Distributing the Stochastic Gradient Sampler for Large-Scale LDA

Yuan Yang[†‡], Jianfei Chen[†], Jun Zhu[†*]
[†]Dept. of Comp. Sci. & Tech., State Key Lab. of Intell. Tech. & Sys., TNList Lab, Tsinghua University, Beijing, China
[‡]College of Software Engineering, Beihang University, Beijing 100191, China
mblackout@hotmail.com, chenjian14@mails.tsinghua.edu.cn, dcszj@tsinghua.edu.cn

ABSTRACT

Learning large-scale Latent Dirichlet Allocation (LDA) models is beneficial for many applications that involve large collections of documents.Recent work has been focusing on developing distributed algorithms in the batch setting, while leaving stochastic methods behind, which can effectively explore statistical redundancy in big data and thereby are complementary to distributed computing.The distributed stochastic gradient Langevin dynamics (DSGLD) represents one attempt to combine stochastic sampling and distributed computing, but it suffers from drawbacks such as excessive communications and sensitivity to partitioning of datasets across nodes. DSGLD is typically limited to learn small models that have about 10^3 topics and 10^3 vocabulary size.

In this paper, we present *embarrassingly parallel SGLD* (EPSGLD), a novel distributed stochastic gradient sampling method for topic models. Our sampler is built upon a divide-and-conquer architecture which enables us to produce robust and asymptotically exact samples with less communication overhead than DSGLD. We further propose several techniques to reduce the overhead in I/O and memory usage. Experiments on Wikipedia and ClueWeb12 documents demonstrate that, EPSGLD can scale up to large models with 10^{10} parameters (i.e., 10^5 topics, 10^5 vocabulary size), four orders of magnitude larger than DSGLD, and converge faster.

CCS Concepts

•**Mathematics of computing** → **Probabilistic algorithms;**
•**Computing methodologies** → **Distributed algorithms;**

Keywords

large-scale LDA, stochastic gradient sampler, embarrassingly parallel MCMC

1. INTRODUCTION

Topic models are useful statistical tools for mining latent topic representations in document data, with Latent Dirichlet Allocation (LDA) [8] as one of the most popular examples. LDA has been used in various tasks, including text classification [27], information retrieval [21], recommendation [10] and social network analysis [3, 5]. Since exact inference is intractable, various approximate methods have been developed, with Markov Chain Monte Carlo (MCMC) as a main workhorse. A MCMC method builds a sampler that simulates samples from a proposal distribution and accepts each sample with a rate specified by its posterior probability given observed data. A conventional MCMC method adopts batch update, that is, the whole dataset is processed for a single iteration. Apparently, the overhead for such batch methods increases significantly as the model size and/or data size get larger.

There are two ways to scale up inference for topic models — *stochastic subsampling* and *distributed computing*. Stochastic subsampling reduces the per-iteration cost by randomly drawing a subset (or mini-batch) of documents and constructing an (unbiased) estimate of the data statistics (e.g., gradients). Such methods save computation cost by effectively exploring the statistical redundancy that is commonly observed in large-scale datasets. For topic models, representative work includes stochastic variational inference (SVI) [9] and stochastic gradient MCMC, such as stochastic gradient Riemannian Langevin dynamics (SGRLD) [17], an extension of stochastic gradient Langevin dynamics (SGLD) [23] for LDA (See [28] for an overview). Compared to SVI, which often relies on some restricting mean-field assumption and has dense per-token update for LDA, we focus on SGLD methods, which converge to the target posterior distribution under certain scheme of annealing the step sizes and has sparse per-token update, a desirable property for learning a large number of topics.

Distributed methods parallelize the computation on the full dataset over multiple compute nodes. Representative work for topic models includes Yahoo!LDA [1], LightLDA [25] and many others [4, 13, 12, 16], which follow a common pattern that the globally shared topic-word matrix is updated asynchronously over multiple workers, and workers constantly communicate with a master node to keep the local copy of the topic-word matrix up-to-date through some operations, e.g., a parameter server in Yahoo!LDA or pull/push-ing a matrix block from/to the master node in LightLDA. In either case the write operation is exclusive for any element in the matrix, which means only one worker at a time can modify the element (See Fig. 1(b) for an illustration of LightLDA). Various efforts have been made on reducing the per-token sampling complexity for better efficiency on learning large-scale LDA models [24, 11], with the recent success on an $\mathcal{O}(1)$ sampler [25]. However, these methods rely heavily on the communications among compute nodes and can have substantial latency in waiting to gain the lock of element, which can seriously slow down the convergence.

Though fast per-token samplers have been extensively investigated in a sophisticated distributed system, little work has been

*Corresponding author.

KDD '16, August 13-17, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2939672.2939821

done on investigating stochastic methods within a distributed paradigm to jointly explore data redundancy and distributed computing. One exception is distributed SGLD (DSGLD) [2]. In DSGLD, the dataset is split and stored locally on multiple compute nodes, and each individual SGLD sampler moves from one node to another after each iteration in order to visit the local corpus there. This behavior forms a unique trajectory (or chain) for the sampler across nodes. The distribution of computation is achieved by running multiple chains at the same time, and exchanging them among nodes after each iteration. Fig. 1(a) illustrates DSGLD, where the topic-word distribution (a core part for LDA) must be carried together with the model. There are two significant drawbacks in DSGLD. First, since the model needs to be carried to the next node after each iteration, the number of communications grows linearly to the number of chains and the number of iterations. For large-scale LDA, where the model size can be as large as 10^{10} (e.g., $K = 10^5$ topics and $V = 10^5$ unique words), such overhead is prohibitive. Hence, DSGLD is often limited to small-scale applications (e.g., $V = 10^3$ and $K = 10^3$) [2]. Second, DSGLD makes use of a *chain coupling* technique to average over the samples drawn from individual samplers in order to aggregate into global samples. This strategy can generally reduce the estimation variance, however, since samples are drawn from different subset posteriors (or sub-posteriors), such a vanilla averaging operation can lead to bad aggregation if subset posteriors differ significantly. In other words, DSGLD is sensitive to noise or non-random partitioning of the datasets, and this can eventually lead to a slow convergence.

To address the above challenges, we present *embarrassingly parallel SGLD* (EPSGLD), a novel distributed version of SGLD samplers for topic models. EPSGLD has less communication overhead and at the same time overcomes all drawbacks in DSGLD to achieve better scalability. We build our algorithm based on the recent progress on divide-and-conquer (or embarrassingly parallel) sampling methods, which allow worker nodes to draw samples only from their sub-posteriors, and then approximate the full-data posterior (true posterior) samples in master node using some aggregation methods, such as weighted average [15], kernel density estimators [19], and posterior median [14] (See [28] for an overview). One of the central feature for the embarrassingly parallel methods is that individual workers need no/few communications when sampling from sub-posteriors, which significantly reduces the network I/O and latency. This feature is excellent for models that contain too many parameters to be carried across compute nodes, and can help to increase the scalability of distributed methods. In our implementation, EPSGLD can scale up to $V = 10^5$, $K = 10^5$ (or even larger), which is four orders of magnitude larger than DSGLD, while the communication cost grows at $\mathcal{O}(I^{\frac{1}{3}})$ rate, where I is the number of iterations. On the other hand, the aggregation methods produce asymptotically exact samples from the true posterior [15], and are, therefore, less sensitive to noise in datasets, leading to a significantly faster convergence than DSGLD.

Technically, we address two challenging problems to develop EPSGLD for large-scale LDA. First, despite that the embarrassingly parallel methods can work without communication, the synchronization across local LDA models is still needed due to two reasons: (1) the global variables (i.e., topic-word distribution) in LDA depend on the assignments to all the local variables (i.e., topic assignments for all words in local datasets); and (2) in order to draw samples for local variables, samplers need to access the latest global variables that are usually generated using aggregation methods and stored in the master node [1]. This synchronization is important for distributed LDA models to converge to the optimal solution, and Newman et al. suggested to synchronize after each iteration of the local sampler [16]. For EPSGLD, this practice can again lead to $\mathcal{O}(I)$ communication cost, and is thus not suitable. To avoid frequent synchronizations, we investigate the properties of embarrassing parallel methods, and find that they can correct the sub-posteriors towards the true posterior as the training proceeds, and this phenomenon can be characterized as an Expectation-maximization (EM) process. In that case, EPSGLD can converge to the optimal solution with significantly fewer synchronizations. In the experiments, we conduct a series of carefully designed tests to demonstrate the power of this property, and show how the performance of EPSGLD can be influenced by the frequency of synchronizations.

