# Peacock: Learning Long-Tail Topic Features for Industrial Applications

YI WANG, XUEMIN ZHAO, ZHENLONG SUN, HAO YAN, LIFENG WANG, ZHIHUI JIN, and LIUBIN WANG, Tencent
YANG GAO, School of Computer Science and Technology, Soochow University
CHING LAW, Tencent
JIA ZENG, School of Computer Science and Technology, Soochow University & Huawei Noah's Ark Lab

Latent Dirichlet allocation (LDA) is a popular topic modeling technique in academia but less so in industry, especially in large-scale applications involving search engine and online advertising systems. A main underlying reason is that the topic models used have been too small in scale to be useful; for example, some of the largest LDA models reported in literature have up to $10^3$ topics, which difficultly cover the long-tail semantic word sets. In this article, we show that the number of topics is a key factor that can significantly boost the utility of topic-modeling systems. In particular, we show that a "big" LDA model with at least $10^5$ topics inferred from $10^9$ search queries can achieve a significant improvement on industrial search engine and online advertising systems, both of which serve hundreds of millions of users. We develop a novel distributed system called *Peacock* to learn big LDA models from big data. The main features of Peacock include hierarchical distributed architecture, real-time prediction, and topic de-duplication. We empirically demonstrate that the Peacock system is capable of providing significant benefits via highly scalable LDA topic models for several industrial applications.

Categories and Subject Descriptors: H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Latent Dirichlet allocation, big topic models, big data, long-tail topic features, search engine, online advertising systems

**ACM Reference Format:**
Yi Wang, Xuemin Zhao, Zhenlong Sun, Hao Yan, Lifeng Wang, Zhihui Jin, Liubin Wang, Yang Gao, Ching Law, and Jia Zeng. 2015. Peacock: Learning long-tail topic features for industrial applications. ACM Trans. Intell. Syst. Technol. 6, 4, Article 47 (July 2015), 23 pages.
DOI: http://dx.doi.org/10.1145/2700497



## 1. INTRODUCTION

In academia, latent Dirichlet allocation (LDA) [Blei et al. 2003] is a popular topic modeling technique that uses an unsupervised learning algorithm to infer semantic word sets called *topics*. However, very few successes of LDA have been reported in the industry, mostly because the largest LDA models reported in the literature [Yan et al.

Authors' addresses: Y. Wang, X. Zhao, Z. Sun, H. Yan, L. Wang, Z. Jin, L. Wang, and C. Law, Tencent; Y. Gao, School of Computer Science and Technology, Soochow University, Shuzhou 215006, China, and Collaborative Innovation Center of Novel Software Technology and Industrialization; J. Zeng (corresponding author), School of Computer Science and Technology, Soochow University, Shuzhou 215006, China, Collaborative Innovation Center of Novel Software Technology and Industrialization, and Huawei Noah's Ark Lab, Hong Kong; email: zeng.jia@acm.org.

DOI: http://dx.doi.org/10.1145/2700497

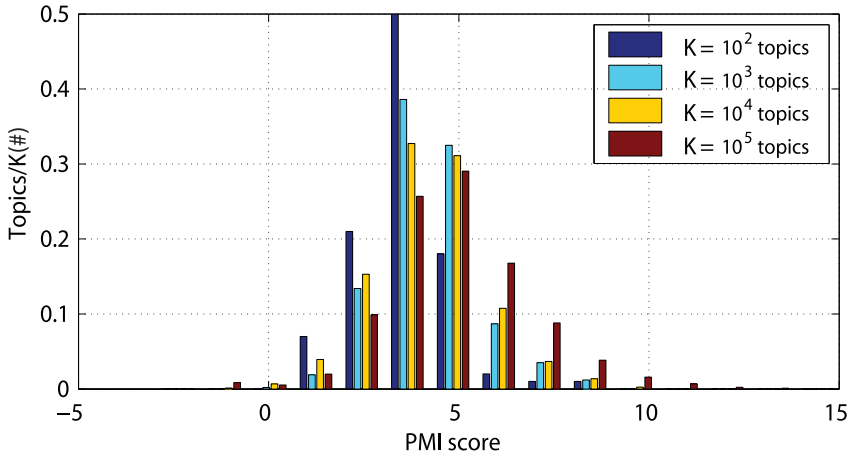


Fig. 1. Percentage of topics over the PMI score by LDA models with different number of topics $K$.

