Scalable Parallel EM Algorithms for Latent Dirichlet Allocation in Multi-Core Systems

Xiaosheng Liu[1,2], Jia Zeng[1,2,3,*], Xi Yang[1,2], Jianfeng Yan[1,2], Qiang Yang[3,4]
[1]School of Computer Science and Technology, Soochow University, Suzhou 215006, China
[2]Collaborative Innovation Center of Novel Software Technology and Industrialization
[3]Huawei Noah's Ark Lab, Hong Kong
[4]Department of Computer Science and Engineering, Hong Kong University of Science and Technology
[*]Corresponding Author: zeng.jia@acm.org

ABSTRACT

Latent Dirichlet allocation (LDA) is a widely-used probabilistic topic modeling tool for content analysis such as web mining. To handle web-scale content analysis on just a single PC, we propose multi-core parallel expectation-maximization (PEM) algorithms to infer and estimate LDA parameters in shared memory systems. By avoiding memory access conflicts reducing the locking time among multiple threads and residual-based dynamic scheduling, we show that PEM algorithms are more scalable and accurate than the current state-of-the-art parallel LDA algorithms on a commodity PC. This parallel LDA toolbox is made publicly available as open source software at mloss.org.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Latent Dirichlet allocation, parallel EM algorithms, multi-core systems, shared memory systems, scalability

1. INTRODUCTION

Latent Dirichlet allocation (LDA) [4, 3] is a popular probabilistic topic model for content analysis such as web mining [12]. It can automatically infer the hidden thematic groups of observed words called topics from a collection of documents represented as an input sparse document-word matrix. In the big data era, scalable parallel LDA algorithms have attracted intensive research interests because billions of tweets, images and videos on the web become increasingly common. The aim of this paper is to develop efficien parallel LDA algorithms for big data.

There are two widely available parallel architectures: 1) multi-processor [19] and 2) multi-core [29] systems, where the main difference lies in the way to use the memory. In the multi-processor architecture, all processes allocate independent memory space and communicate to synchronize LDA parameters at the end of each learning iteration [21, 27, 19, 15, 25, 1, 38]. The reduction of communication cost still remains an unsolved problem because this cost is often too big to be masked by computation time in web-scale applications. The experimental results confir that the communication cost may exceed the computation cost to become the primitive cost of large-scale topic modeling [27, 15]. In the multi-core architecture, all threads access the shared memory space so that the race condition is serious. A major scalability issue is the locking time among multiple threads. An example of shared memory architecture is GPU-LDA [29], which shares LDA document-topic and topic-word distribution parameters by multiple threads in multi-core GPU. To avoid access conflict in parameter matrices, the input document-word matrix is partitioned into independent data blocks with non-overlapping rows and columns. A preprocessing algorithm is used to balance the number of words in data blocks so that different threads can finis scanning non-conflictin blocks with almost the same time. However, it is difficul for absolute data block balancing and faster threads need to wait for the slowest one causing longer locking time. Yahoo!LDA [25, 1] presents a blackboard architecture that uses the *memcached* technique, a distributed shared cache service, to maintain LDA parameter matrices in the shared memory environment. Parallel processing of two or more corpus shards would lead to serious access conflicts Yahoo!LDA addresses this problem by locking accesses to conflictin shards, but this locking mechanism degenerates its scalability performance when the number of threads increases.

In this paper, we focus on developing more scalable LDA algorithms in shared memory systems, which can handle web-scale content analysis on just a commodity PC having the multi-core architecture. For example, our parallel solution can learn 1000 topics from 8 million documents on just a PC with 12 cores using around 3 hours. In contrast, previous parallel LDA algorithms on 1024 CPUs need around 4.5 hours to do the same task [19]. This result suggests that parallel LDA algorithms in multi-core system is not only competitive even compared to parallel processing over large clusters (multi-processor systems), but it is very affordable also. Generally, there are two main steps to develop scalable parallel LDA algorithms in shared memory systems. The firs step is to choose a batch/online LDA inference algorithm with the fast convergence speed on a single machine. The second step is to parallelize this algorithm in the shared memory system with small locking costs.

As far as the firs step is concerned, batch LDA inference algorithms include expectation maximization (EM) [7], variational Bayes (VB) [4], collapsed Gibbs sampling (GS) [10], collapsed variational Bayes (CVB) [26, 2], and belief propagation (BP) [35]. Most parallel inference solutions of LDA choose GS algorithms

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741106.

because they are more memory-efficien than other algorithms [21, 27, 19, 15, 25, 1]. For example, GS does not need to maintain the large posterior matrix in memory. In addition, GS stores LDA parameters using the integer type by sparse matrices, and often obtains higher topic modeling accuracy than VB [2]. So, GS is generally agreed to be a more scalable choice in many parallel LDA solutions. However, recent research [2] shows that EM [7], CVB0 [2] and BP [35, 34] converge much faster and produce higher topic modeling accuracy than GS. Online LDA inference algorithms [11, 16, 5, 9] become popular with two reasons. First, they combine the stochastic optimization framework [22] with the corresponding batch LDA algorithms, which theoretically can converge to the local optimal point of the LDA's objective function. Second, online algorithms are memory-efficien because they load each mini-batch of data in memory for online processing, and remove the processed mini-batch and local parameters from memory after one look. Although online algorithms converge slower than batch counterparts, they can process big data streams with a high velocity [23].

We choose EM [7] for LDA because of its fast convergence speed as well as high topic modeling accuracy. To justify this choice, we derive the batch EM (BEM), incremental EM (IEM) [18] and online EM (OEM) [6] algorithms for learning LDA, and compare them with other widely used LDA algorithms such as VB, GS, CVB and BP. Then, we parallelize EM algorithms in the multi-core systems called parallel EM (PEM). Inspired by the recent development of locking-free parallel stochastic gradient descent for matrix factorization [20, 39, 33], we further propose a residual-based scheduling method to reduce the locking time among multiple threads. In practice, this scheduling method can significantl speed up the convergence of PEM algorithms. Experiments confir that PEM algorithms converge significantl faster and scale up to more data and topics when compared with the current state-of-the-art parallel LDA algorithms [29, 25, 23] in multi-core systems. Note that the proposed PEM can be also deployed in multi-processor systems, similar to previous multi-core solutions that work in multi-processor systems [25, 33]. To summarize, we have the following contributions in this paper:

- We develop scalable PEM algorithms in both batch and online versions for LDA in shared memory environment. These efficien PEM algorithms can converge to the local maximum of the LDA log-likelihood function.
- We propose a residual-based scheduling method, which can reduce the locking time among multiple threads, and in the meanwhile speeds up the convergence of PEM in the multi-core architecture.
- Experiments on three large-scale data sets confir that the proposed PEM algorithms converge faster and are more scalable than the current state-of-the-art [29, 25, 23].

This paper is organized as follows: Section 2 discusses why we choose EM for LDA (Appendix shows the derivation of BEM, IEM, OEM and their convergence analysis). Section 3 describes scalable PEM algorithms by avoiding memory access conflicts reducing the locking time among multiple threads and residual-based dynamic scheduling. Section 4 shows extensive experiments on three large-scale data sets. Section 5 makes conclusions and envisions further work.

2. WHY EM INFERENCE FOR LDA?

LDA allocates a set of thematic topic labels, $\mathbf{z} = \{z^k_{w,d}\}$, to explain nonzero elements in the document-word co-occurrence matrix $\mathbf{x}_{W\times D} = \{x_{w,d}\}$, where $1 \le w \le W$ denotes the word index in the vocabulary, $1 \le d \le D$ denotes the document index in the corpus, and $1 \le k \le K$ denotes the topic index. Usually, the number of topics K is provided by users. The nonzero element $x_{w,d} \neq 0$ denotes the number of word counts at the index $\{w, d\}$. For each word token $x_{w,d,i} = \{0, 1\}$, $x_{w,d} = \sum_i x_{w,d,i}$, there is a topic label $z^k_{w,d,i} = \{0, 1\}$, $\sum_{k=1}^K z^k_{w,d,i} = 1, 1 \le i \le x_{w,d}$. Each nonzero element $x_{w,d} \neq 0$ is associated with a topic probability vector $\sum_k \mu_{w,d}(k) = 1$, which denotes the posterior probability of a topic label $z^k_{w,d} = 1$ given the observed word $\{w, d\}$. The objective of inference algorithms is to infer posterior probability from the full joint probability $p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta)$, where $\mathbf{z}$ is the topic labeling configuration $\boldsymbol{\theta}_{K\times D}$ and $\boldsymbol{\phi}_{K\times W}$ are two non-negative matrices of multinomial parameters for document-topic and topic-word distributions, satisfying $\sum_k \theta_d(k) = 1$ and $\sum_w \phi_w(k) = 1$. Both multinomial matrices are generated by two Dirichlet distributions with hyperparameters α and β. For simplicity, we consider the smoothed LDA with fi ed symmetric hyperparameters [10]. Table 1 summarizes the important notations in this paper.

Table 1: Definition of Notation.