Second, though various methods have been proposed to increase the scalability of the distributed batch samplers, such as fast sampling algorithms [24, 11, 25] and model-parallelism [26, 25], such work is still lacking for stochastic samplers. We fills up this research void and discuss several memory and I/O strategies that can improve the model scalability. For example, some embarrassingly parallel methods such as Weierstrass sampler [19] require to store all the global samples in each worker node in order to perform aggregation, which lead to excessive usage of disk and memory as the space complexity grows linearly. We propose to take advantage of synchronization operation and only hold the latest copy of the global samples in each worker node, resulting in an $\mathcal{O}(1)$ space complexity in disk. We also note that the number of unique words contained in each mini-batch is sufficiently smaller than the vocabulary size, and we propose the matrix slicing technique—for each iteration, we only load into memory those word-slices that will be used by the SGLD sampler, saving a great amount of memory space. Additionally, we also show how fast sampling algorithms can be integrated into SGLD samplers. All these techniques help EPSGLD to scale up to models with 10^{10} parameters.

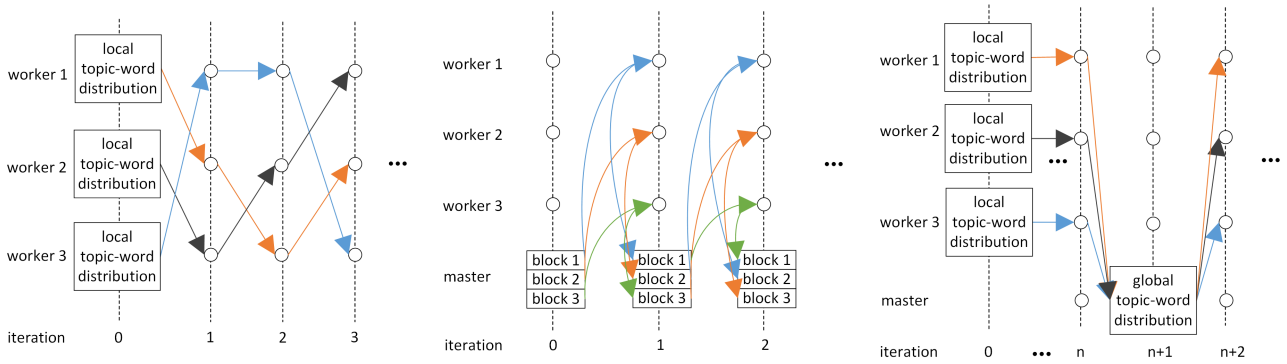
To summarize, we propose EPSGLD, a novel distributed stochastic gradient MCMC sampler, to learn large-scale LDA models. Built upon an embarrassingly parallel method, EPSGLD can produce asymptotically exact samples with significantly fewer synchronizations than DSGLD. We also propose strategies to reduce communication overhead and memory usage to increase scalability. The experiments demonstrate that EPSGLD can scale up to LDA models with 10^{10} parameters, where the number of communications can be reduced to $\mathcal{O}(I^{\frac{1}{3}})$. We also compare with LightLDA, a state-of-the-art distributed batch method. Our results show that EPSGLD converges faster at the beginning, and reaches a comparable optimum with LightLDA at the scale of $V = 10^5$ and $K = 10^5$.

Outline: We first introduce the preliminaries about LDA, SGLD and distributed sampling. Then we present EPSGLD and solutions to the above two issues. Section 4 conducts a set of experiments to evaluate our method and compare with state-of-the-art distributed samplers such as DSGLD and LightLDA. Finally, we conclude.

2. PRELIMINARIES

2.1 Latent Dirichlet Allocation

LDA is a probabilistic generative model for topic discovery in text documents. It describes how the words in documents are explained by a set of K topics, each of which is a V-dimensional topic-word distribution φ_k, where V is the vocabulary size. The topics are often assumed to follow a conjugate Dirichlet prior, that is, $\varphi_k \sim Dir(\beta)$, with hyperparameter β. For each document $\mathbf{w}_d$ that contains N_d tokens, where each token in it is denoted as w_{dn}, a K-dimensional topic mixing distribution γ_d is sampled from a Dirchlet prior $Dir(\alpha)$. Then, for each token, a topic assignment



(a) (b) (c)

Figure 1: Illustrations of (a) DSGLD: 3 chains are deployed into 3 workers; each chain randomly visits next worker after finishing the current update; (b) LightLDA: it splits the topic-word counts matrix into multiple blocks, each worker pulls one block at a time and updates it using its local data; and (c) EPSGLD: each worker updates its local model independently, then after certain steps, the master collects the samples to aggregate them into global one and sends back to all workers.

z_{dn} is sampled from a multinomial distribution $z_{dn} \sim \text{Multi}(\gamma_d)$, followed by sampling the token itself again from a multinomial distribution $w_{dn} \sim \text{Multi}(\varphi_{z_{dn}})$. The matrix that contains all φ_k is denoted as Φ, and the one that contains all γ_d is Γ.

The inference problem for LDA is to determine the posterior distribution $P(\mathbf{z}, \Phi, \Gamma|\mathbf{w})$, where $\mathbf{w}$ and $\mathbf{z}$ denote all the tokens and their corresponding topic assignments in the whole dataset respectively. However, the exact inference is intractable. The solution to this is to use approximate inference methods such as Gibbs sampling which is a special case of MCMC simulation [7]. By exploring the conjugacy, we can do collapsed Gibbs sampling [6] by integrating out Φ and Γ. Namely, we have the collapsed posterior $P(\mathbf{z}|\mathbf{w}) \propto P(\mathbf{z}|\alpha)P(\mathbf{w}|\mathbf{z}, \beta)$, where the terms are evaluated as $P(\mathbf{w}|\mathbf{z}, \beta) = \int P(\mathbf{w}|\mathbf{z}, \Phi)P(\Phi|\beta)d\Phi$ and $P(\mathbf{z}|\alpha) = \int P(\mathbf{z}|\Gamma)P(\Gamma|\alpha)d\Gamma$. Then, a collapsed Gibbs sampler iterates over all the tokens and draws the topic assignment for each token w_{dn} from the local conditional distribution $P(z_{dn}|rest, \mathbf{w})$, where $rest$ denotes the rest topic assignments excluding z_{dn}. By some algebra, we can show that the local conditional distribution is:

$$P(z_{dn} = k|rest) \propto \frac{(\alpha + n_{dk}^{-z_{dn}})(\beta + n_{kw}^{-z_{dn}})}{\sum_k \beta + n_{kw}^{-z_{dn}}}, \qquad (1)$$

where n_{dk} denotes the number of times that topic k is assigned to document d, n_{kw} denotes the number of times that topic k is assigned to word w, superscript $-z_{dn}$ denotes the counts matrix without z_{dn}, and we omit the condition $\mathbf{w}$ for simplicity. Note that token w_{dn} is different from word w: different tokens can refer to the same word, but the word w is unique and the number of unique words is V. In this process, one updates the counts matrices n_{dk} and n_{kw} according to the sampling results, and after a sufficiently large number of iterations, we can normalize the counts matrices by row to obtain the Γ and Φ.

