	ABC
Topics	K = 2
LDA	32557.60
DLDA	3929.08
Speedup	8.291

In Tables II,III,IV, the execution time and speedup for distributed and sequential models with regard to three datasets and varied numbers of topics are provided. Overall, it can be seen that employing DLDA to analyse tiny datasets increases the execution time when compared to normal LDA. Using DLDA increased execution time for the ABC News subset dataset by roughly thrice compared to the sequential approach. Given the complexity of initiating the Spark context, data distribution, and the time required for global count vales updates and broadcasts, DLDA cannot compete with sequential LDA since, although processing documents sequentially, it does not need extra variable synchronisation. Only when this model is used to medium- to large-sized datasets can the

speedup and performance enhancements of DLDA be noticed, as the parallel processing of documents drastically reduces the described overhead. For the NIPS abstracts dataset, the DLDA model produced a speedup of 13,505, 11,170, and 9,672 when the number of topics was 5, 10, and 20, respectively, however for the ABC News headlines, the distributed method was around eight times quicker than the sequential model.

C. *Interval between updates*

The parameter *interval between updates* is a crucial factor impacting the quality of the model and training time in the DLDA case. To examine the influence of this parameter, perplexity and execution time for the DLDA model were evaluated when the method was executed on the ABC News headlines S and NIPS abstracts datasets with parameter values of 5, 10, 20, and 50 iterations. Figures 4,5,6 present the results for the ABC dataset and 7,8,9 present the results for the NIPS dataset. The findings for the CVPR dataset were comparable to those for the NIPS dataset, and since certain sentences were missing from the dataset, we omitted them from the Figures presented here.

As anticipated, raising the amount of the *interval between updates* results in a decrease of the convergence rate, as the urge for sequential sampling is violated more and more. However, raising this amount also lowers execution time since it restricts the number of updates and broadcasts for global count values. In addition, it should be noted that this issue emerges when the dimensions within a dataset increase, therefore this parameter should be adjusted to a value that strikes a compromise between model quality and execution time.

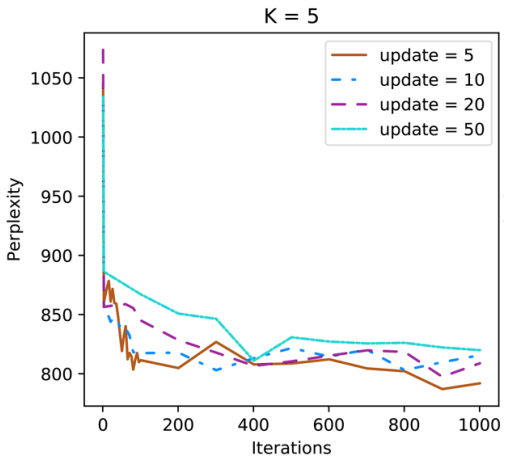

Fig. 4: Perplexity during training DLDA for ABC dataset of the *interval between updates* parameter for K=5.

Ultimately, the results for evaluating the Metropolis Hastings Random Walker in terms of running time for different number of topics are illustrated in Figure 10.

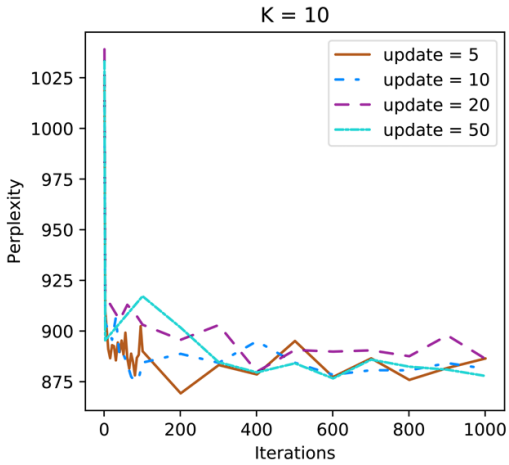

Fig. 5: Perplexity during training DLDA for ABC dataset of the *interval between updates* parameter for K=10.