$1 \le d \le D$	Document index
$1 \le w \le W$	Word index in vocabulary
$1 \le k \le K$	Topic index
$1 \le m \le M \times M$	$M \times M$ data blocks
$1 \le n \le N$	Thread index
NNZ	Number of nonzero elements
$\mathbf{x}_{W\times D} = \{x_{w,d}\}$	Document-word matrix
$\mathbf{z}_{W\times D} = \{z^k_{w,d}\}$	Topic labels for words
$\boldsymbol{\theta}_{K\times D}$	Document-topic distribution
$\boldsymbol{\phi}_{K\times W}$	Topic-word distribution
$\boldsymbol{\mu}_{K\times NNZ}$	Responsibility matrix
α, β	Dirichlet hyperparameters

VB [4] infers the following posterior from the full joint probability,

$$p(\boldsymbol{\theta}, \mathbf{z}|\mathbf{x}, \boldsymbol{\phi}, \alpha, \beta) = \frac{p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta)}{p(\mathbf{x}, \boldsymbol{\phi}|\alpha, \beta)}. \quad (1)$$

This posterior means that if we learn the topic-word distribution $\boldsymbol{\phi}$ from training data, we want to infer the best $\{\boldsymbol{\theta}, \mathbf{z}\}$ from unseen test data given $\boldsymbol{\phi}$, i.e., for the best generalization performance. However, computing this posterior is intractable because the denominator contains intractable integration, $\int_{\boldsymbol{\theta},\mathbf{z}} p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta)$. So, VB infers an approximate variational posterior based on the variational EM algorithm [17]:

- Variational E-step:

$$\mu_{w,d}(k) \propto \frac{\exp[\Psi(\hat{\theta}_d(k) + \alpha)] \exp[\Psi(\hat{\phi}_w(k) + \beta)]}{\exp[\Psi(\sum_w [\hat{\phi}_w(k) + \beta])]}, \quad (2)$$

$$\hat{\theta}_d(k) = \sum_w x_{w,d}\mu_{w,d}(k). \quad (3)$$

- Variational M-step:

$$\hat{\phi}_w(k) = \sum_d x_{w,d}\mu_{w,d}(k). \quad (4)$$

In variational E-step, we update $\mu_{w,d}(k)$ and $\hat{\theta}_d(k)$ until convergence, which makes the variational posterior approximate the true posterior $p(\boldsymbol{\theta}, \mathbf{z}|\mathbf{x}, \boldsymbol{\phi}, \alpha, \beta)$ by minimizing the Kullback-Leibler (KL) divergence between them. In the variational M-step, we update $\hat{\phi}_w(k)$ to maximize the variational posterior. Here, we use

Table 2: Time and Space Complexities of LDA Inference Algorithms.

	Posterior	Time	Space (Memory)
VB [4]	$p(\boldsymbol{\theta}, \mathbf{z}\|\mathbf{x}, \boldsymbol{\phi}, \alpha, \beta)$	$2 \times K \times NNZ \times digamma$	$2 \times K \times (D + W)$
GS [31]	$p(\mathbf{z}\|\mathbf{x}, \alpha, \beta)$	$\delta_1 \times K \times ntokens$	$\delta_2 \times K \times W + ntokens$
CVB [26]	$p(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathbf{z}\|\mathbf{x}, \alpha, \beta)$	$\delta_3 \times 2 \times K \times NNZ$	$K \times (2 \times (W + D) + NNZ)$
BEM (Section 7.1)	$p(\boldsymbol{\theta}, \boldsymbol{\phi}\|\mathbf{x}, \alpha, \beta)$	$2 \times K \times NNZ$	$2 \times K \times (D + W)$
Modifie IEM (Section 7.2)	$p(\boldsymbol{\theta}, \boldsymbol{\phi}\|\mathbf{x}, \alpha, \beta)$	$2 \times K \times NNZ$	$K \times (D + W)$
OEM (Section 7.3)	$p(\boldsymbol{\theta}, \boldsymbol{\phi}\|\mathbf{x}, \alpha, \beta)$	$2 \times K \times NNZ$	$K \times (D_s + W + NNZ_s)$

the notation $\hat{\phi}(k) = \sum_w \hat{\phi}_w(k)$ for the denominator in (2). Normalizing $\{\hat{\theta}, \hat{\phi}\}$ yields the multinomial parameters $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$. However, the variational posterior cannot touch the true posterior for inaccurate solutions [4]. In addition, the calculation of exponential digamma function $\exp[\Psi(\cdot)]$ is computationally complicated. As shown in Table 2, the time complexity of VB for one iteration is $\mathcal{O}(2 \times K \times NNZ \times digamma)$, where $digamma$ is the computing time for exponential digamma function, and NNZ is the number of nonzero elements in document-word sparse matrix. For each nonzero element, we need K iterations for variational E-step and K iterations for normalizing $\mu_{w,d}(k)$. The space complexity is $\mathcal{O}(2 \times K \times (D + W))$ for two multinomial parameters and temporary storage for variational M-step.

In contrast to VB, the collapsed GS [10] algorithm infers the posterior by integrating out $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$,

$$p(\mathbf{z}|\mathbf{x}, \alpha, \beta) = \frac{p(\mathbf{x}, \mathbf{z}|\alpha, \beta)}{p(\mathbf{x}|\alpha, \beta)} \propto p(\mathbf{x}, \mathbf{z}|\alpha, \beta). \tag{5}$$

This posterior means that we want to fin the best topic labeling configuratio $\mathbf{z}$ given the observed words $\mathbf{x}$. Because the multinomial parameters $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ have been integrated out, the best labeling configuratio $\mathbf{z}$ is insensitive to the variation of $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$. Maximizing the joint probability $p(\mathbf{x}, \mathbf{z}|\alpha, \beta)$ is intractable (i.e., there are $K^{ntokens}$ configuration that increase exponentially), an approximate inference called Markov chain Monte Carlo (MCMC) EM [17] is used as follows:

- MCMC E-step:

$$\mu_{w,d,i}(k) \propto \frac{[\hat{\theta}_d^{-z_{w,d,i}^{k,old}}(k) + \alpha][\hat{\phi}_w^{-z_{w,d,i}^{k,old}}(k) + \beta]}{\sum_w [\hat{\phi}_w^{-z_{w,d,i}^{k,old}}(k) + \beta]}, \tag{6}$$

$$\text{Random Sampling } z_{w,d,i}^{k,new} = 1 \text{ from } \mu_{w,d,i}(k). \tag{7}$$

- MCMC M-step:

$$\hat{\theta}_d(k) = \hat{\theta}_d^{-z_{w,d,i}^{k,old}}(k) + z_{w,d,i}^{k,new}, \tag{8}$$

$$\hat{\phi}_w(k) = \hat{\phi}_w^{-z_{w,d,i}^{k,old}}(k) + z_{w,d,i}^{k,new}. \tag{9}$$

In the MCMC E-step, GS infers the topic posterior per word token, $\mu_{w,d,i}(k) = p(z_{w,d,i}^{k,new} = 1|\mathbf{z}_{w,d,-i}^{k,old}, \mathbf{x}, \alpha, \beta)$, and randomly samples a new topic label $z_{w,d,i}^{k,new} = 1$ from this posterior. The notation $-z_{w,d,i}^{k,old}$ means excluding the old topic label from the corresponding matrices $\{\hat{\theta}, \hat{\phi}\}$. In the MCMC M-step, GS updates immediately $\{\hat{\theta}, \hat{\phi}\}$ by the new topic label of each word token. In this sense, GS can be viewed as an incremental algorithm that learns parameters by processing data point sequentially. In Table 2, the time complexity of GS for one iteration is $\mathcal{O}(\delta_1 \times K \times ntokens)$, where $\delta_1 \ll 2$. The reason is that we require K iterations in MCMC E-step and less K iterations for normalizing $\mu_{w,d,i}(k)$. According to sparseness of $\mu_{w,d,i}(k)$, efficien sampling techniques [31, 13, 32] can make δ_1 even smaller. Practically, when K is larger than 1000, $\delta_1 \approx 0.05$. Generally, we do not need to store $\hat{\theta}_{K \times D}$ in memory because $\mathbf{z}$ can recover $\hat{\theta}_{K \times D}$. So, the space complexity is $\mathcal{O}(\delta_2 \times K \times W + ntokens)$ because $\hat{\phi}_{K \times W}$ can be compressed due to sparseness [31, 13]. When K is larger than 1000, $\delta_2 \approx 0.8$. Note that all parameters in GS are stored in integer type, saving half memory space than double type used by other algorithms.

Unlike VB and GS, CVB [26] infers the complete posterior given the observed data $\mathbf{x}$,

$$p(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathbf{z}|\mathbf{x}, \alpha, \beta) \propto p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta). \tag{10}$$

Maximizing this posterior means that we want to obtain the best combination of multinomial parameters $\{\theta, \phi\}$ for the best topic labeling configuratio $\mathbf{z}$. However, inference of this posterior is intractable so that the Gaussian approximation is used [26]. In this sense, CVB optimizes an approximate LDA model, which cannot achieve the best topic modeling accuracy. The variational E-step and M-step in CVB are similar to those in GS. The main difference is that the variational E-step requires multiplying an exponential correction factor containing variance update for each nonzero element rather than word token. In Table 2, the time complexity of CVB is $\mathcal{O}(\delta_3 \times 2 \times K \times NNZ)$, where $\delta_3 > 1$ denotes the additional cost for calculating exponential correction factor. The space complexity is $K \times (2 \times (W + D) + NNZ)$ because CVB needs to store one copy of matrix $\mu_{K \times NNZ}$, and two copies of matrices $\hat{\theta}_{K \times D}$ and $\hat{\phi}_{K \times W}$ in memory (one for the original and the other for the variance). Details can be found in [26, 2].