2.2 Fast per-token sampling algorithms

In the standard form, Eq. (1) takes $\mathcal{O}(K)$ time to sample a new topic z_{dn} for token w_{dn}. A variety of methods have been proposed to reduce this per-token sampling complexity. Here we briefly introduce three of them. SparseLDA [24] decomposes Eq. (1) into three terms:

$$P(z_{dn} = k|rest) \propto \frac{\alpha\beta}{\sum_k \beta + n_{kw}^{-z_{dn}}} + \frac{n_{dk}^{-z_{dn}}\beta}{\sum_k \beta + n_{kw}^{-z_{dn}}} + \frac{(\alpha + n_{dk}^{-z_{dn}})n_{kw}^{-z_{dn}}}{\sum_k \beta + n_{kw}^{-z_{dn}}}.$$

When sampling, one first uniformly chooses one of the three probability buckets, and then samples the topic from that bucket. If the second or third term is chosen, it takes $\mathcal{O}(K_d)$ or $\mathcal{O}(K_w)$ time to sample z_{dn}, where K_d is the number of different topics that document d has, and K_w is the number of different topics to which word w is assigned. Since LDA is often sparse, the complexity $\mathcal{O}(K_d + K_w)$ is much lower than $\mathcal{O}(K)$. AliasLDA [11] presents another way to decompose Eq. (1):

$$P(z_{dn} = k|rest) \propto \frac{\alpha(\beta + n_{kw})}{\sum_k \beta + n_{kw}} + \frac{n_{dk}^{-z_{dn}}(\beta + n_{kw}^{-z_{dn}})}{\sum_k \beta + n_{kw}^{-z_{dn}}},$$

where the second term is sparse and can be sampled with $\mathcal{O}(K_d)$ time, and the first term uses a stale copy of n_{kw}, out of which one can build an alias table and sample from it with $\mathcal{O}(1)$ time. Since the sampler draws topic from a stale copy, an extra Metropolis process is used to correct the bias in the samples. Recently, LightLDA reduces the complexity to $\mathcal{O}(1)$. It decomposes Eq. (1) as:

$$P(z_{dn} = k|rest) \propto (\alpha + n_{dk}^{-z_{dn}}) \times \frac{(\beta + n_{kw})}{\sum_k \beta + n_{kw}}.$$

Instead of choosing the bucket, it samples topic from either the first term or the second term at one time, and chooses the other term the next time. While the second term is sampled through an alias table, the first term can also be sampled with $\mathcal{O}(1)$ time as long as it keeps track of the topic assignments list $z_d = \{z_{d1}, \cdots, z_{dN_d}\}$ for document d (as it is often the case in practice), reaching the $\mathcal{O}(1)$ overall complexity.

2.3 Stochastic gradient Langevin dynamics

As the data size gets extremely large, the time for running one iteration in batch methods can be unacceptable. One approach to scaling up the inference process is to use stochastic subsampling. Due to the statistical redundancy in big data, the idea has proven to be effective in various SGLD samplers [23, 17]. However, the vanilla SGLD cannot be readily applied to LDA due to two reasons [17]: (1) the probability simplex that defines LDA has its own boundary, such that a gradient step can get outside of the simplex; and (2) the distribution for LDA can be very skewed, and the vanilla SGLD is likely to be insufficient in exploring the probability space. To address the problems, the stochastic gradient Riemannian Langevin dynamics (SGRLD) [17] sampler for LDA was proposed. Here we briefly introduce SGRLD, and for simplicity, we do not distinguish between SGLD and SGRLD in the sequel.

Let $\mathbf{w}^{(s)}$ denote the mini-batch sampled from $\mathbf{w}$ at the sth iteration, and the numbers of documents in $\mathbf{w}^{(s)}$ and $\mathbf{w}$ are denoted as

$D^{(s)}$ and D respectively. The inference in SGRLD is performed on the unnormalized topic-word matrix T, where we compute the gradient and update T with it in each iteration. In T, the distribution of topic k and its probability mass of each word w is denoted as t_k and t_{kw} (Correspondingly, we also have φ_k and φ_{kw}). The prior of T (topic k and word w) is given as $P(t_{kw}) = Gamma(1, 1)$. Thus the normalized distribution φ_k is obtained from $\varphi_k = \frac{t_k}{\sum_w t_{kw}}$. At each iteration s, T is updated as follows:

$$t_{kw}^{(s+1)} \leftarrow |t_{kw}^{(s)} + \frac{\epsilon^{(s)}}{2}(\beta - t_{kw}^{(s)} + \frac{D}{D^{(s)}} \sum_{d:\ \mathbf{w}_d \in \mathbf{w}^{(s)}} \mathbb{E}_{z_{dn}|rest,\Phi}[n_{dkw} - \varphi_{kw} n_{dk}]) + (t_{kw}^{(s)})^{\frac{1}{2}} \eta_{kw}|, \tag{2}$$

where $\epsilon^{(s)}$ denotes the step size at the sth iteration and n_{dkw} denotes the number of times topic k is assigned to w in document d. η_{kw} is obtained from $\eta_{kw} \sim \mathcal{N}(0, \epsilon^{(s)})$ and $d : \mathbf{w}_d \in \mathbf{w}^{(s)}$ denotes each document in the mini-batch. The expectation $\mathbb{E}_{z_{dn}|rest,\Phi}$ is not analytically tractable, but can be estimated by performing the collapsed Gibbs sampling on distribution $P(z_{dn}|rest, \Phi)$ over the mini-batch data

$$P(z_{dn} = k|rest, \Phi) \propto (\alpha + n_{dk}^{-z_{dn}})\varphi_{kw}.$$

In this case, two temporary counts matrices n_{dk} and n_{kw} are created to record the updates in topic assignments. Then these two matrices are used in Eq. (2) to compute the expectation. In other words, SGLD still uses collapsed Gibbs sampling, but with a smaller size of documents, which helps to reduce the computation cost. Therefore, SGLD is compatible with the alias table technique used in AliasLDA and LightLDA, which means SGLD can also draw samples with $\mathcal{O}(1)$ complexity.

2.4 Distributed LDA and embarrassingly parallel

The other approach to increasing the scalability is distributed computing. The distributed systems for batch samplers have been extensively studied in order to learn large-scale LDA models [1, 25, 4, 13, 12, 16]. Despite of different techniques and implementations proposed in each method, the overall process is similar. In Yahoo!LDA, the global counts matrix n_{kw} is stored in a master node; each worker node i maintains a local copy of it $\tilde{n}_{kw,i}$ and its own local counts matrix $n_{kw,i}$. We note that the local copy $\tilde{n}_{kw,i}$ can be different from that in the master node (i.e., n_{kw}). This is referred to as the *delayed update* technique, where the worker can delay the synchronization of n_{kw} with the master for a certain number of iterations in order to reduce communications. For each worker, it updates $n_{kw,i}$ with Eq. (1); and the master synchronizes and updates the global matrtix n_{kw} using

$$n_{kw} \leftarrow n_{kw} + \sum_i (\tilde{n}_{kw,i} - n_{kw,i}). \tag{3}$$

Another similar method is proposed in LightLDA. As shown in Fig.1(b), the global matrix is split into multiple blocks along the word-axis. For every worker, it pulls a block from master, updates it with its local data and sends it back to the master node. During this process, the block in master is locked until it is returned by the worker, so that these blocks can be updated asynchronously without conflicts. Hence, LightLDA is referred to as a *model-parallelism* method, where it partitions the model and storing a part of the model on each node; then, partial updates are carried out on each node [28]. In these methods, synchronizations play a central role in counts matrix update, and they expect equality between the global matrix and its per-machine copies after a complete synchronization.

Another parallel design is the *embarrassingly parallel* method, which assumes that there is a generative process that leads to divergent copies of the same random variables in each node [1]. This method follows several criteria: (1) each node only has access to its local data; (2) each node performs MCMC independently, without communication; and (3) samples from each worker are aggregated using approximate methods which yield asymptotically exact samples from the full-data posterior [15]. Formally, if we develop embarrassingly parallel method based on the SGRLD sampler, the sub-posterior samples for LDA should be T_is (local unnormalized topic-word matrices); then the global unnormalized topic-word matrix which is aggregated from those T_is is denoted as Θ and the true (full-data) posterior is therefore denoted as $P(\Theta|\mathbf{w})$. Suppose there are n workers and we split our data $\mathbf{w}$ into $\{\mathbf{w}_1, \cdots, \mathbf{w}_n\}$. Then, we have

$$P(\Theta|\mathbf{w}) \propto \prod P(\Theta|\mathbf{w}_i), \tag{4}$$

where $\mathbf{w}_i$ is the subset at node i and each term $P(\Theta|\mathbf{w}_i)$ is a *sub-posterior* [15]. Note that we neglect the local variables $\mathbf{z}$ and Γ since they are documents-related and are unnecessary to aggregate globally. Eq. (4) suggests that the posterior of full data can be represented by the product of sub-posteriors. Sampling from this product of posteriors remains the main difficulty in the embarrassingly parallel design [19]. To do this, various approximate methods have been proposed [15, 14, 19], such as weighted average and kernel density estimators. These methods are excellent for models with a large amount of parameters, since they allow the update on model to be performed without communication. But just like the case for SGLD and SGRLD, these frameworks are not readily applicable to LDA—two problems need to be addressed, and Section 3 is dedicated to solving these problems.