2009; Newman et al. 2007; Asuncion et al. 2008; Wang et al. 2009; Liu et al. 2011; Zhai et al. 2012] have up to $10^3$ topics, which cannot cover completely the long-tail semantic word sets in big data. Industrial applications like search engines and online advertising require the capability of learning many semantic word sets (or topics) that cover a large part of human knowledge, particularly the long-tail part. As reported by the Linguistic Data Consortium (LDC), there are millions of vocabulary words in English, Chinese, Spanish, and Arabic [Graff and Cieri 2003]. Taking polysemy and synonyms into consideration, a rough estimate of the number of word senses is close to the same magnitude of vocabulary words—that is, $10^5$ or $10^6$ topics for semantics of long-tail word sets.

To the best of our knowledge, the number of topics in applications mentioned earlier is around two to three orders of magnitude larger than that in the current state of the art [Smola and Narayanamurthy 2010; Ahmed et al. 2012]. The effectiveness of the large number of topics is inspired by Li et al. [2008], which proposes a MapReduce-based *frequent itemset mining* algorithm to find the long-tail word sets. We notice that there is a word set containing the words "whorf piraha chomsky anthropology linguistics." Using Web search, we find that this word set has a clear semantic meaning on the research by Whorf and Chomsky, which is related to anthropology and linguistics based on their study of the Piraha language. However, it is a regret that the *frequent itemset mining* algorithm cannot interpret new documents out of the training corpus. Fortunately, LDA overcomes this shortcoming and can infer highly interpretable and semantically coherent topics from new documents.

To illustrate the advantage of the large number of topics $K$, we use pointwise mutual information (PMI) to measure the interpretability or semantic coherence of topics [Newman et al. 2010]. The higher PMI score corresponds to a better topic quality. First, we learn LDA models with different numbers of topics $K \in \{10^2, 10^3, 10^4, 10^5\}$ on the SOSO dataset described later in Section 4.1. Second, we calculate the PMI score of each topic and obtain the topic histograms over the PMI score bucket of different LDA models. Figure 1 shows the percentage of topics over the PMI score by four LDA models with $10^2$, $10^3$, $10^4$, and $10^5$ topics, respectively. With the increase in topics from $10^2$ to $10^5$, we see that the topic histograms shift toward the larger mean PMI score—in other words, more and more topics have higher PMI scores. This phenomenon suggests that larger LDA models tend to encode more interpretable and semantically coherent topics.

In this article, we confirm that big LDA models with at least $10^5$ topics can achieve a significant improvement in two industrial applications: search engine and online advertising systems. This finding motivates us to pursue scalable topic modeling systems for big data. To achieve this goal, we develop a hierarchical distributed learning system called *Peacock*, which can generate a much larger number of topics than before. For example, Peacock can learn at least $10^5$ topics from $10^9$ search queries. This improvement is nontrivial and raises many new technical challenges. First, how can the system be made scalable to process big query data and LDA parameters with fault tolerance? Second, how can real-time topic prediction be done for new queries and how can duplicate topics be removed to obtain high-quality ones? Finally, how can big LDA models be integrated into existing search engine and online advertising systems for better performance? We address these technical issues and summarize our contributions as follows:

—We design a new hierarchical distributed architecture including model parallelism to handle a large number of LDA parameters as well as data parallelism to handle massive training corpora. We also use the pipeline and lock-free techniques to reduce communication and synchronization costs. This architecture runs on a computer cluster including thousands of CPU cores, which can learn $\geq 10^5$ topics from $\geq 10^9$ search queries, around two orders of magnitude larger than the current state of the art reported in the literature [Smola and Narayanamurthy 2010; Ahmed et al. 2012].
—When performing topic modeling for big data, we solve two new practical problems in real-world applications: real-time prediction and topic de-duplication. A new real-time prediction algorithm called *RT-LDA* is developed to infer the topic distributions of unseen queries in search engine and online advertising systems. As far as topic de-duplication is concerned, we use two methods: (1) learning asymmetric Dirichlet priors [Wallach et al. 2009a] and (2) clustering similar topics by their $L_1$-similarities.
—We examine the effectiveness of big LDA models in two online industrial applications: search engine and online advertising systems. The performance improvements in both systems grow with the increasing number of the learned topics. We observe a significant improvement on search relevance when the number of topics increases from $10^3$ to $10^4$. In addition, the topic features significantly improve the accuracy of ad click-through rate prediction when the number of topics increases from $10^4$ to $10^5$.

The rest of this article is organized as follows. In Section 2, we discuss related work. In Section 3, we present the hierarchical distributed architecture, real-time prediction, and topic de-duplication in the proposed Peacock system. In Section 4, we show that Peacock is more scalable to the larger number of topics than the state-of-the-art industrial solution Yahoo!LDA [Smola and Narayanamurthy 2010; Ahmed et al. 2012]. Section 5 shows two online industrial applications of Peacock: search engine and online advertising systems. In Section 6, we make conclusions and envision future work.

## 2. RELATED WORK AND DISCUSSION

There are five categories of batch inference algorithms proposed to estimate LDA parameters: variational Bayes (VB) [Blei et al. 2003], Gibbs sampling (GS) [Griffiths and Steyvers 2004; Porteous et al. 2008; Yao et al. 2009], expectation propagation (EP) [Minka and Lafferty 2002], belief propagation (BP) [Zeng et al. 2013; Zeng 2012], and collapsed variational Bayes (CVB or CVB0 with zero-order approximation) [Teh et al. 2006; Asuncion et al. 2009; Sato and Nakagawa 2012]. Except for GS, all other inference methods are based on the coordinate ascent algorithm [de Freitas and Barnard

2001; Murphy 2012], which first calculates the topic posterior distribution over each word token and then updates the parameters based on the inferred posterior distribution. Although CVB0 and BP converge much faster and produce higher held-out log-likelihood [Wallach et al. 2009b] than GS, they require storing the posterior probability matrix of all words in memory. The size of this matrix increases linearly with the number of unique document/word pairs and the number of topics. It is difficult to distribute this big matrix to a common computer cluster when the number of words and the number of topics are very large. In addition, CVB0 and BP store the parameters of document-topic and topic-word distributions in double precision, consuming more memory to handle sparse datasets. Most parallel inference solutions of LDA choose batch GS algorithms because they are more memory efficient than other algorithms [Smola and Narayanamurthy 2010; Ahmed et al. 2012; Porteous et al. 2008; Wang et al. 2009; Newman et al. 2007; Liu et al. 2011; Yan et al. 2009]. 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 (e.g., higher held-out log-likelihood) than VB [Griffiths and Steyvers 2004]. So, GS is generally agreed to be a more scalable choice in many parallel LDA solutions. We will discuss how to distribute an accelerated GS algorithm with low time and space complexities called *SparseLDA* later in Section 3.1.