We advocate the standard EM [7] algorithm that infers the posterior by integrating out the topic labeling configuratio $\mathbf{z}$,

$$p(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}, \alpha, \beta) = \frac{p(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta)}{p(\mathbf{x}|\alpha, \beta)} \propto p(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta). \tag{11}$$

Unlike the posteriors of VB and GS, this posterior means that we want to fin the best parameters $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ given observations $\mathbf{x}$, no matter what topic labeling configuratio $\mathbf{z}$ is. To this end, we integrate out the labeling configuratio $\mathbf{z}$ in full joint probability, and use the standard batch EM algorithm [8] to optimize this objective (11):

- E-step:

$$\mu_{w,d}(k) \propto \frac{[\hat{\theta}_d(k) + \alpha - 1][\hat{\phi}_w(k) + \beta - 1]}{\sum_w [\hat{\phi}(k) + \beta - 1]}, \tag{12}$$

- M-step:

$$\hat{\theta}_d(k) = \sum_w x_{w,d} \mu_{w,d}(k), \tag{13}$$

$$\hat{\phi}_w(k) = \sum_d x_{w,d} \mu_{w,d}(k). \tag{14}$$

In the E-step, EM infers the responsibility $\mu_{w,d}(k)$ conditioned on parameters $\{\hat{\theta}, \hat{\phi}\}$. In the M-step, EM updates parameters $\{\hat{\theta}, \hat{\phi}\}$ based on the inferred responsibility $\mu_{w,d}(k)$. Unlike VB, EM can touch the true posterior distribution $p(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}, \alpha, \beta)$ in the E-step for maximization. When compared with VB, the time complexity of EM for one iteration is $\mathcal{O}(2 \times K \times NNZ)$ without calculating exponential digamma functions. The space complexity of EM is the same as VB with $\mathcal{O}(2 \times K \times (D + W))$ because of storing $\{\hat{\theta}, \hat{\phi}\}$ as well as the temporary variables in the M-step.

In the past decade, VB and GS have been two main inference algorithms in LDA literatures, while EM has been rarely discussed and used in learning LDA. We show two main reasons to use EM:

1. **EM yields a higher topic modeling accuracy measured by predictive perplexity than both VB and GS.** Predictive perplexity is a standard performance measure for different LDA inference algorithms [4, 2, 35], which is calculated as follows: 1) We randomly partition the data set into training and test sets in terms of documents. 2) We estimate $\hat{\phi}$ on the training set by 500 iterations. 3) We randomly partition each document into 80% and 20% subsets on the test set. Fixing $\hat{\phi}$, we estimate $\hat{\theta}$ on the 80% subset by 500 iterations, and then calculate the predictive perplexity on the rest 20% subset,
$$\exp\left\{-\frac{\sum_{w,d} x_{w,d}^{20\%} \log\left[\sum_k \theta_d(k)\phi_w(k)\right]}{\sum_{w,d} x_{w,d}^{20\%}}\right\}, \quad (15)$$
where $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ are multinomial parameters by normalizing $\{\hat{\theta}, \hat{\phi}\}$, and $x_{w,d}^{20\%}$ denotes the word counts in the the 20% subset. The lower predictive perplexity represents a better generalization ability. It is clear that Eq. (15) is a function of multinomial parameters $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$, and EM infers the best multinomial parameters $p(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}, \alpha, \beta)$ for the low predictive perplexity. By contrast, VB and GS produce higher predictive perplexity than EM because they infer different posteriors as discussed before.

2. **EM converges significantl faster than both VB and GS.** In Appendix (Section 7), we derive BEM [7], IEM [18] and OEM [6] algorithms for LDA. Convergence analysis shows that all these EM algorithms can converge to the local maximum of LDA's objective function, because in the E-step the lower-bound can touch the true posterior. The modifie IEM has a low space complexity, and OEM is able to process big data streams. We see that the zero-order approximation of CVB called CVB0 [2] and asynchronous BP [35, 34] are equivalent to IEM, which have been confirme empirically to converge faster than both VB and GS. Also, online BP (OBP) [37] and stochastic CVB (SCVB) [9] are implementations of OEM, which have been also confirme to be faster than some state-of-the-art online LDA algorithms.

3. SCALABLE PEM FOR LDA

Fig. 1 shows PEM for learning LDA with N threads in shared memory systems. First, we random shuffl and partition the input document-word matrix $\mathbf{x}_{W\times D}$ into $M \times M$ data blocks where $M > N$ (line 1). Second, we run N threads in parallel and each thread performs BEM, IEM and OEM in Figs. 9, 10 and 11 (line 5). Third, we update the residual $r^{m,t}$ (16) after sweeping each block and dynamically schedule the free threads to the free data blocks with the largest residual (lines 6 and 7). Finally, we synchronize the global parameter vector $\hat{\phi}(k)$ after some data blocks (e.g., N) are swept (line 8).

```
   input  : x, K, α, β.
   output : φ̂_{W×K}, θ̂_{K×D}.
1  random shuffl and partition x_{W×D} into blocks
   x^m, 1 ≤ m ≤ M × M, M > N;
2  initialize θ̂_d(k), φ̂_w(k), φ̂(k);
3  repeat
4      for n ← 1 to N thread in parallel do
5          free block x^m: do BEM/IEM/OEM ;
6          free block x^m: update residual r^m ;
7          residual-based dynamic scheduling ;
8      synchronize φ̂(k);
9  until converged;
```

Figure 1: Scalable PEM for LDA.

3.1 Residual-based Dynamic Scheduling

Multiple threads in parallel LDA algorithms have long locking time, which is a main factor that we should try to reduce [29]. This motivates us to develop the residual-based dynamic scheduling method. The document-word matrix is partitioned into multiple independent data blocks for parallel computation in different threads. PEM in shared-memory systems use all available N threads to perform E-step and M-step on different data blocks simultaneously. The challenge is that different threads may read and write the same elements in parameter matrices $\hat{\theta}_d(k)$, $\hat{\phi}_w(k)$ and $\hat{\phi}(k)$, which leads to a serious access conflic problem. The K-length parameter vector $\hat{\phi}(k)$ will be visited by all threads at the same time, so that the best method to avoid access conflict is to make N independent copies of vector $\hat{\phi}(k)$ for N threads [19]. After sweeping a certain number of data blocks (e.g., N), we synchronize the parameter vector by summing the updated parameter matrix $\hat{\phi}_w(k)$, i.e., $\hat{\phi}(k) = \sum_w \hat{\phi}_w(k)$. This synchronization cost is not the main bottleneck in PEM because the sum operation on the K-length vectors is very simple.

According to [29], we can avoid access conflict in $\hat{\theta}_d(k)$ and $\hat{\phi}_w(k)$ by partitioning the document-word matrix into $1 \le m \le M \times M$ blocks. In this case, the number of threads $N = M$. Fig. 2 shows an example of data blocks when $M = 4$ in the firs column. The second and third columns show the parameter matrices $\hat{\phi}_{W\times K}$ and $\hat{\theta}_{K\times D}$, respectively. We use four colors (red, yellow, blue and green) to denote four threads. Fig. 2A shows that four threads simultaneously process four data blocks in diagonal. In this way, four threads will visit only the independent rows in $\hat{\phi}_w(k)$ and independent columns in $\hat{\theta}_d(k)$ without conflicting After processing four diagonal data blocks, four threads will simultaneously move to another four data blocks as shown in Fig. 2B, which also use only the independent rows in $\hat{\phi}_w(k)$ and independent columns in $\hat{\theta}_d(k)$ without conflicting This continues in Fig. 2C and Fig. 2D until all 4×4 data blocks have been processed.

To avoid access conflicts all threads in Fig. 2A need to wait for the slowest thread before moving to the next data block in Fig. 2B. Due to data block imbalance (the number of nonzero elements is unequal in different data blocks), the locking time is the main bottleneck in PEM. There are currently two solutions to make data blocks more balanced. First, an approximate integer programming method is used to fin the better data partition efficien y before parallel computation [29]. Second, the random shufflin method works very well empirically [39], which randomly permutes documents (columns) and vocabulary words (rows) of document-word

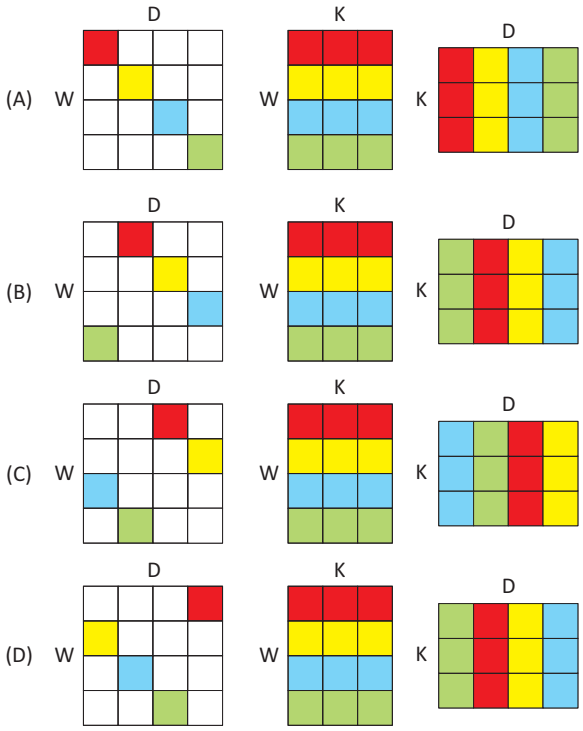

Figure 2: Locking problem occurs when four threads (red, yellow, blue and green) process four independent blocks. The first column denotes the document-word matrix. The second and third columns denote the parameter matrices $\hat{\phi}_w(k)$ and $\hat{\theta}_d(k)$.