3. EMBARRASSINGLY PARALLEL SGLD

We now present how to combine the SGRLD sampler with an embarrassingly parallel method to explore data redundancy and distributed computing for better scalability. For simplicity, we develop our distributed method based upon the Weierstrass sampler [19] (Note that the following discussion is also applicable to other embarrassingly parallel methods, such as [15, 14], since they follow a similar distributed design).

For $\mathbf{x} \in \mathbf{R}^q$, the Weierstrass transform [22] of a function $f(\mathbf{x})$ is

$$f(\mathbf{x}) = \lim_{h \to 0} \int_{-\infty}^{\infty} K_h(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y},$$

where the multivariate kernel function $K_h(\mathbf{x}, \mathbf{y})$ is defined as:

$$K_h(\mathbf{x}, \mathbf{y}) = \frac{1}{\sqrt{(2\pi)^q |H|}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{y})^T H^{-1}(\mathbf{x}-\mathbf{y})}, \tag{5}$$

in which H is a q-by-q diagonal matrix, where the elements on its diagonal are the same and defined as the tuning parameter h. We denote θ_k as the distribution of the topic with integer label k in Θ. Under the common independent prior $P_0(\Theta) = \prod_k P_0(\theta_k)$, we have the factorization form of each sub-posterior $P(\Theta|\mathbf{w}_i) = \prod_k P(\theta_k|\mathbf{w}_i)$ in Eq. (4). Therefore, we can deal with each θ_k separately. Formally, according to Eq. (4) and the factorization form of each sub-posterior as above, we have $P(\theta_k|\mathbf{w}) \propto \prod_i P(\theta_k|\mathbf{w}_i)$. If we substitute $P(\theta_k|\mathbf{w}_i)$ with $f(\mathbf{x})$, and $\mathbf{y}$ with $t_{k,i}$ (t_k in node i), we have

$$\begin{aligned} P(\theta_k|\mathbf{w}) &\propto \prod_i P(\theta_k|\mathbf{w}_i) \\ &\approx \int \prod_i K_h(\theta_k, t_{k,i}) P(t_{k,i}|\mathbf{w}_i) dt_{k,1} \cdots dt_{k,n}. \end{aligned} \tag{6}$$

Note that we use approximation to replace the limit $h \to 0$, as long as we set h to a small enough value (A recommended value of h is around $\sqrt{n}\sigma^2$ [19], where σ is the posterior variance). Note that $P(t_{k,i}|\mathbf{w}_i)$ now represents the sub-posterior in node i. The integration in Eq. (6) is in fact a marginalization performed on joint probability distribution

$$P(\theta_k, t_{k,1} \cdots, t_{k,n}|\mathbf{w}) \propto \prod_i K_h(\theta_k, t_{k,i}) P(t_{k,i}|\mathbf{w}_i). \quad (7)$$

It marginalizes all samples from sub-posteriors and gives the predictive distribution of θ_k. Hence this intergation can be estimated by a standard block-wise Gibbs sampler:

$$\begin{aligned} P(t_{k,i}|\theta_k, \mathbf{w}_i) &\propto K_h(\theta_k, t_{k,i}) P(t_{k,i}|\mathbf{w}_i) \\ P(\theta_k|t_{k,1}, \cdots, t_{k,n}) &\propto \mathcal{N}(\bar{t}_k, H_0), \end{aligned} \quad (8)$$

where H_0 is the diagonal matrix with all its diagonal elements equal $\frac{1}{nh^{-2}}$, and $\bar{t}_k = \frac{1}{n}\sum_i t_{k,i}$. The first part of Eq. (8) indicates how the samples are drawn from each sub-posterior: The sub-posterior is multiplied with a kernel term which penalizes the samples that are far from the true posterior mode; in such way the sub-posteriors are corrected towards the true posterior. The second part of Eq. (8) suggests that the global samples are obtained by sampling from a Gaussian distribution. This process is usually performed by a master node which collects local $t_{k,i}$s from workers asynchronously. The inference of the corrected local sub-posterior $P(t_{k,i}|\theta_k, \mathbf{w}_i)$ is performed in each worker node, and can be implemented by the SGRLD sampler. Formally, we take the derivative of $P(t_{k,i}|\theta_k, \mathbf{w}_i)$, and re-arrange it into the SGRLD update equation, where we obtain the corrected version (one parameter case)

$$\begin{aligned} t_{kw,i}^{(s+1)} \leftarrow |t_{kw,i}^{(s)} + \frac{\epsilon^{(s)}}{2}(-h(t_{kw,i}^{(s)} - \theta_{kw}^{(s)}) + \\ \beta - t_{kw,i}^{(s)} + \mathbb{E}) + (t_{kw,i}^{(s)})^{\frac{1}{2}} \eta_{kw}|. \end{aligned} \quad (9)$$

Note we use $\mathbb{E}$ as the abbreviation of the expectation term in Eq. (2). The kernel term $-h_i(t_{kw,i}^{(s)} - \theta_{kw}^{(s)})$ shifts the gradient towards the true posterior mode by an amount which is governed by h. As the training proceeds, h shall decrease and finally vanish [19].

3.1 The EM process in EPSGLD

Embarrassing parallel methods can produce samples without communication. However, as we discussed above, synchronizations are important for distributed LDA methods to converge to the optimal state. This property also holds in our method. As we inspect Eq. (8), the sampling of $P(t_{k,i}|\theta_k, \mathbf{w}_i)$ depends on the global matrix Θ; and the sampling of $P(\theta_k|t_{k,1}, \cdots, t_{k,n})$ in turn depends on the local T_is. Therefore, Θ has a similar role as the counts matrix $\tilde{n}_{kw,i}$ in Yahoo!LDA, and should be synchronized across all nodes. Formally, EPSGLD can be categorized as a *data-parallelism* method, where the whole dataset is partitioned across machines and computations are performed on each node given a local copy of the globally share model [28]. Other examples of this type include approximate distributed LDA (AD-LDA) [16] and Yahoo!LDA. In [16], Newman et al. did an extensive analysis on the role of synchronization in data-parallelism methods: when sampling independently, the topics (with the same integer label) in each worker can drift apart, so that topic k on one worker may diverge from topic k in another worker. Through synchronization, matrices in each node are updated to be consistent, leading to the equivalent predictive power as their non-distributed counterparts. However, if the synchronization interval increases, it is likely for this type of methods to converge to a suboptimal solution, since topics in each node can drift far apart. They concluded that there are two ways to prevent data-parallelism methods from failing: (1) increase the number of workers; and (2) apply frequent synchronizations.

Though the first method is easy to follow for all distributed algorithms, the second advice is not suitable in our method, since the number of iterations required for the stochastic sampler to converge to a stationary state can be in the order of thousands and an $\mathcal{O}(I)$ communication overhead is unacceptable. In order to avoid frequent synchronizations without imposing too much accuracy loss, we seek solutions by investigating the properties of Eq. (8). We know that the main function of synchronization is to prevent topics in each node to drift apart, so if we are able to constrain local samplers so that they tend to draw samples with similar statistical meaning, we can then safely reduce the frequency of synchronizations. Therefore, we can propose the third way to ensure the validity of the data-parallelism methods, that is to correct the sub-posterior towards the true posterior, so that topics are not likely to drift apart even though they are sampled independently.