Previous distributed LDA systems have explored two main architectures: (1) parallel computing using processors tightly coupled with shared memory [Yan et al. 2009] and and (2) distributed computing using processors loosely connected via network [Newman et al. 2007; Asuncion et al. 2008; Wang et al. 2009; Liu et al. 2011; Zhai et al. 2012; Yan et al. 2013, 2014]. Yahoo!LDA [Smola and Narayanamurthy 2010; Ahmed et al. 2012] can be viewed as a hybrid parallel architecture using the shared memory technique over multiple machines based on the *memcached* technique. Mr. LDA [Zhai et al. 2012] distributes the batch VB algorithm to the MapReduce framework, which requires frequent I/O operations causing the slow speed. However, all reported distributed LDA systems learn up to $10^3$ topics, although $10^3$ might be far less than the real number of semantics or word senses in human language. On the other hand, LDA is a nonnegative matrix factorization method [Buntine and Jakulin 2005; Zeng et al. 2013] in which many parallel matrix factorization architectures can be used. However, recent parallel matrix factorization architectures [Gemulla et al. 2011; Zhuang et al. 2013] have difficulty in learning $10^5$ topics because they focus on only low-rank approximation to the big sparse matrix. The rank is often very low, such as $10^2 \sim 10^3$, where the rank in matrix factorization has the same meaning with the number of topics in LDA. Unlike previous distributed LDA solutions, Peacock introduces both data and model parallelism for big data ($\geq 10^9$ documents) and big LDA models ($\geq 10^9$ parameters including topic assignment vector and topic-word matrix) within a hierarchical distributed architecture, which uses pipeline and lock-free techniques to reduce both communication and synchronization costs. In addition, Peacock has two new components—real-time prediction and topic de-duplication, which play important roles in real-world applications.

Recently, online LDA algorithms [Hoffman et al. 2010; Zeng et al. 2012b; Mimno et al. 2012; Patterson and Teh 2013; Broderick et al. 2013; Foulds et al. 2013] have attracted intensive research interests for two reasons. First, except for online Bayesian updating [Broderick et al. 2013], most online LDA algorithms combine the stochastic optimization framework [Robbins and Monro 1951] with the corresponding batch LDA algorithms, which theoretically can converge to the local optimal point of the LDA objective function. Second, online algorithms partition the entire dataset into several mini-batches. They load each mini-batch in memory for online processing and discard it after one look. This streaming method significantly reduces the memory consumption

for big data and big models. However, given the same amount of training samples, batch algorithms converge significantly faster and yield higher held-out log-likelihood than online counterparts [Zeng et al. 2012a, 2012b]. The main reason is that the convergence rate of stochastic algorithms is slower than that of batch algorithms, which has been discussed in Schmidt et al. [2013]. If we have enough computing resources, it is better to distribute batch algorithms than online ones. If the memory is limited and the data come in the streaming manner, we prefer distributing online algorithms. For example, D-SGLD [Ahn et al. 2014] is based on a stochastic gradient Monte Carlo Markov chain (MCMC) and is distributed with adaptive load balance by making the faster workers work longer until the slower workers finish their tasks. POBP [Yan et al. 2013] parallelizes the online BP algorithm and has a dynamic communication scheduling scheme to reduce the overall cost in processing big data streams. As discussed earlier, both batch and online algorithms have their advantages and disadvantages. Although Peacock is designed for distributed batch algorithms, its architecture can be readily extended to distributed online algorithms for streaming data. In this case, Peacock just replaces batch algorithms with online counterparts without changing the hierarchical architecture, which will be studied in our future work.

Other topic models such as hierarchical Dirichlet processes (HDP) [Teh et al. 2004] and author-topic models (ATM) [Steyvers et al. 2004] have similar batch inference algorithms with LDA, such as GS and VB [Teh et al. 2004; Hoffman et al. 2013]. These inference algorithms estimate similar parameters as those in LDA—for example, the multinomial parameters for topic-word distributions. Therefore, the parallel implementation of these inference algorithms can be also deployed in Peacock. For instance, ATM can be implemented in Peacock by replacing the multinomial parameters over documents with those over authors [Zeng et al. 2013].