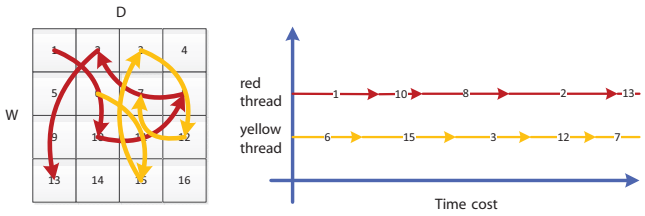

Figure 3: Residual-based dynamic scheduling of two threads (denoted by red and yellow) to process free data blocks without the locking time cost.

sparse matrix before processing. However, the locking problem still exists because NNZ in each data block may not be exactly the same, and there is also slight difference in computing efficien y among different threads.

Our approach is the residual-based scheduling to solve the locking problem in Fig. 3. First, we set $M > N$, i.e., the number of data blocks is more than the number of threads. For example in Fig. 3, we have $N = 2$ threads and partition data matrix into $M \times M = 4 \times 4$ blocks. As far as $N = 2$ threads are concerned, this will create more "free" data blocks without access conflicts On the left figure the red and yellow threads simultaneously process non-conflictin blocks 1 and 6. If the yellow thread finis sweeping block 6 earlier than the red thread, it can directly jump to process a free block 15 without waiting for the red thread. Then, the red thread can jump to process a free block 10 when the yellow thread is processing 15. On the right figure we show the scheduling order of red and yellow threads, where the overlapping data blocks in the time axis have no access conflicts For example, block 1

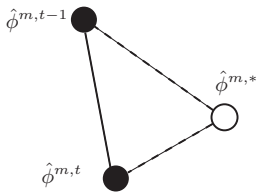

Figure 4: Residual reflect the convergence speed.

(red thread) has no access conflict with blocks 6 and 15 (yellow thread), and block 15 (yellow thread) has no access conflict with blocks 1, 10 and 8 (red thread). In this asynchronous parameter update strategy, there are little locking costs (i.e., few threads need waiting) but scheduling costs.

We propose an efficien residual-based scheduling method that can speed up convergence of PEM. For each data block m, we defin the residual as follows,

$$r^{m,t} = \sum_{w,k} \|\hat{\phi}_w^{m,t}(k) - \hat{\phi}_w^{m,t-1}(k)\|, \tag{16}$$

where $\hat{\phi}_w^{m,t}(k)$ is the updated parameter submatrix at sweep t and $\hat{\phi}_w^{m,t-1}(k)$ is the parameter submatrix before updating at sweep $t-1$. The residual implies the convergence speed when sweeping the current data block m. EM shows that the parameter submatrix $\hat{\phi}^{m,t}$ will converge to a stationary point $\hat{\phi}^{m,*}$ when $t \to \infty$. So, if we minimize the largest distance $\|\hat{\phi}^{m,t} - \hat{\phi}^{m,*}\|$ at higher priority, we will speed up convergence of PEM. However, we do not know this distance because the stationary point $\hat{\phi}^{m,*}$ is unknown. Alternatively, we turn to minimizing the lower bound (16) on this distance that can be calculated easily. Using the triangle inequality, we get the lower bound

$$\begin{aligned} r^{m,t} &= \|\hat{\phi}^{m,t} - \hat{\phi}^{m,t-1}\| \\ &\leq \|\hat{\phi}^{m,t} - \hat{\phi}^{m,*}\| + \|\hat{\phi}^{m,t-1} - \hat{\phi}^{m,*}\|. \end{aligned} \tag{17}$$

Fig. 4 shows the definitio of residual (solid line), which is the lower bound of the distance to be minimized (dashed lines) according to the triangle inequality. In this way, the scheduling order is to sweep the free block with the largest residual first When $t \to \infty$, the residual $r^{m,t} \to 0$ due to convergence. This property ensures that the residual of each free block will become smaller when $t \to \infty$ so that all blocks have the chance to be swept in residual-based dynamic scheduling.

3.2 Implementation Issues

We fin that using single-precision floating-poin computation does not suffer from numerical error accumulation. Empirically, using single precision runs around 10% faster than using double precision by saving around 50% memory. Modern CPU provides Streaming SIMD Extension (SSE) instructions that can concurrently run floating-poin multiplications and additions. To speed up both E-step and M-step, we apply SSE instructions for vector inner products and additions in Fig. 9 (lines 5-6), Fig. 10 (lines 4-6), and Fig. 11 (lines 7-11). Using SSE reduces significantl the time cost of line 4 in Fig. 10.

4. EXPERIMENTS

In our experiments, we call parallel BEM as PBEM, parallel IEM as PIEM, and parallel OEM as POEM. We compare these PEM algorithms with the following state-of-the-art parallel LDA algo-

Table 3: Statistics of data sets.

Data	Pubmed	Wiki	Nytimes
W	6,902	7,871	5,363
D_{train}	8,190,000	4,350,095	290,000
D_{test}	10,000	10,000	10,000
NNZ_{tr}	222,127,506	154,574,168	42,921,633
NNZ_{te}	271,871	360,140	1,457,642

Table 4: Convergence time cost (second) when $K = 100$.

Algorithms	Pubmed	Wiki	Nytimes
PBEM-noScheduling	1162.04	630.75	176.8
PIEM-noScheduling	1280.12	613.69	271.30
PBEM	996.17	597.17	124.21
PIEM	976.96	535.08	192.68
PBGS	4370.29	2648.85	882.48
PBCVB0	2037.28	1317.25	550.48

rithms in shared memory systems: Yahoo!LDA[1] (parallel batch GS, PGS) [25, 1], GPU-LDA (parallel batch CVB0, PCVB0) [29, 2], parallel online VB[2] (POVB) [11], and distributed stochastic MCMC (parallel online GS, POGS) [23]. According to [2], for a fair comparison, we set Dirichlet hyperparameters $\{\alpha = \beta = 0.01\}$ in PGS, POGS and PCVB0, $\alpha - 1 = 0.01, \beta - 1 = 0.01$ in PEM, and $\alpha - 0.5 = 0.01, \beta - 0.5 = 0.01$ in POVB. We carry out experiments on a server with two Intel Xeon X5690 3.47G processors and 140G memory. There are 6 cores in each processor for a total of 12 cores (threads). Likewise to the previous studies [4, 2, 35], we use the predictive perplexity (15) to evaluate the topic modeling accuracy. The lower perplexity the higher topic modeling accuracy. If the difference of predictive perplexity between two consecutive iterations ≤ 5, the algorithm is considered to be converged [35]. In this way, we can compare the convergence time cost among all algorithms.

Table 3 shows publicly available training and held-out test sets in our experiments.[3] The infrequent words are removed from the vocabulary similar to [4] so that the number of nonzero elements in these data sets becomes smaller than that of the original sets. Among these data sets, Pubmed contains $8,200,000$, Wiki contains $4,360,095$, and Nytimes contains $300,000$ documents. These data sets are big enough for evaluating parallel LDA algorithms. For residual-based dynamic scheduling in PEM, we set $M = 2 \times N = 24$ for free data blocks similar to [39]. In benchmark algorithms without residual-based dynamic scheduling, we set $M = N = 12$ similar to [29].

4.1 PBEM and PIEM

In this subsection, we compare parallel batch LDA algorithms in multi-core systems (PBEM/PIEM v.s. PGS/PCVB0). Table 4 shows the convergence time cost of all algorithms when $K = 100$. We see that PBEM or PIEM converges around 1.2 or 1.3, 1.1 or 1.2, 1.4 or 1.4 times faster than PBEM-noScheduling or PIEM-noScheduling on Pubmed, Wiki and Nytimes, respectively. On average, residual-based dynamic scheduling can reduce around $10\% \sim 20\%$ running time for convergence excluding the speedup

[1] https://github.com/shravanmn/Yahoo_LDA
[2] http://radimrehurek.com/2014/09/multicore-lda-in-\python-from-over-night-to-over-lunch/
[3] http://archive.ics.uci.edu/ml/datasets/Bag\%20of\%20Words

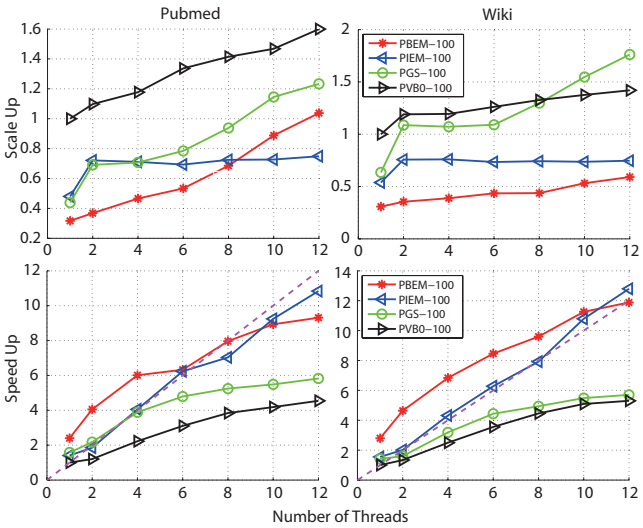

Figure 6: Scalability of PBEM and PIEM ($K = 100$).

effects brought by implementations in Subsection 3.2. This confirm the effectiveness of the residual-based dynamic scheduling to reduce the overall locking time. In addition, we fin that PIEM benefi more from dynamic scheduling than PBEM. The major reason is that IEM often passes the influenc of data blocks with largest residuals more efficientl than BEM. Both PBEM and PIEM converge to almost the same perplexity level, which indicates almost the same topic modeling accuracy. On Nytimes data set, PIEM converges significantl slower than PBEM even after adding residual-based dynamic scheduling. Indeed, there is currently no theory that IEM always converges faster than BEM [18, 14], though some limited experiments [14] indicate that IEM converges faster than BEM. In our experiments, three data sets have quite different word distributions. We see that PIEM converges slightly faster than PBEM on Pubmed and Wiki, but it converges slower on Nytimes. This gives an example that BEM sometimes converges faster than IEM.