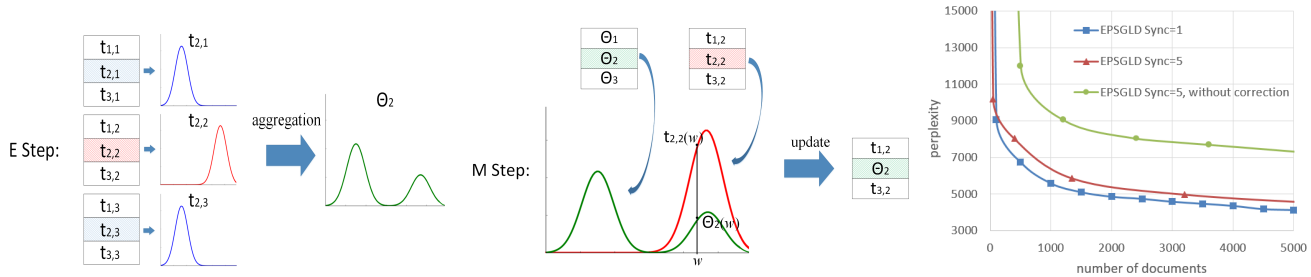
Fortunately, such mechanism indeed exists in Eq. (8). As we inspect Eq. (8), we find that the sub-posterior is multiplied with a kernel term which makes the distribution biased towards the global matrix Θ. And Θ is sampled from $P(\theta_k|t_{k,1}, \cdots, t_{k,n})$ in order to estimate the unknown true posterior. As the training proceeds, this approximation becomes accurate and results in the local T_is being biased towards the true posterior. Hence, we find that the third solution is already contained in our method, and functions as an EM-style process, where in the E-step we aggregate Θ as an approximation of the true posterior; and in the M-step we correct sub-posteriors towards Θ so that samples in T_is tend to have similar statistically meaning. By iteratively applying these two steps, the sub-posteriors are then biased towards the true posterior.

We use a toy example to illustrate this process. Suppose we are to aggregate 3 T_is from workers 1, 2 and 3. As shown in Fig.2, for the second topic, $t_{2,1}$ and $t_{2,3}$ have the same statistical property, while topic $t_{2,2}$ in T_2 diverges form them. In E step, we average over T_is and use Eq.(8) to obtain Θ and send it to workers. In M step, the worker 2 updates T_2. In sampling each topic, Eq. (9) is biased towards Θ so that the SGRLD sampler updates the $t_{2,2}$ towards θ_2. Then for the next EM step, the aggregated second topic θ_2 should be more similar to those in T_1 and T_3, which in turn makes $t_{2,2}$ to be more similar to $t_{2,1}$ and $t_{2,3}$. Given enough number of such steps, the topics will not drift apart even with less frequent synchronizations. We note that this process is essentially different from what is applied in DSGLD, where it also produces global samples by averaging the T_is using chain coupling technique. As we have discussed, the EM process holds only if the sub-posteriors are biased towards Θ. In DSGLD, such mechanism is missing, thus increasing the synchronization interval (or trajectory length in DSGLD) can significantly influence the convergence.

To demonstrate the effectiveness of the EM process, we run a modest scale inference task ($K = 10^2$,$V = 10^5$) with 3 distributed settings: EPSGLD with synchronization after each iteration, EPSGLD with synchronization after 5 iterations, and EPSGLD with synchronization after 5 iterations but the kernel term in Eq. (9) is removed (which is a similar setting to that of the DSGLD). The result is shown in Fig.2(c). The "EPSGLD Sync=1" curve sets up the lower bound for approximation error. The "EPSGLD Sync=5" curve has a similar performance when compared to the lower bound, while the EPSGLD without correction converges to a suboptimal state. This suggests that the EM process can significantly reduce the accuracy loss caused by the decrease of synchronization frequency.

3.2 Communication scheduling

We have shown that the EM process contained in EPSGLD can enable the method to work with less frequent synchronizations. But as we notice, the decrease of synchronizations not only influences the convergence of our method, but also the effectiveness of how the kernel term can correct the sub-posterior in Eq. (8). It suggests that if one updates T_i and Θ rigorously according to Eq. (8), then worker nodes will need to acquire the latest Θ to compute the kernel term after each iteration. We note that this property is resulted from the design of Weierstrass sampler itself, and differs



(a) (b) (c)

Figure 2: The graph uses a toy example to illustrate the EM process: (a) 3 T_is are aggregated into Θ. The second topic $t_{2,2}$ in T_2 diverges from those in T_1 and T_3; (b) worker updates T_2 using Eq. (9). Since the sub-posterior is biased towards Θ, the second topic changes towards θ_2; (c) perplexity of stochastic samplers under 3 settings: EPSGLD with synchronization after each iteration, denoted by "Sync=1", EPSGLD with synchronization after 5 iterations and EPSGLD with synchronization after 5 iterations but the kernel term in Eq. (9) is removed. (Best viewed in color).

from the need of synchronizations for LDA as discussed above. In other words, this property still exists if we apply our method to other models such as logistic regression. In order to eliminate the communications, Wang et al. [19] suggested to generate all Θs using approximate methods such as Laplace approximation and store them for training, so that the sampler can draw new T_i without any communication. However, since our model can contain 10^{10} parameters, storing all the samples is not viable. Given that communications are inevitable, we resort to address this issue through a series of well-scheduled synchronizations, as detailed below.

A useful strategy for this problem is to apply delayed update. For example, Yahoo!LDA allows worker i to save a local stale copy of the global counts matrix $\tilde{n}_{kw,i}$, and updates global n_{kw} using Eq. (3) after a few iterations. Another example is DSGLD — Instead of jumping to another worker at each iteration as shown in Fig.1(a), the *trajectory sampling* is used to delay the jump by a number of steps τ, so the communications are then $\mathcal{O}(\frac{I}{\tau})$; this technique can be used to control the level of approximation by trading off computation time with asymptotic accuracy [2]. The idea underlying this strategy is that LDA is usually very sparse and changes slowly in each iteration of the update, so the stale copy of the model is not likely to be out-of-date in a short period.

In EPSGLD, h is initially set to be relatively large in order to encourage samplers to explore the posterior space, so the kernel term penalizes less for the samples being different from the stale copy of Θ; as the training proceeds, h becomes smaller and matters a lot in penalizing the samples being far from Θ, but at this point since the model is adequately sparse and the step size is small, the new samples usually concentrate on the true posterior mode and thus are similar to Θ. Therefore, in either case, given that the aggregated Θ can correctly approximate the true posterior (as is often the case), we can reuse the stale Θ in local sampling for a certain number of iterations. Additionally, the timing for synchronization in EPSGLD is determined by the master node, so worker nodes will respond to the master node immediately after receiving the notification. This helps to reduce the latency in synchronization.

The only remaining issue is to determine the schedule for communications. It is not a trivial task as we take into account its effect on the performance of the EM process, the effectiveness of the kernel correction and the overall computational efficiency. Here we propose to determine the schedule by putting different choices into test and see how EPSGLD performs under certain communication patterns. To do this, we first present a model in order to describe different schedules formally:

$$\phi(\bar{I}) = \begin{cases} 1 & , \bar{I} \in \{c_1 \times n^{c_2} | n = 0, 1, 2, ..\} \\ 0 & , otherwise, \end{cases} \tag{10}$$

where $\bar{I} = \frac{1}{n}\sum_i I_i$, and I_i is the number of iterations in worker i, c_1 and c_2 are parameters to be determined. During training, the master keeps track of the iteration of each worker and computes $\bar{I}$, once $\phi(\bar{I})$ gives 1, the master will inform workers to synchronize. The model is flexible to describe most of the likely schedules: $c_1 = 1, c_2 = 1$ means communicate-per-iteration, $c_1 = \tau, c_2 = 1$ lets the synchronization delay for τ iterations, $c_1 = 1, c_2 = 2$ concentrates more communications at the beginning and less afterwards, and so on. To determine the parameters. Our approach is to run a grid search on c_1, c_2 and compare the speed and perplexity of each case. By investigating the curves of perplexity against iterations under different settings of c_1, c_2, we gain some insights into the relationship between communication scheduling and the overall performance. The experiment is shown in Section 4.