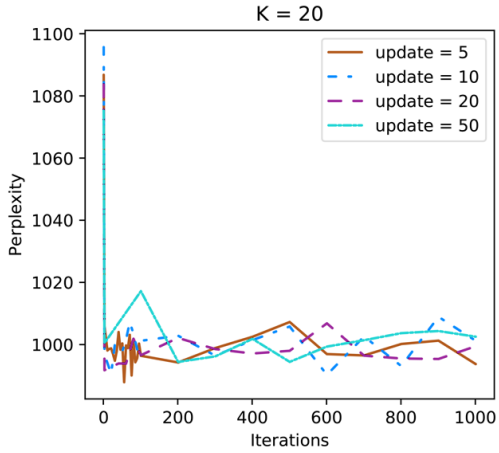

Fig. 6: Perplexity during training DLDA for ABC dataset of the *interval between updates* parameter for K=20.

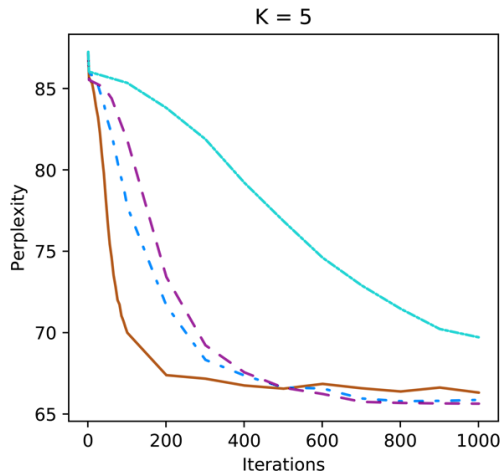

Fig. 7: Perplexity during training DLDA for NIPS abstracts of the *interval between updates* parameter for K=5.

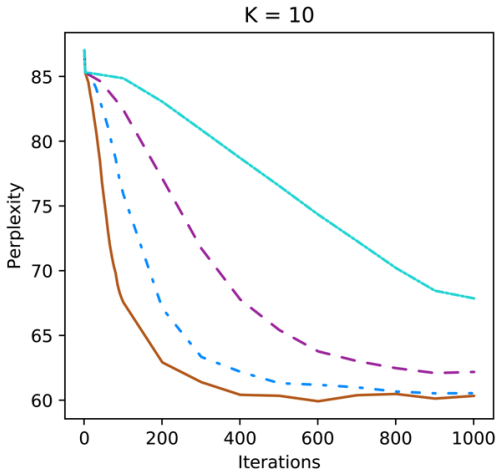

Fig. 8: Perplexity during training DLDA for NIPS abstracts of the *interval between updates* parameter for K=10.

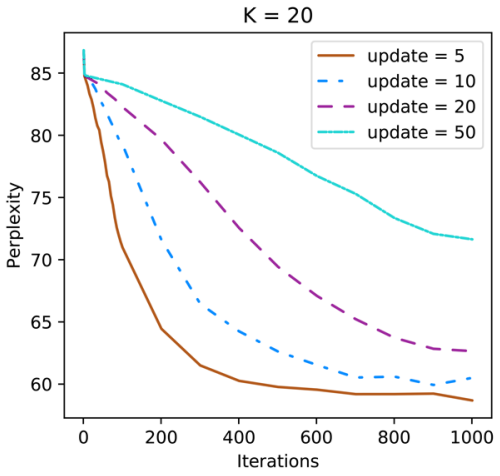

Fig. 9: Perplexity during training DLDA for NIPS abstracts of the *interval between updates* parameter for K=20.

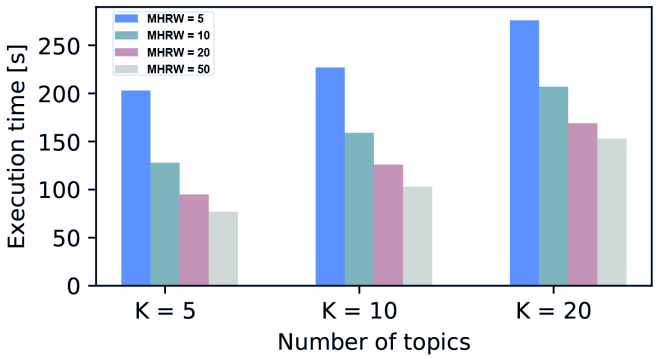

Fig. 10: Metropolis Hastings Random Walker time across different number of topics.