In Fig. 5, we compare the predictive perplexity of PBEM/PIEM and PGS/PCVB0 when $K \in \{50, 100, 150, 200, 250\}$ on three data sets. PBEM/PIEM always converges to a much lower predictive perplexity than PGS. On average, there is around $30\% \sim 50\%$ predictive perplexity improvement. Because PCVB0 is similar to PIEM, it converges to almost the same predictive perplexity as PIEM. Clearly, both PBEM and PIEM converge significantl faster than PGS and PCVB0. Their perplexity curves always locate on those of PGS/PCVB0's left. The speedup has been largely attributed to the residual-based dynamic scheduling methods as well as fast convergence speed of EM. In practice, PBEM and PIEM can process $820,000,000$ documents in Pubmed using no more than 16 minutes on a single PC, which is comparable with the previous multi-processor solution on 1024 CPUs (23 minutes) [19]. Therefore, parallel LDA algorithms in multi-core systems are not only competitive but also affordable in big data era.

To test scalability, we perform two types of experiments on both Pubmed and Wiki ($K = 100$): *Scale Up* and *Speed Up*. In *Scale Up*, we establish scalability in terms of the number of documents. We fi each thread to process 10M data and increase the number of threads from 1 to 12 (x-axis). Thus, the scale of processed data increases from 10M to 120M. The *Scale Up* (y-axis) is the division between the convergence time of all other algorithms and that of PCVB0 using 1 thread for 10M data. In *Speed Up*, we establish

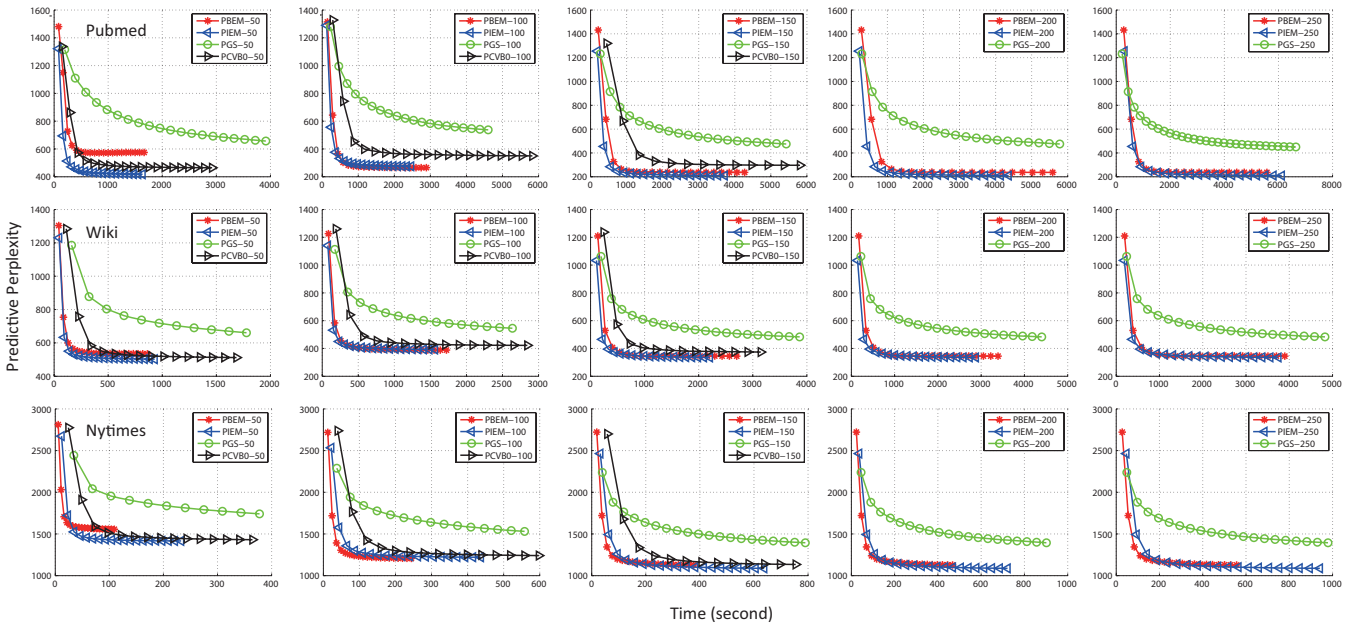

Figure 5: Comparisons of predictive perplexity when $K \in \{50, 100, 150, 200, 250\}$ on three data sets.

scalability in terms of a speedup in convergence time as we increase the number of threads available. We use the entire Pubmed and Wiki data sets, and increase the number of threads from 1 to 12 (x-axis). The *Speed Up* (y-axis) is the division between the convergence time of all other algorithms and that of PCVB0 using 1 thread for the entire data set.

Fig. 6 shows that PIEM performs the best in terms of *Scale Up* and *Speed Up*. The top row shows that the *Scale Up* curve of PIEM remains almost a horizontal line when the processed data increase. This means that PIEM uses a large fraction of runtime to do topic modeling when the volume of data increases. As a comparison, the *Scale Up* curves of PGS, PCVB0 and PBEM increase linearly with respect to the volume of processed data. The bottom row shows that the *Speed Up* curve of PIEM is almost linear with respect to the number of threads. This means that more threads will lead to faster speed. As a comparison, the *Speed Up* curves of PGS, PCVB0 and PBEM bend obviously when the number of threads increases. But PBEM's scalability is still much better than both PGS and PCVB0. The major reason why PIEM has the best scalability is that the residual-based dynamic scheduling performs very well in PIEM so that locking time has been significantl reduced. In this paper, we advocate PIEM in shared memory systems for large-scale data sets due to good scalability performance.

4.2 POEM

In this subsection, we compare POEM with two parallel online LDA algorithms: POVB (multi-core OVB with open source codes)[4] and POGS. Similar to previous work [11], we set the learning parameters as $\{\tau_0 = 64, \kappa = 0.5, D_s = 4096\}$. First, we compare parallel online LDA algorithms with batch algorithms: PBEM and PIEM. Fig. 7 (left panel) shows the convergence time costs (x-axis) and predictive perplexity (y-axis) achieved on three data sets Wiki (red color), Nytimes (blue color) and Pubmed (green color) when $K = 100$. If the algorithm locates in the left-bottom area, it

[4] `http://radimrehurek.com/2014/09/multicore-lda-in-python\-from-over-night-to-over-lunch/`

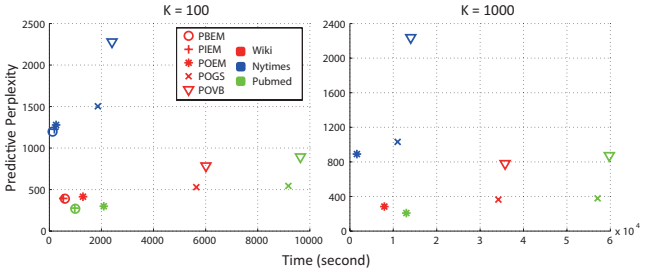

Figure 7: POEM convergence speed and predictive perplexity.

indicates the desirable topic modeling result (i.e., fast convergence speed as well as high topic modeling accuracy). We see that the batch algorithms converge faster than the online ones since they use the global gradient ascent of all data points while the online algorithms use only the local gradient ascent of each mini-batch to update parameters. This is consistent with the previous finding that the convergence rate of stochastic algorithms is often slower than that of batch algorithms [24]. POEM (star sign) can converge at almost the same predictive perplexity of PBEM (circle sign) and PIEM (plus sign), which confirm that POEM can converge to the local maximum of the LDA's log-likelihood function in Section 7.3. As a comparison, POGS (cross sign) and POVB (triangle sign) converge around $3 \sim 4$ times slower than POEM. The main reason is that POVB uses computationally complicated digamma functions in Table 2, while POGS uses much more iterations for learning each mini-batch due to its Markov Chain Monte Carlo (MCMC) nature. Also, we see that POVB and POGS converge at the higher level of predictive perplexity than POEM, supporting our analysis in Section 2 that EM yields a higher topic modeling accuracy because of its inferred posterior $p(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}, \alpha, \beta)$. Although [2] states that the topic modeling accuracy of different inference algorithms can be almost the same by tuning the hyperparameters $\{\alpha, \beta\}$, we still

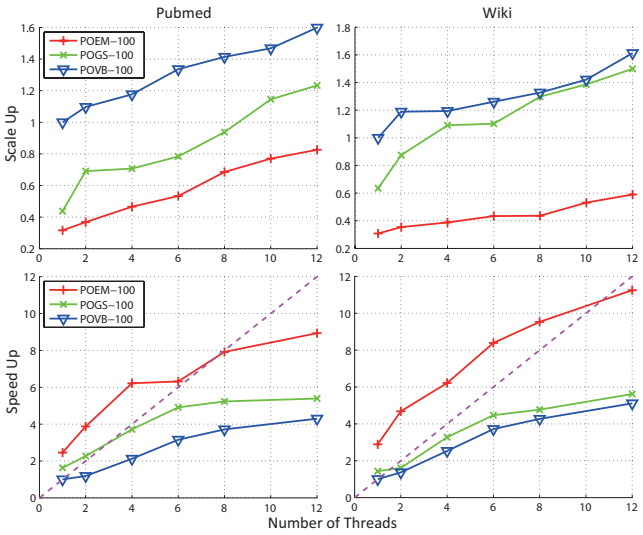

Figure 8: Scalability of POEM ($K = 100$).

advocate the standard EM framework because it converges much faster than both VB and GS.