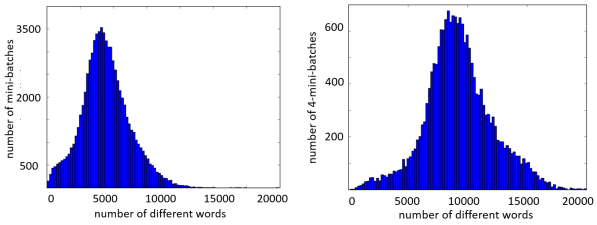
3.3 Memory strategy

In large-scale LDA, manipulating a complete T_i (or Θ) can be difficult as either holding it in memory or sending it to other nodes is a nontrivial task. Hence, an appropriate memory strategy for EPSGLD is needed to improve the scalability. Here we draw inspirations from LightLDA, which partitions the counts matrix n_{kw} into multiple data blocks (along the word-axis) that are small enough to be held in memory, and the worker only holds and updates one block at each time. For our method, we find that the number of unique words contained in one mini-batch is significantly smaller than the vocabulary size, so it is sensible to only load those word-slices that occur in the mini-batch.

To better understand our corpus, Fig.3 illustrates the histogram of the number of unique words in a mini-batch and 4-mini-batches set of the Wiki corpus (See Section 4 for detail on this corpus). It suggests that most of the mini-batches contain only 4,000 different words, and the maximum is approximately bounded by 10,000, while a 4-mini-batches set usually contains 10,000 words and is bounded by 20,000. That means one only needs to load a T_i slice with 10^4 unique words in order to update for several iterations. In practice, we can load as many word-slices as possible at once to avoid frequent disk I/O. Given the model size and hardware environment, we load T_i slice that covers words in 4-mini-batches set. In our implementation, we group the documents into 4-mini-batches datasets, and each set has a data structure

$$[[batch_1, \cdots, batch_4], [\delta, \delta_1, \cdots, \delta_4]],$$

where $batch$ is the actual dataset, $\delta_1, \cdots, \delta_4$ are V-dimensional boolean vectors that indicate which word slices should be loaded for these batches respectively, and $\delta = \delta_1 | \delta_2 | \delta_3 | \delta_4$, where $|$ denotes the "or" operation.



(a) (b)

Figure 3: (a) The histogram of the number of different words in one mini-batch data of the Wiki corpus; (b) The histogram of the number of different words in 4-mini-batches data of the Wiki corpus.

Algorithm 1 Master node

init: $\bar{I}$
while not called to stop **do**
  while $\phi(\bar{I})$ is 0 **do**
    receive: I_i from workers
    $\bar{I} \leftarrow \frac{1}{n}\sum_{i=1}^{n} I_i$
  end while
  synchronize: $\tilde{\delta}_i, \cdots, \tilde{\delta}_n, \tilde{\delta}$
  receive: $\tilde{\delta}_i(T_i)$ from workers
  send: $\tilde{\delta}(\Theta) \leftarrow \frac{1}{n}\sum_{i=1}^{n} \tilde{\delta}_i(T_i)$
end while

This technique is also applicable in synchronization. Worker and master nodes can send the matrix slice together with its mask vector, instead of the complete one. If we denote the mask for worker i at iteration s as $\delta_i^{(s)}$, and the matrix slice made up with these slices as $\delta_i^{(s)}(T_i)$. The mask for all modified slices after last synchronization is obtained by $\tilde{\delta}_i = \delta_i^{(s)}|\delta_i^{(s+1)}|\cdots$. Each time for the synchronization, the worker only sends $\tilde{\delta}_i(T_i)$ and $\tilde{\delta}_i$ to master; the master first computes the global mask $\tilde{\delta} = \tilde{\delta}_1|\tilde{\delta}_2|\cdots$, and prepares buffer to receive the local samples.

3.4 Algorithms

Based on our previous discussions, we present the pseudo code of our EPSGLD in **Algorithm 1** and **Algorithm 2**. In the method, the master node keeps collecting the number of iterations I_i from each worker, and tests if $\phi(\bar{I})$ is 1. If true, all nodes first synchronize their masks and obtain the $\tilde{\delta}$, and then the master receives $\tilde{\delta}_i(T_i)$ from workers and averages them to get $\tilde{\delta}(\Theta)$, finally sends it to workers. For the worker, it continues to update with a SGRLD sampler, until it is informed to synchronize, and the following operations are similar to those in master.

4. EXPERIMENTS

We now present more empirical results on large datasets. We focus on two tasks: (1) We determine the parameters in communication scheduling, and examine the effectiveness of the EM process at a larger scale; and (2) We compare the performance with DSGLD and LightLDA, two representative distributed methods. Though our main focus is to develop a better distributed SGLD sampler, LightLDA is also included as a state-of-the-art distributed batch method to demonstrate the scalability of EPSGLD.

Algorithm 2 Worker node

input: dataset $\mathbf{w}_i$, tuning parameter h, local copy Θ
while not called to stop **do**
  send: I_i
  update: $T_i \leftarrow SGRLD(\mathbf{w}_i, h, \Theta)$ using Eq. (9)
  if master calls for T_i **then**
    synchronize: $\tilde{\delta}_i, \cdots, \tilde{\delta}_n, \tilde{\delta}$
    send: $\tilde{\delta}_i(T_i)$
    receive: $\tilde{\delta}(\Theta)$
  end if
end while

Table 1: Statistics of the two datasets, where L denotes the number of tokens.

Datasets	D	L	V	L/D
Wiki	4.2M	1B	100K	238
Clueweb12	40M	14.7B	100K	360

4.1 Datasets and Setups

Table 1 summaries the datasets in our experiments. The Wiki dataset contains the latest web pages from Wikipedia[1]; and Clueweb12 is a subset of Clueweb12 dataset, a large crawl of web pages[2]. Note that the Clueweb12 data also share the property that we discussed in Section 3.3. For each corpus, we take the top 10^5 most frequent words (with stop words excluded) as our vocabulary and parse the documents into a bags-of-words format. In order to apply large-scale LDA inference, the Gibbs sampling used in SGLD is implemented using alias table. The step size for SGLD is represented using function $\epsilon = (a + b \cdot I)^{-\frac{1}{3}}$ as recommended in [18]. We determine the a and b by comparing the performance on a grid based on 242 runs, and we pick $a = 10^{5.2}, b = 10^{-6}$ as our default values. For each document, 20% words are extracted, and all these words are used as the held-out test set.

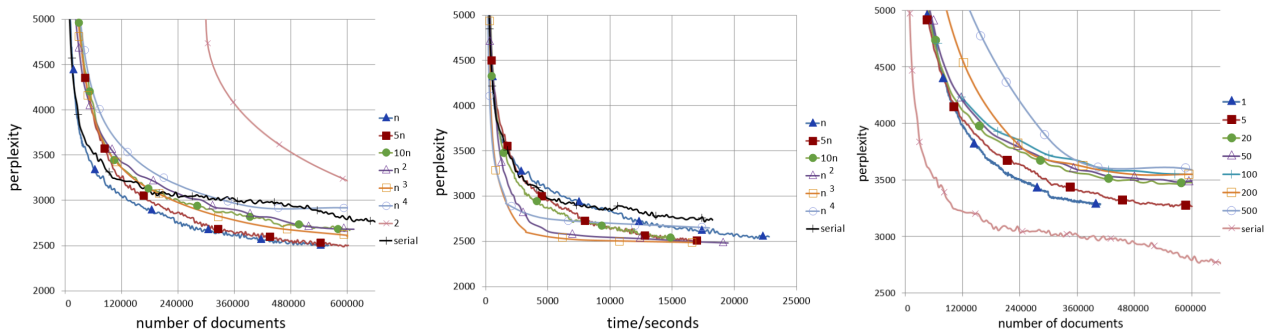
All codes are implemented using Python and its supporting libraries (numpy, h5py, etc.). The code of SGRLD provided in [17] is used as reference. The critical parts of the code are compiled into C using Cython to improve performance. The token throughput of a serial Gibbs sampler (used in both SGLD and LightLDA) with alias table is approximately 300K/sec. It is relatively slower than what is provided in Yuan et al.'s implementation, which is around 1M/sec [25]. This is due to the fact that the code is implemented using Python and Cyhton instead of C.