V. Conclusions and Future Work

In this paper, a distributed implementation of Gibbs Sampling is presented along with a method for implementing latent Dirichlet allocation model in distributed environments such as Apache Spark. To perform so, the dataset is split into P partitions and LDA is performed locally on each partition. Then, the resutls are derived with the use of Resilient Distributed Datasets (RDDs). Moreover, a Metropolis Hastings Random Walker is presented for graph exploration in the Facebook network. This method is capable to explore posts made by users within Facebook network so this could be an evaluation metric for the Gibbs sampling method presented, but this requires further investigation. The results of the experiments show that the proposed algorithm is able to achieve comparable results in terms of model quality when compared to the sequential LDA, while providing significant speedup due to using parallelized approximate Gibbs sampling procedure. For the first dataset used, the DLDA model achieved speedup of 13.505, 11.170 and 9.672 for number of topic equal to 5, 10 and 20 respectively, while for the second dataset the distributed algorithm was about 8 times faster than the sequential model. However, the current implementation on DLDA could be further improved by reducing the amount of data to be synchronized through a optimized partitioning and shuffling involving distribution of the data at the level of words.

Future directions of this work include modifications in the proposed algorithmic scheme as well as methods for utilizing alternative Monte Carlo methods. In particular, Hamiltonian Monte Carlo (HMC) [29]–[38] is an advanced and efficient alternative to sampling techniques such as Metropolis-Hastings and Gibbs Sampler. Similar to these other algorithms, Hamiltonian Monte Carlo is well-suited to Bayesian statistical approaches since it just needs the kernel of the target distribution. By assigning a given random variable to each dimension of the target random variable that we wish to sample, as well as requiring that the target and momentum satisfy Hamilton's equations for a given time step, Hamiltonian Monte Carlo forces our probability space to behave as if it were a random physical system. Moreover, this construction accelerates the Markov chain through the probability space in potentially improbable directions while still allowing for correction via memory conservation, allowing the chain to traverse regions of low probability that may have been an impediment to mixture when using a naive algorithm like Metropolis-Hastings, which gravitates almost exclusively towards regions of high probability density. As shown in the proposed framework, the mobility given by DLDA makes it well suited for sampling from distributions that need large traversal lengths in the Markov chain, such as multimodal or high-dimensional distributions. Moreover, ensemble learning methods as in [39] or optimization schemes for query expansion [40] or p2p network simulations [41] [42] can be utilized for possible integration with this work for further fine-tuning adjustments. Each of the preceding directions is set to be implemented in future work.

REFERENCES

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[2] U. Chauhan and A. Shah, "Topic modeling using latent Dirichlet allocation: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 7, pp. 1–35, 2021.

[3] I. Sutherland, Y. Sim, S. K. Lee, J. Byun, and K. Kiatkawsin, "Topic modeling of online accommodation reviews via latent dirichlet allocation," *Sustainability*, vol. 12, no. 5, p. 1821, 2020.

[4] E. S. Negara, D. Triadi, and R. Andryani, "Topic modelling twitter data with latent dirichlet allocation method," in *2019 International Conference on Electrical Engineering and Computer Science (ICECOS)*. IEEE, 2019, pp. 386–390.

[5] Y. Wang, Y. Tong, and D. Shi, "Federated latent dirichlet allocation: A local differential privacy based framework," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 6283–6290.

[6] W. Wang, Y. Feng, and W. Dai, "Topic analysis of online reviews for two competitive products using latent Dirichlet allocation," *Electronic Commerce Research and Applications*, vol. 29, pp. 142–156, 2018.

[7] J. Rieger, C. Jentsch, and J. Rahnenführer, "RollingLDA: An update algorithm of Latent Dirichlet Allocation to construct consistent time series from textual data," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, 2021, pp. 2337–2347.