Although POEM, POVB and POGS converge slower than PBEM and PIEM, they are more memory-efficien to handle larger scale topic modeling tasks on a single PC because they do not need to store the large matrix $\hat{\theta}_{K\times D}$. For example, PBEM and PIEM cannot efficientl process Pubmed data set when $K = 1000$, while POEM, POVB and POGS can do it. Fig. 7 (right panel) compares the convergence time costs and predictive perplexity of POEM, POVB and POGS when $K = 1000$. Similar to the results when $K = 100$, POEM performs significantl faster and achieves a lower perplexity than POVB and POGS. In practice, POEM processes Pubmed data set ($K = 1000$) using less than 3 hours, while PGS using 1024 CPUs requires around 4.5 hours on the same scale of data set [19].

Fig. 8 compares the *Scale Up* and *Speed Up* curves of POEM and POVB/POGS. In *Scale Up*, we f x each thread to process 30M data and increase the number of threads from 1 to 12 (x-axis). The *Scale Up* (y-axis) is the division between the convergence time of all other algorithms and that of POVB using 1 thread for 30M data. The perfect *Scale Up* curve is a horizontal line in the bottom area. Clearly, the *Scale Up* curve of POEM locates significantl lower than those of POGS and POVB, indicating a much better *Scale Up* performance. In *Speed Up*, we divide between the convergence time of all other parallel online LDA algorithms and that of POVB using 1 thread for the entire data set (y-axis). We increase the number of threads from 1 to 12 (x-axis) and see if the convergence speed increases. The perfect *Speed Up* curve is a linear line with a high slope without bending. We see that the *Speed Up* curve of POEM is significantl higher than those of POVB and POGS. This shows that POEM can reach a higher speedup when more threads are used. Both Fig. 6 and Fig. 7 confir that the proposed PEM algorithms are more scalable than the current state-of-the-art solutions.

5. CONCLUSIONS

Scalable LDA algorithms in shared memory systems are needed for big data on widely used multi-core systems. Unlike previous parallel solutions using batch/online VB and GS inference, we advocate the EM framework to build more scalable parallel LDA algorithms. Using the efficien residual-based dynamic scheduling,

input : $\mathbf{x}, K, T, \alpha, \beta$.
output : $\hat{\boldsymbol{\phi}}_{W\times K}, \hat{\boldsymbol{\theta}}_{K\times D}$.
initialize $\hat{\theta}_d(k)$; $\hat{\phi}_w(k)$; $\hat{\phi}(k)$;
for $t \leftarrow 1$ **to** T **do**
  $\hat{\theta}^{\text{new}}_d(k) \leftarrow 0$; $\hat{\phi}^{\text{new}}_w(k) \leftarrow 0$; $\hat{\phi}^{\text{new}}(k) \leftarrow 0$;
  for $x_{w,d} \neq 0$ **do**
    $\mu_{w,d}(k) \leftarrow \text{normalize}([\hat{\theta}_d(k) + \alpha - 1][\hat{\phi}_w(k) + \beta - 1]/[\hat{\phi}(k) + W(\beta - 1)])$;
    $\hat{\theta}^{\text{new}}_d(k) \leftarrow \hat{\theta}^{\text{new}}_d(k) + x_{w,d}\mu_{w,d}(k)$;
    $\hat{\phi}^{\text{new}}_w(k) \leftarrow \hat{\phi}^{\text{new}}_w(k) + x_{w,d}\mu_{w,d}(k)$;
    $\hat{\phi}^{\text{new}}(k) \leftarrow \hat{\phi}^{\text{new}}(k) + x_{w,d}\mu_{w,d}(k)$;
  $\hat{\theta}_d(k) \leftarrow \hat{\theta}^{\text{new}}_d(k)$; $\hat{\phi}_w(k) \leftarrow \hat{\phi}^{\text{new}}_w(k)$;
  $\hat{\phi}(k) \leftarrow \hat{\phi}^{\text{new}}(k)$;

Figure 9: BEM for LDA.

we propose scalable PEM algorithms for LDA with faster convergence speed and shorter locking time than the current state-of-the-art. Experiments show that the residual-based dynamic scheduling can effectively reduce the locking time and speed up convergence of PEM, which can be used in other latent variable models where EM inference works. In our future work, we shall study how to extend PEM from the multi-core systems to multi-processor systems [30, 28].

6. ACKNOWLEDGEMENTS

This work was supported by National Grant Fundamental Research (973 Program) of China under Grant 2014CB340304, NSFC (Grant No. 61373092 and 61033013), Hong Kong RGC project 620812, and Natural Science Foundation of the Jiangsu Higher Education Institutions of China (Grant No. 12KJA520004). This work was partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization.

7. APPENDIX

In this appendix, we derive BEM, IEM and OEM algorithms for LDA, which infer the posterior $p(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}, \alpha, \beta) \propto p(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta)$ from the full joint probability of LDA. This objective is quite different from VB [4], GS [10] and CVB algorithms, which infer the posterior $p(\boldsymbol{\theta}, \mathbf{z}|\mathbf{x}, \boldsymbol{\phi}, \alpha, \beta)$, $p(\mathbf{z}|\mathbf{x}, \alpha, \beta)$, and $p(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathbf{z}|\mathbf{x}, \alpha, \beta)$, respectively. The time and space complexity comparison of these algorithms has been shown in Table 2.

7.1 Batch EM (BEM)

We maximize the likelihood function of LDA in terms of multinomial parameter set $\lambda = \{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ as follows,

$$p(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta) = \prod_{w,d,i} \Big[\sum_k p(x_{w,d,i} = 1, z^k_{w,d,i} = 1 \,|\theta_d(k), \phi_w(k))\Big] \prod_d p(\theta_d(k)|\alpha) \prod_k p(\phi_w(k)|\beta). \quad (18)$$

Employing the Bayes' rule and the definitio of multinomial distributions, we get the word likelihood,

$$p(x_{w,d,i} = 1, z^k_{w,d,i} = 1|\theta_d(k), \phi_w(k)) = p(x_{w,d,i} = 1|z^k_{w,d,i} = 1, \phi_w(k)) \times p(z^k_{w,d,i} = 1|\theta_d(k)), = x_{w,d,i}\phi_w(k)\theta_d(k), \quad (19)$$

which depends only on the word index $\{w, d\}$ instead of the word token index i. Then, according to the definitio of Dirichlet distributions, the log-likelihood of (18) is

$$\ell(\lambda) \propto \sum_{w,d,i} x_{w,d,i} \Big[\log \sum_k \mu_{w,d}(k) \frac{\theta_d(k)\phi_w(k)}{\mu_{w,d}(k)} \Big] + \sum_d \sum_k \log[\theta_d(k)]^{\alpha-1} + \sum_k \sum_w \log[\phi_w(k)]^{\beta-1}, \quad (20)$$

where $\mu_{w,d}(k)$ is some topic distribution over the word index $\{w, d\}$ satisfying $\sum_k \mu_{w,d}(k) = 1, \mu_{w,d}(k) \geq 0$. In (19), we observe that $\sum_{w,d,i}[x_{w,d,i} = 1] = \sum_{w,d} x_{w,d}$, so that we can cancel the word token index i in (20). Because the logarithm is concave, by Jensen's inequality, we have

$$\ell(\lambda) \geq \ell(\boldsymbol{\mu}, \lambda) = \sum_{w,d} \sum_k x_{w,d}\mu_{w,d}(k) \Big[\log \frac{\theta_d(k)\phi_w(k)}{\mu_{w,d}(k)} \Big] + \sum_d \sum_k \log[\theta_d(k)]^{\alpha-1} + \sum_k \sum_w \log[\phi_w(k)]^{\beta-1}, \quad (21)$$

which gives the lower bound of log-likelihood (20). The equality holds true if and only if

$$\mu_{w,d}(k) \propto \theta_d(k)\phi_w(k). \quad (22)$$

In EM, the K-length posterior probability vector $\mu_{w,d}(k)$ is the **responsibility** that the topic k takes for word index $\{w, d\}$ [17]. For this choice of $\mu_{w,d}(k)$, Eq. (21) gives a **tight lower bound** on the log-likelihood (20) we are trying to maximize. This is called the E-step in EM [8].