4.2 Communication scheduling

We first determine the parameters c_1 and c_2 in Eq. (10) that control the frequency of synchronizations, and investigate how this frequency influences the convergence of EPSGLD. The performance measure is the perplexity on the test set with respect to the number of documents visited. This helps us to compare how effective different methods can explore the data information for learning. In this case, we would expect the serial sampler to have the best performance, since the approximation in distributed samplers may lead to a waste of data. Therefore, comparing with the serial sampler can help to investigate the power of distributed methods in approximating the true posterior, which further indicates whether the method design is good or not. We also present the curves of perplexity against time for comparison.

We search for the parameters by running a grid search on c_1 and c_2, where c_1 ranges from 1 to 10 and c_2 ranges from 1 to 4. EPSGLD is distributed to 12 machines with model size $K = 10^3$,

[1]https://dumps.wikimedia.org/
[2]http://www.lemurproject.org/clueweb12.php/



(a) (b) (c)

Figure 4: (a) Perplexity of EPSGLDs on Wiki with different schedules against the number of documents visited, where "serial" denotes the curve of SGRLD. The curve with legend "2" means the method that synchronizes when $\bar{I} = 500, 1000$; (b) perplexity of EPSGLDs on Wiki with different schedules against time, where "serial" denotes the curve of SGRLD; (c) perplexity of DSGLDs on Wiki dump with different trajectory lengths against the number of documents visited, where "serial" denotes the curve of SGRLD. DSGLD is distributed to 12 machines.

$V = 10^5$. We report the results on the Wiki dataset in Fig.4(a) and Fig.4(b), where each legend in the graph represents a certain parameter configuration (e.g., "n^2" means $c_1 = 1, c_2 = 2$). To keep illustration uncluttered, we only present those curves that are informative for determining c_1 and c_2.

Schedule and Convergence: We can see that EPSGLD is generally insensitive to the value of c_1, c_2, except the "2" case, where EPSGLD synchronizes only twice at the 500th iteration and the 1000th iteration. The most frequent case "n" converges just as fast as the least frequent case "n^4", which has 5 synchronizations in 1000 iterations. The only difference comes from the slight increase of ending perplexity, which means as the frequency of synchronization decreases, EPSGLD may converge to slightly bad sub-optimums. Additionally, Fig.4(b) demonstrates that as the communication frequency decreases the convergence speeds up significantly.

Another insight into the role of synchronizations can be gained by inspecting Fig.4(a): Since the model changes drastically at the beginning of training, it is important to cover this stage with a sufficiently large number of synchronizations (e.g., with schedule "n^2" or "n^3"); otherwise, if the initial timing is missed, as it is the case in the curve marked by "2", the EM process may fail to bring it back to the right track. This fact is sensible since the step size decreases at $\mathcal{O}(I^{-\frac{1}{3}})$ speed. Thus, if the Θ fails to approximate the true posterior at the beginning due to the lack of synchronizations, it will be difficult to correct it afterwards since the step size has annealed. And this eventually leads to a sub-optimal solution.

Comparison with serial SGRLD: The curve of the serial SGRLD is also shown in Fig.4(a) in order to set up the upper bound of the performance for distributed samplers. However, Fig.4(a) suggests that the best performance is actually bound by EPSGLD with "n" schedule; EPSGLD with some other schedules also outperforms the serial sampler in convergence. This phenomenon seems counter-intuitive, but is in fact sensible: it can be explained by the effect of *variance reduction* [2]. As the master node averages over T_is and generates Θ, the variance of the gradient estimator reduces by $\frac{1}{n}$. This means that the noise injected by the SGRLD sampler reduces by certain amount, and therefore, the gradient is less stochastic, leading to a faster convergence.

Suggestion on the parameters: The experiment demonstrates that, with the help of the EM process, EPSGLD requires significantly fewer synchronizations to reach satisfactory results. In determining an appropriate schedule for training, one must ensure that the initial stage is covered with a sufficiently large number of synchronizations. As indicated by Fig. 4(a), it is recommended to take the "n^3" schedule (i.e., $c_1 = 1, c_2 = 3$) as the lower bound for the frequency of synchronizations, in which case the communications grow only at $\mathcal{O}(I^{\frac{1}{3}})$ rate, and the loss in accuracy is acceptable.

Comparison with DSGLD: We also test the DSGLD under the same distributed setting. The perplexity-documents curves for DSGLD with different trajectory lengths (iterations delayed before jumping to another node) are shown in Fig.4(c). As the graph suggests, DSGLD converges slower than serial SGRLD, this is due to DSGLD is sensitive to the noise of partitioning of the datasets across nodes; as the trajectory length increases, DSGLD converges to significantly worse sub-optimums due to the lack of correction in sub-posterior.

4.3 Benchmark for EPSGLD

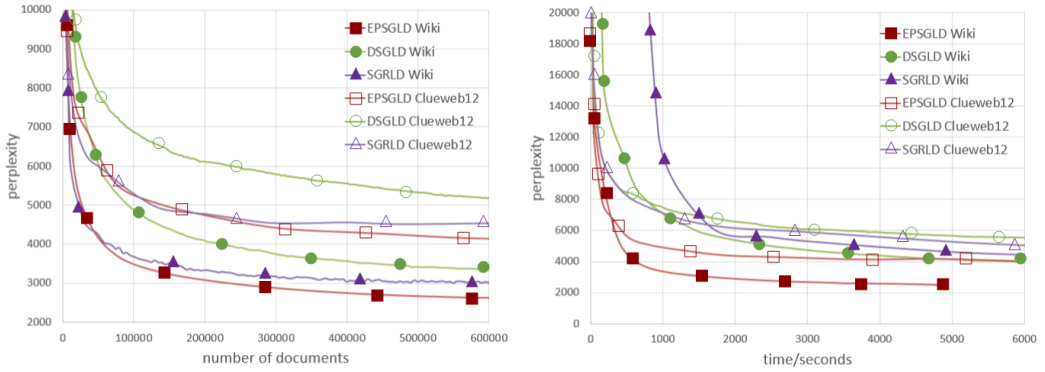
In this section, we test the performance of EPSGLD in handling large-scale models in two settings: (1) we compare the performance of two stochastic distributed methods—EPSGLD and DSGLD, together with the serial SGRLD sampler; and (2) we compare with LightLDA to demonstrate the scalability of EPSGLD.

4.3.1 Comparison with DSGLD and SGRLD

We distribute EPSGLD and DSGLD to 24 machines with the model size $K = 10^5, V = 10^5$. The communication schedule of EPSGLD is "$2n^2$", and the trajectory length of DSGLD is 10. The results are shown in Fig. 5. In Fig. 5(a), with more workers and a larger model size, EPSGLD still maintains an advantage in approximating true posterior samples, leading to a faster convergence than the serial SGRLD. In general, EPSGLD reaches the same perplexity as DSGLD does using only a quarter of the documents. When comparing the performance against time, as shown in Fig. 5(b), we note that the overhead in communication has seriously bounded the performance of DSGLD, while EPSGLD maintains an obvious advantage against the other two stochastic samplers.