[8] J. Joung and H. M. Kim, "Automated keyword filtering in latent dirichlet allocation for identifying product attributes from online reviews," *Journal of Mechanical Design*, vol. 143, no. 8, 2021.

[9] Y. Wang and J. E. Taylor, "DUET: Data-driven approach based on latent Dirichlet allocation topic modeling," *Journal of Computing in Civil Engineering*, vol. 33, no. 3, 2019.

[10] D. Newman, A. Asuncion, P. Smyth, and M. Welling, "Distributed algorithms for topic models," *Journal of Machine Learning Research*, vol. 10, no. Aug, pp. 1801–1828, 2009.

[11] B. Zhao, H. Zhou, G. Li, and Y. Huang, "ZenLDA: Large-scale topic model training on distributed data-parallel platform," *Big Data Mining and Analytics*, vol. 1, no. 1, pp. 57–74, 2018.

[12] J. Chen, J. Zhu, J. Lu, and S. Liu, "Scalable training of hierarchical topic models," *Proceedings of the VLDB Endowment*, vol. 11, no. 7, pp. 826–839, 2018.

[13] A. R. Pathak, M. Pandey, and S. Rautaray, "Adaptive model for dynamic and temporal topic modeling from big data using deep learning architecture," *International Journal of Intelligent Systems and Applications*, vol. 11, no. 6, p. 13, 2019.

[14] C. Karras, A. Karras, and S. Sioutas, "Pattern recognition and event detection on iot data-streams," *arXiv preprint arXiv:2203.01114*, 2022.

[15] C. Karras, A. Karras, G. Drakopoulos, K. Tsakalidis, P. Mylonas, and S. Sioutas, "Weighted Reservoir Sampling On Evolving Streams: A Sampling Algorithmic Framework For Stream Event Identification," in *Proceedings of the 12th Hellenic Conference on Artificial Intelligence*, ser. SETN '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3549737.3549767

[16] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[17] C. Karras and A. Karras, "Dbsop: An efficient heuristic for speedy mcmc sampling on polytopes," *arXiv preprint arXiv:2203.10916*, 2022.

[18] C. Karras, A. Karras, M. Avlonitis, and S. Sioutas, "An Overview of MCMC Methods: From Theory to Applications," in *Artificial Intelligence Applications and Innovations. AIAI 2022 IFIP WG 12.5 International Workshops*, I. Maglogiannis, L. Iliadis, J. Macintyre, and P. Cortez, Eds. Cham: Springer International Publishing, 2022, pp. 319–332.

[19] C. Karras, A. Karras, M. Avlonitis, I. Giannoukou, and S. Sioutas, "Maximum Likelihood Estimators on MCMC Sampling Algorithms for Decision Making," in *Artificial Intelligence Applications and Innovations. AIAI 2022 IFIP WG 12.5 International Workshops*, I. Maglogiannis, L. Iliadis, J. Macintyre, and P. Cortez, Eds. Cham: Springer International Publishing, 2022, pp. 345–356.

[20] C. P. Robert, V. Elvira, N. Tawn, and C. Wu, "Accelerating MCMC algorithms," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 10, no. 5, p. e1435, 2018.

[21] J. S. Speagle, "A conceptual introduction to markov chain monte carlo methods," *arXiv preprint arXiv:1909.12313*, 2019.

[22] M. Titsias and P. Dellaportas, "Gradient-based adaptive markov chain monte carlo," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[23] D. Van Ravenzwaaij, P. Cassey, and S. D. Brown, "A simple introduction to Markov Chain Monte–Carlo sampling," *Psychonomic bulletin & review*, vol. 25, no. 1, pp. 143–154, 2018.

[24] L. Martino, V. Elvira, and G. Camps-Valls, "Group importance sampling for particle filtering and MCMC," *Digital Signal Processing*, vol. 82, pp. 133–151, 2018.