In the successive M-step, we then maximize (21) with respect to parameters to obtain a new setting of λ. Since the hyperparameters $\{\alpha, \beta\}$ are fi ed, without loss of generality, we derive the M-step update for the parameter $\theta_d(k)$. There is an additional constraint that $\sum_k \theta_d(k) = 1$ because $\theta_d(k)$ is parameter of a multinomial distribution. To deal with this constraint, we construct the Lagrangian from (21) by grouping together only the terms that depend on $\theta_d(k)$,

$$\ell(\theta) = \sum_d \sum_k \Big[\sum_w x_{w,d}\mu_{w,d}(k) + \alpha - 1 \Big] \log \theta_d(k) + \delta(\sum_k \theta_d(k) - 1), \quad (23)$$

where δ is the Lagrange multiplier. Taking derivatives, we fin

$$\frac{\partial}{\partial_{\theta_d(k)}} \ell(\theta) = \frac{\sum_w x_{w,d}\mu_{w,d}(k) + \alpha - 1}{\theta_d(k)} + \delta. \quad (24)$$

Setting this to zero, we get

$$\theta_d(k) = \frac{\sum_w x_{w,d}\mu_{w,d}(k) + \alpha - 1}{-\delta}. \quad (25)$$

Using the constraint that $\sum_k \theta_d(k) = 1$, we easily fin that $-\delta = \sum_k [\sum_w x_{w,d}\mu_{w,d}(k) + \alpha - 1]$. We therefore have our M-step update for the parameter $\theta_d(k)$ as

$$\theta_d(k) = \frac{\hat{\theta}_d(k) + \alpha - 1}{\sum_k \hat{\theta}_d(k) + K(\alpha - 1)}. \quad (26)$$

where $\hat{\theta}_d(k) = \sum_w x_{w,d}\mu_{w,d}(k)$ is the expected sufficien statistics. Similarly, another multinomial parameter can be estimated by

$$\phi_w(k) = \frac{\hat{\phi}_w(k) + \beta - 1}{\hat{\phi}(k) + W(\beta - 1)}, \quad (27)$$

input : $\mathbf{x}, K, T, \alpha, \beta$.
output : $\hat{\boldsymbol{\phi}}_{W\times K}, \hat{\boldsymbol{\theta}}_{K\times D}$.
1 *initialize* $\hat{\theta}_d(k)$, $\hat{\phi}_w(k)$, $\hat{\phi}(k)$;
2 **for** $t \leftarrow 1$ **to** T **do**
3 **for** $x_{w,d} \neq 0$ *in random order* **do**
4 $\hat{\theta}_d(k) \leftarrow (1 - x_{w,d}/\sum_w x_{w,d})\hat{\theta}_d(k)$;
 $\hat{\phi}_w(k) \leftarrow (1 - x_{w,d}/\sum_d x_{w,d})\hat{\phi}_w(k)$;
 $\hat{\phi}(k) \leftarrow (1 - x_{w,d}/\sum_{w,d} x_{w,d})\hat{\phi}(k)$;
5 $\mu_{w,d}(k) \leftarrow$ normalize$([\hat{\theta}_d(k) + \alpha - 1][\hat{\phi}_w(k) + \beta - 1]/[\hat{\phi}(k) + W(\beta - 1)])$;
6 $\hat{\theta}_d(k) \leftarrow \hat{\theta}_d(k) + x_{w,d}\mu_{w,d}(k)$;
 $\hat{\phi}_w(k) \leftarrow \hat{\phi}_w(k) + x_{w,d}\mu_{w,d}(k)$;
 $\hat{\phi}(k) \leftarrow \hat{\phi}(k) + x_{w,d}\mu_{w,d}(k)$;

Figure 10: Modifie IEM for LDA.

where $\hat{\phi}_w(k) = \sum_d x_{w,d}\mu_{w,d}(k)$ is the expected sufficien statistics and $\hat{\phi}(k) = \sum_w \hat{\phi}_w(k)$. Note that the denominator of (26) is a constant. Replacing (26) and (27) into (22), we obtain the E-step in terms of sufficien statistics,

$$\mu_{w,d}(k) \propto \frac{[\hat{\theta}_d(k) + \alpha - 1] \times [\hat{\phi}_w(k) + \beta - 1]}{\hat{\phi}(k) + W(\beta - 1)}, \quad (28)$$

where the EM iterates the E-step and M-step to refin sufficien statistics $\hat{\theta}_d(k)$ and $\hat{\phi}_w(k)$, which can be normalized to be the multinomial parameters according to (26) and (27).

Fig. 9 shows BEM for LDA. We initialize three temporary matrices $\hat{\phi}_w^{new}(k), \hat{\theta}_d^{new}(k), \hat{\phi}^{new}(k)$ (line 3) to accumulate responsibilities in E-step for all words (line 6) without storing the large responsibility matrix $\boldsymbol{\mu}_{K\times NNZ}$ in memory. At the end of each iteration $t, 1 \leq t \leq T$, we copy the three temporary matrices back to $\hat{\phi}_w(k), \hat{\theta}_d(k), \hat{\phi}(k)$ in M-step (line 7). BEM iterates E-step and M-step repeatedly. Suppose λ^{t-1} and λ^t are the parameters from two successive iterations of EM. It is easy to prove that

$$\ell(\lambda^t) \geq \ell(\boldsymbol{\mu}^{t-1}, \lambda^t) \geq \ell(\boldsymbol{\mu}^{t-1}, \lambda^{t-1}) = \ell(\lambda^{t-1}), \quad (29)$$

which shows that EM always monotonically improves the LDA's log-likelihood (20) for convergence. The EM can be also viewed as a coordinate ascent on the lower bound $\ell(\boldsymbol{\mu}, \lambda)$ (21), in which the E-step maximizes it with respect to $\boldsymbol{\mu}$, and the M-step maximizes it with respect to λ.

7.2 Incremental EM (IEM)

In batch EM (BEM), the M-step is performed until the E-step updates all responsibilities $\mu_{w,d}(k)$, which slows down the convergence since the updated responsibility of each word in the E-step does not immediately influenc the parameter estimation in the M-step. This problem motivates incremental EM (IEM) [18]. When compared with BEM (28), IEM alternates a single E-step and M-step for each nonzero element $x_{w,d}$ sequentially. Thus, the E-step of IEM becomes

$$\mu_{w,d}(k) \propto \frac{[\hat{\theta}_{-w,d}(k) + \alpha - 1] \times [\hat{\phi}_{w,-d}(k) + \beta - 1]}{\hat{\phi}_{-(w,d)}(k) + W(\beta - 1)}. \quad (30)$$

The *expected sufficien statistics* are

$$\hat{\theta}_{-w,d}(k) = \sum_{-w} x_{w,d}\mu_{w,d}(k), \quad (31)$$

$$\hat{\phi}_{w,-d}(k) = \sum_{-d} x_{w,d}\mu_{w,d}(k), \quad (32)$$

$$\hat{\phi}_{-(w,d)}(k) = \sum_{-(w,d)} x_{w,d}\mu_{w,d}(k), \quad (33)$$

where $-w$, $-d$ and $-(w,d)$ denote all word indices except w, all document indices except d, and all word indices except $\{w,d\}$. After the E-step for each word, the M-step will update the sufficien statistics immediately by adding the updated posterior $\mu_{w,d}(k)$ (30) into (31), (32) and (33).

Comparing the E-step between BEM and IEM, we fin that the major difference between (28) and (30) is that IEM excludes the current posterior $x_{w,d}\mu_{w,d}(k)$ from sufficien statistics in (31), (32) and (33). As a result, IEM's space complexity is $\mathcal{O}(K\times(D+W+NNZ))$ by storing the large responsibility matrix $\boldsymbol{\mu}_{K\times NNZ}$. For example, if $K=100$, the responsibility matrix will occupy around 360GB (using double-precision floating-poin format) memory on the Pubmed data set [21] having $483,450,157$ nonzero elements. This space is currently too large to be afforded by a single commodity PC. Note that CVB0 [2] and asynchronous BP [35, 36] are equivalent to IEM, which are also memory-consuming for big data on a single PC. So, we propose a modifie IEM in Fig. 10 that do not need to store the large responsibility matrix. After random initialization, we reduce the parameter matrices $\hat{\theta}_d(k), \hat{\phi}_w(k), \hat{\phi}(k)$ in a certain proportion (line 4). This avoids to subtract the current responsibility from parameter matrices in (31), (32) and (33). Then, the E-step of incremental EM becomes (28) rather than (30). In this way, we do not need to store the large responsibility matrix $\boldsymbol{\mu}_{K\times NNZ}$ in memory. After E-step (line 5) for each nonzero element, the parameter matrices can be compensated by the updated K-tuple responsibility $\mu_{w,d}(k)$ in M-step (line 6). In this way, the change of parameter matrix will immediately influenc the update of the responsibility for the next nonzero element (line 5). In anticipation, this incremental update method is more efficien to pass the influenc of the updated responsibility than batch EM in Fig. 9. Likewise, it is easy to see that IEM can also converge to the local stationary point of LDA's log-likelihood because

$$\ell(\lambda^t) = \ell(\boldsymbol{\mu}^t, \lambda^t) \geq \ell(\mu^t_{w,d}, \boldsymbol{\mu}^{t-1}_{-(w,d)}, \lambda^t)$$
$$\geq \ell(\mu^t_{w,d}, \boldsymbol{\mu}^{t-1}_{-(w,d)}, \lambda^{t-1}) \geq \ell(\boldsymbol{\mu}^{t-1}, \lambda^{t-1}) = \ell(\lambda^{t-1}). \quad (34)$$

7.3 Online EM (OEM)

The basic idea of online algorithms is to partition a stream of D documents into small mini-batches with size D_s, and use the online gradient produced by each mini-batch to estimate topic distributions incrementally. OEM [6] combines IEM with the stochastic approximation, which achieves convergence to the stationary points of the likelihood function by interpolating between sufficien statistics based on a learning rate ρ_s satisfying Robbins-Monro conditions [22],

$$\rho_s = (\tau_0 + s)^{-\kappa}, \quad (35)$$

where τ_0 is a pre-define number of mini-batches, s is the mini-batch index and $\kappa \in (0.5, 1]$ is provided by users. Similar to (34), it is easy to observe that

$$\ell(\hat{\phi}^s) = \ell(\boldsymbol{\mu}^{s+1:\infty}, \mu^s, \hat{\phi}^s) \geq \ell(\boldsymbol{\mu}^{s+1:\infty}, \mu^s, \hat{\phi}^{s-1})$$
$$\geq \ell(\boldsymbol{\mu}^{s:\infty}, \mu^{s-1}, \hat{\phi}^{s-1}) = \ell(\hat{\phi}^{s-1}), \quad (36)$$

input : $x^s_{w,d}, D_s, \tau_0, \kappa, K, \alpha, \beta$.
output : $\hat{\boldsymbol{\phi}}^S_{K\times W}$.