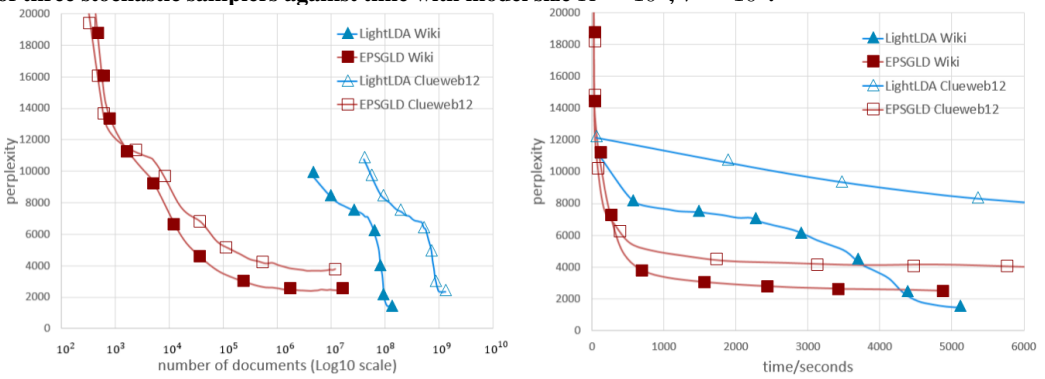
4.3.2 Comparison with LightLDA

Finally, we examine how EPSGLD performs in comparison with LightLDA, a state-of-the-art distributed batch method, on both Wiki and Clueweb12 datasets. The distributed settings are the same as in the previous test. In LightLDA, the n_{kw} is split into 24 slices, and the distributed method is implemented using MPI so that it maintains all the important features. But the parameter server and the hybrid data structure are not implemented, since they are generally applicable to any method. The collapsed Gibbs sampler with



(a) (b)

Figure 5: (a) Perplexity of three stochastic samplers against the number of documents visited with model size $K = 10^5, V = 10^5$; (b) Perplexity of three stochastic samplers against time with model size $K = 10^5, V = 10^5$.



(a) (b)

Figure 6: (a) Perplexity of EPSGLD and LightLDA against the number of documents visited with model size $K = 10^5, V = 10^5$; (b) Perplexity of EPSGLD and LightLDA samplers against time with model size $K = 10^5, V = 10^5$.

$\mathcal{O}(1)$ complexity is shared by both EPSGLD and LightLDA. The results are shown in Fig. 6. Fig. 6(a) suggests that EPSGLD can reach the same perplexity by visiting much fewer documents than LightLDA on both datasets. Fig. 6(b) demonstrates that EPSGLD has a faster speed to reach a good model than LightLDA, especially on the larger Clueweb12 dataset.

However, we should be aware that there are still some potential drawbacks in EPSGLD compared with LightLDA. First, the two figures suggest that LightLDA has a better ending perplexity. This is due to two disadvantages of SGLD, as stated in [23]: (1) SGLD does not require an MH step to correct the samples (since it is performed based on the whole dataset), and this leads to a certain amount of discretization error; and (2) as the step size decreases, the mixing rate slows down, and thus leading to a strong correlation in samples. Such phenomenon is also observed in the experiments of [17], where the Gibbs sampler has a better ending perplexity than SGRLD. But thanks to the variance reduction, this disadvantage can be alleviated, and EPSGLD even reaches a comparable optimum with LightLDA. Another drawback is the difficulty for EPSGLD to scale up to the same magnitude as LightLDA ($V = 10^6$,$K = 10^6$). The main issue is due to the element of T_i is float instead of integer, so it can be hard to convert the T_i into a sparse matrix in order to hold it in memory.

But as we mentioned before, EPSGLD can be seen as an example of data-parallelism, and LightLDA is the one of model-parallelism. These two paradigms are complementary. For example, [20] presented a two layer LDA system, where layer 1 is model-parallelism and layer 2 consists of multiple local model-parallelism clusters performing updates on a globally distributed model [28]. In that case, we can jointly combine the power of two methods. We leave a systematical investigation in future work.

5. CONCLUSIONS

We present EPSGLD, a novel distributed stochastic gradient sampling method for large-scale LDA inference, which scales up the inference by jointly exploring statistical redundancy and distributed computing. Different from previous methods such as DSGLD, EPSGLD is built upon an embarrassingly parallel method, so that it can produce asymptotically exact samples from the true posterior with significantly less communication overhead. We also present communication scheduling and useful memory strategies to reduce the I/O and memory usage. Finally, a set of carefully designed tests are conducted, which demonstrate the effectiveness of our methods. Our results on large-scale LDA models demonstrate that EPSGLD significantly outperforms DSGLD and has a faster speed to reach a good model compared with LightLDA.

Acknowledgments

This work is supported by the National Basic Research Program (973 Program) of China (No. 2013CB329403), National NSF of China (Nos. 61322308, 61332007), the Youngth Top-notch Talent Support Program, and the Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund (the second phase).

6. REFERENCES

[1] 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, 2012.

[2] S. Ahn, B. Shahbaba, and M. Welling. Distributed stochastic gradient mcmc. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1044–1052, 2014.

[3] J. Chang and D. M. Blei. Relational topic models for document networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 81–88, 2009.

[4] J. Chen, K. Li, J. Zhu, and W. Chen. WarpLDA: a Cache Efficient O(1) Algorithm for Latent Dirichlet Allocation. *Proceedings of the 41st International Conference on Very Large Data Bases*, pages 744–755, 2016.

[5] N. Chen, J. Zhu, F. Xia, and B. Zhang. Discriminative relational topic models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 37(5):973–986, 2015.

[6] T. Griffiths. Gibbs sampling in the generative model of latent dirichlet allocation. 2002.

[7] G. Heinrich. Parameter estimation for text analysis. *University of Leipzig, Tech. Rep*, 2008.

[8] M. Hoffman, F. R. Bach, and D. M. Blei. Online learning for latent dirichlet allocation. In *Proceedings of Advances in Neural Information Processing Systems*, pages 856–864, 2010.

[9] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[10] R. Krestel, P. Fankhauser, and W. Nejdl. Latent dirichlet allocation for tag recommendation. In *Proceedings of the third ACM Conference on Recommender Systems*, pages 61–68, 2009.

[11] A. Q. Li, A. Ahmed, S. Ravi, and A. J. Smola. Reducing the sampling complexity of topic models. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 891–900, 2014.

[12] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, pages 583–598, 2014.

[13] 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*, 2(3):26, 2011.

[14] S. Minsker, S. Srivastava, L. Lin, and D. B. Dunson. Robust and scalable bayes via a median of subset posterior measures. *arXiv preprint arXiv:1403.2660*, 2014.

[15] W. Neiswanger, C. Wang, and E. Xing. Asymptotically exact, embarrassingly parallel mcmc. *arXiv preprint arXiv:1311.4780*, 2013.

[16] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed algorithms for topic models. *The Journal of Machine Learning Research*, 10(1):1801–1828, 2009.

[17] S. Patterson and Y. W. Teh. Stochastic gradient riemannian langevin dynamics on the probability simplex. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3102–3110, 2013.

[18] Y. W. Teh, A. Thiéry, and S. Vollmer. Consistency and fluctuations for stochastic gradient langevin dynamics. *arXiv preprint arXiv:1409.0578*, 2014.

[19] X. Wang and D. B. Dunson. Parallel mcmc via weierstrass sampler. *arXiv preprint arXiv:1312.4605*, 2013.

[20] Y. Wang, X. Zhao, Z. Sun, H. Yan, L. Wang, Z. Jin, L. Wang, Y. Gao, J. Zeng, Q. Yang, et al. Towards topic modeling for big data. *ACM Transactions on Intelligent Systems and Technology*, 2014.

[21] X. Wei and W. B. Croft. Lda-based document models for ad-hoc retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 178–185, 2006.

[22] K. Weierstrass. Über die analytische darstellbarkeit sogenannter willkürlicher functionen einer reellen veränderlichen. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, 2:633–639, 1885.

[23] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning*, pages 681–688, 2011.

[24] L. Yao, D. Mimno, and A. McCallum. Efficient methods for topic model inference on streaming document collections. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 937–946, 2009.

[25] J. Yuan, F. Gao, Q. Ho, W. Dai, J. Wei, X. Zheng, E. P. Xing, T.-Y. Liu, and W.-Y. Ma. Lightlda: Big topic models on modest computer clusters. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1351–1361, 2015.

[26] X. Zheng, J. K. Kim, Q. Ho, and E. P. Xing. Model-parallel inference for big topic models. *arXiv preprint arXiv:1411.2305*, 2014.

[27] J. Zhu, A. Ahmed, and E. P. Xing. Medlda: maximum margin supervised topic models. *The Journal of Machine Learning Research*, 13(1):2237–2278, 2012.

[28] J. Zhu, J. Chen, and W. Hu. Big learning with bayesian methods. *arXiv preprint arXiv:1411.6370*, 2014.