[25] D. Luengo, L. Martino, M. Bugallo, V. Elvira, and S. Särkkä, "A survey of Monte Carlo methods for parameter estimation," *EURASIP Journal on Advances in Signal Processing*, vol. 2020, no. 1, pp. 1–62, 2020.

[26] L. Martino and V. Elvira, "Compressed Monte Carlo with application in particle filtering," *Information Sciences*, vol. 553, pp. 331–352, 2021.

[27] L. Martino, R. Casarin, F. Leisen, and D. Luengo, "Adaptive independent sticky MCMC algorithms," *EURASIP Journal on Advances in Signal Processing*, vol. 2018, no. 1, pp. 1–28, 2018.

[28] L. Martino, V. Elvira, D. Luengo, J. Corander, and F. Louzada, "Orthogonal parallel MCMC methods for sampling and optimization," *Digital Signal Processing*, vol. 58, pp. 64–84, 2016.

[29] M. Betancourt, "A conceptual introduction to Hamiltonian Monte Carlo," *arXiv preprint arXiv:1701.02434*, 2017.

[30] T. Chen, E. Fox, and C. Guestrin, "Stochastic gradient hamiltonian monte carlo," in *International conference on machine learning*. PMLR, 2014, pp. 1683–1691.

[31] M. Betancourt and M. Girolami, "Hamiltonian Monte Carlo for hierarchical models," *Current trends in Bayesian methodology with applications*, vol. 79, no. 30, pp. 2–4, 2015.

[32] A. Campbell, W. Chen, V. Stimper, J. M. Hernandez-Lobato, and Y. Zhang, "A gradient based strategy for hamiltonian monte carlo hyperparameter optimization," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1238–1248.

[33] A. D. Cobb and B. Jalaian, "Scaling Hamiltonian Monte Carlo inference for Bayesian neural networks with symmetric splitting," in *Uncertainty in Artificial Intelligence*. PMLR, 2021, pp. 675–685.

[34] W. T. Mongwe, R. Mbuvha, and T. Marwala, "Magnetic Hamiltonian Monte Carlo with partial momentum refreshment," *IEEE Access*, vol. 9, pp. 108 009–108 016, 2021.

[35] C. Mak, F. Zaiser, and L. Ong, "Nonparametric Hamiltonian Monte Carlo," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7336–7347.

[36] G. Zhou, "Mixed Hamiltonian Monte Carlo for mixed discrete and continuous variables," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17 094–17 104, 2020.

[37] N. Bou-Rabee and J. M. Sanz-Serna, "Randomized hamiltonian monte carlo," *The Annals of Applied Probability*, vol. 27, no. 4, pp. 2159–2194, 2017.

[38] N. Bou-Rabee, A. Eberle, and R. Zimmer, "Coupling and convergence for Hamiltonian monte carlo," *The Annals of applied probability*, vol. 30, no. 3, pp. 1209–1250, 2020.

[39] A. Karras and C. Karras, "Integrating User and Item Reviews in Deep Cooperative Neural Networks for Movie Recommendation," *arXiv preprint arXiv:2205.06296*, 2022.

[40] C. Karras, A. Karras, L. Theodorakopoulos, I. Giannoukou, and S. Sioutas, "Expanding Queries with Maximum Likelihood Estimators and Language Models," in *Proceedings of the ICR'22 International Conference on Innovations in Computing Research*, K. Daimi and A. Al Sadoon, Eds. Cham: Springer International Publishing, 2022, pp. 201–213.

[41] A. Karras, C. Karras, K. C. Giotopoulos, I. Giannoukou, D. Tsolis, and S. Sioutas, "Download Speed Optimization in P2P Networks Using Decision Making and Adaptive Learning," in *Proceedings of the ICR'22 International Conference on Innovations in Computing Research*, K. Daimi and A. Al Sadoon, Eds. Cham: Springer International Publishing, 2022, pp. 225–238.

[42] C.-P. Balatsouras, A. Karras, C. Karras, D. Tsolis, and S. Sioutas, "Wichord: A chord protocol application on p2p lora wireless sensor networks," in *2022 13th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 2022, pp. 1–8.