1 **for** $s \leftarrow 1$ **to** S **do**
2 Load $x^s_{w,d}, d \in D_s$ in memory;
3 $\rho_s = (\tau_0+s)^{-\kappa}$; *initialize* $\boldsymbol{\mu}^s$;
$\hat{\theta}^s_d(k) \leftarrow \sum_w x^s_{w,d}\mu^s_{w,d}(k), d \in D_s$;
4 $\hat{\phi}^s_w(k) \leftarrow \hat{\phi}^s_w(k) + \sum_d x^s_{w,d}\mu^s_{w,d}(k), d \in D_s$;
5 $\hat{\phi}^s(k) \leftarrow \hat{\phi}^s(k) + \sum_{w,d} x^s_{w,d}\mu^s_{w,d}(k), d \in D_s$;
6 **repeat**
7 **for** $x^s_{w,d} \neq 0$ *in random order* **do**
8 $\hat{\theta}^s_{-w,d}(k) \leftarrow \hat{\theta}^s_d(k) - x^s_{w,d}\mu^s_{w,d}(k)$;
$\hat{\phi}^s_{w,-d}(k) \leftarrow \hat{\phi}_w(k) - x^s_{w,d}\mu^s_{w,d}(k)$;
$\hat{\phi}^s_{-(w,d)}(k) \leftarrow \hat{\phi}(k) - x^s_{w,d}\mu^s_{w,d}(k)$;
9 $\mu^s_{w,d}(k) \leftarrow \mathsf{normalize}([\hat{\theta}^s_d(k)+\alpha-1][\hat{\phi}^s_w(k)+\beta-1]/[\hat{\phi}^s(k)+W(\beta-1)])$;
10 $\hat{\theta}^s_d(k) \leftarrow \hat{\theta}^s_d(k) + x^s_{w,d}\mu^s_{w,d}(k)$;
$\hat{\phi}^s_w(k) \leftarrow \hat{\phi}^s_w(k) + x^s_{w,d}\mu^s_{w,d}(k)$;
$\hat{\phi}^s(k) \leftarrow \hat{\phi}^s(k) + x^s_{w,d}\mu^s_{w,d}(k)$;
11 **until** *converged*;
12 $\hat{\phi}^s_w(k) \leftarrow (1-\rho_s)\hat{\phi}^{s-1}_w(k) + \rho_s[\sum_d x^s_{w,d}\mu^s_{w,d}(k)]$;
$\hat{\phi}^s(k) \leftarrow (1-\rho_s)\hat{\phi}^{s-1}(k) + \rho_s[\sum_{w,d} x^s_{w,d}\mu^s_{w,d}(k)]$;
13 Free $x^s_{w,d}$, $\hat{\boldsymbol{\theta}}^s_{K\times D_s}$, $\boldsymbol{\mu}^s_{K\times NNZ_s}$ from memory;

Figure 11: OEM for LDA.

where $\boldsymbol{\mu}^{s+1:\infty}$ denotes responsibilities of unseen mini-batches from $s+1$ to ∞. Note that the lower bound (36) will not touch the log-likelihood (20) until all responsibilities for data streams have been updated in (21). The inequality (36) confirm that OEM can improve $\hat{\phi}^s$ to maximize the LDA's log-likelihood (20). In practice, OEM reads each mini-batch $x^s_{w,d}$ into memory and runs IEM until $\boldsymbol{\mu}^s$ converged. Then, the sufficien statistics $\hat{\phi}^s$ is updated by a linear combination between previous $\hat{\phi}^{s-1}$ and the updated sufficien statistics $\sum_d x^s_{w,d}\mu^s_{w,d}(k)$,

$$\hat{\phi}^s = (1-\rho_s)\hat{\phi}^{s-1} + \rho_s\left[\sum_d x^s_{w,d}\mu^s_{w,d}(k)\right]. \quad (37)$$

Since OEM only stores the current mini-batch $x^s_{w,d}$, the local parameters $\boldsymbol{\mu}^s$, $\hat{\boldsymbol{\theta}}_{K\times D_s}$ and the global parameter $\hat{\phi}^s$ in memory, it is easy to process big data stream with low space complexities $\mathcal{O}(K\times(D_s+W+NNZ_s))$, where D_s is the number of documents and NNZ_s the number of nonzero elements in the sth mini-batch. Fig. 11 summarizes the OEM algorithm for LDA, where online BP [37] and stochastic CVB [9] are some implementations of OEM. Note that OEM can revisit previous processed mini-batch.

8. REFERENCES

[1] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. Smola. Scalable inference in latent variable models. In *WSDM*, pages 123–132, 2012.

[2] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh. On smoothing and inference for topic models. In *UAI*, pages 27–34, 2009.

[3] D. M. Blei. Introduction to probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

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

[5] T. Broderick, N. Boyd, A. Wibisono, A. C. Wilson, and M. I. Jordan. Streaming variational bayes. In *NIPS*, pages 1727–1735, 2013.

[6] O. Cappé and E. Moulines. Online expectation-maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B*, 71(3):593–613, 2009.

[7] N. de Freitas and K. Barnard. Bayesian latent semantic analysis of multimedia databases. Technical report, University of British Columbia, 2001.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[9] J. R. Foulds, L. Boyles, C. DuBois, P. Smyth, and M. Welling. Stochastic collapsed variational bayesian inference for latent Dirichlet allocation. In *KDD*, pages 446–454, 2013.

[10] T. L. Griffith and M. Steyvers. Finding scientifi topics. *Proc. Natl. Acad. Sci.*, 101:5228–5235, 2004.

[11] M. Hoffman, D. Blei, and F. Bach. Online learning for latent Dirichlet allocation. In *NIPS*, pages 856–864, 2010.

[12] D. Jiang, K. W.-T. Leung, and W. Ng. Fast topic discovery from web search streams. In *WWW*, pages 949–960, 2014.

[13] A. Q. Li, A. Ahmed, S. Ravi, and A. J. Smola. Reducing the sampling complexity of topic models. In *KDD*, 2014.

[14] P. Liang and D. Klein. Online EM for unsupervised models. In *Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the ACL*, pages 611–619, 2009.

[15] Z. Liu, Y. Zhang, E. Y. Chang, and M. Sun. PLDA+: Parallel latent dirichlet allocation with data placement and pipeline processing. *ACM Trans. Intell. Syst. Technol.*, 2(3):1–18, 2011.

[16] D. Mimno, M. D. Hoffman, and D. M. Blei. Sparse stochastic inference for latent Dirichlet allocation. In *ICML*, 2012.

[17] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[18] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifie incremental, sparse, and other variants. *Learning in Graphical Models*, 89:355–368, 1998.

[19] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed algorithms for topic models. *J. Mach. Learn. Res.*, 10:1801–1828, 2009.

[20] F. Niu, B. Recht, C. Re, and S. J. Wright. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, pages 693–701, 2011.

[21] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling. Fast collapsed Gibbs sampling for latent Dirichlet allocation. In *KDD*, pages 569–577, 2008.

[22] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[23] B. S. S. Ahn and M. Welling. Distributed stochastic gradient mcmc. In *ICML*, 2014.

[24] M. W. Schmidt, N. L. Roux, and F. Bach. Minimizing finit sums with the stochastic average gradient. *CoRR*, abs/1309.2388, 2013.

[25] A. Smola and S. Narayanamurthy. An architecture for parallel topic models. In *PVLDB*, pages 703–710, 2010.

[26] Y. W. Teh, D. Newman, and M. Welling. A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation. In *NIPS*, pages 1353–1360, 2007.

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

[28] Y. Wang, X. Zhao, Z. Sun, H. Yan, L. Wang, Z. Jin, L. Wang, Y. Gao, C. Law, and J. Zeng. Peacock: Learning long-tail topic features for industrial applications. *ACM Transactions on Intelligent Systems and Technology*, 2015.

[29] F. Yan, N. Xu, and Y. Qi. Parallel inference for latent Dirichlet allocation on graphics processing units. In *NIPS*, pages 2134–2142, 2009.

[30] J.-F. Yan, J. Zeng, Z.-Q. Liu, and Y. Gao. Towards big topic modeling. page arXiv:1311.4150, 2013.

[31] L. Yao, D. Mimno, and A. McCallum. Efficien methods for topic model inference on streaming document collections. In *KDD*, pages 937–946, 2009.

[32] 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 compute clusters. page arXiv:1412.1576 [stat.ML], 2014.

[33] H. Yun, H.-F. Yu, C.-J. Hsieh, S. V. N. Vishwanathan, and I. Dhillon. NOMAD: Nonlocking, stOchastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion. In *PVLDB*, pages 975–986, 2014.

[34] J. Zeng. A topic modeling toolbox using belief propagation. *J. Mach. Learn. Res.*, 13:2233–2236, 2012.

[35] J. Zeng, W. K. Cheung, and J. Liu. Learning topic models by belief propagation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(5):1121–1134, 2013.

[36] J. Zeng, Z.-Q. Liu, and X.-Q. Cao. A new approach to speeding up topic modeling. page arXiv:1204.0170 [cs.LG], 2012.

[37] J. Zeng, Z.-Q. Liu, and X.-Q. Cao. Online belief propagation for topic modeling. *arXiv:1210.2179 [cs.LG]*, 2012.

[38] K. Zhai, J. Boyd-Graber, and N. Asadi. Mr. LDA: A fl xible large scale topic modeling package using variational inference in MapReduce. In *WWW*, pages 879–888, 2012.

[39] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin. A fast parallel SGD for matrix factorization in shared memory systems. In *ACM Recommender Systems*, 2013.