## 3. THE PEACOCK SYSTEM

In this section, we first introduce an accelerated GS algorithm called *SparseLDA* [Yao et al. 2009] for learning LDA. Then, we show how to distribute the GS algorithm in the hierarchical architecture with model and data parallelism to handle the large number of LDA parameters and training samples. In this new architecture, we focus on implementing the following techniques: (1) the distributed GS algorithm, (2) pipeline efficient communication, (3) lock-free synchronization, and (4) fault tolerance. Finally, we introduce how to solve two new problems of big LDA models in real-world industrial applications: (1) real-time topic prediction of new unseen queries/documents and (2) topic de-duplication for better quality and performance. Table I summarizes some important notations used in this article.

### 3.1. Accelerated Gibbs Sampling Inference for Latent Dirichlet Allocation

LDA allocates a set of thematic topic labels, $\boldsymbol{z} = \{z_{ivd} = k\}$, to explain the word tokens, $\boldsymbol{x}_{D\times V} = \{x_{ivd} = \{0, 1\}\}$, in the document-word co-occurrence matrix $\boldsymbol{x}_{D\times V}$, where $i$ is the word token index, $1 \le v \le V$ 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 objective of LDA is to maximize the joint probability $P(\boldsymbol{x}, \theta, \phi|\alpha, \beta)$, where $\theta_{D\times K}$ and $\phi_{V\times K}$ are two nonnegative matrices of multinomial parameters for document-topic and topic-word distributions, satisfying $\sum_k \theta_{dk} = 1$ and $\sum_v \phi_{vk} = 1$. Both multinomial matrices are generated by two Dirichlet distributions with hyperparameters $\alpha$ and $\beta$.

Since we aim to learn $K \ge 10^5$ topic from $D \ge 10^9$ search queries, we choose to distribute SparseLDA, an accelerated sparse GS inference algorithm [Yao et al. 2009], whose time and space complexities are insensitive to the number of topics $K$. In GS, the memory is used to maintain three LDA parameter count matrices: a matrix $\Phi_{V\times K}$

Table I. Definitions of Notation

| | |
|---|---|
| $1 \le d \le D$ | Document index |
| $1 \le v \le V$ | Vocabulary word index |
| $1 \le k \le K$ | Topic index |
| $1 \le m \le M$ | Model and data shard index |
| $1 \le c \le C$ | Configuration index |
| $x_{ivd}$ | Word token in bag-of-word representation |
| $z_{ivd}$ | Topic label for each word token |
| $\neg ivd$ | All word tokens except the $ivd$ word token |
| $\Theta_{dk}$ | Number of tokens in document $d$ assigned to the topic $k$ |
| $\theta_{dk}$ | Multinomial parameters for document-topic distribution |
| $\Phi_{vk}$ | Number of tokens of type $v$ assigned to the topic $k$ |
| $\phi_{vk}$ | Multinomial parameters for topic-word distribution |
| $\Psi_k$ | $\sum_v \Phi_{vk}$ |
| $\alpha_k$ | Asymmetric Dirichlet hyperparameter |
| $\beta$ | Symmetric Dirichlet hyperparameter |
| $l_d$ | Document length for estimating asymmetric hyperparameter $\alpha_k$ |
| $\Omega_{kn}$ | Count matrix for estimating asymmetric hyperparameter $\alpha_k$ |
| $T$ | Number of slots in the pipeline communication |
| $L$ | Packet size in the pipeline communication |

in which each element is the total number of the vocabulary word $v$ assigned to the topic $\boldsymbol{z}_{ivd} = k$, a matrix $\Theta_{D\times K}$ in which each element is the total number of topic $\boldsymbol{z}_{ivd} = k$ assignments in each document $d$, and a count vector $\Psi_k = \sum_{v=1}^{V} \Phi_{vk}$, in which each element is the number of topic $k$ assignments in the training corpus. The relation between LDA multinomial parameters and count matrices are as follows:

$$\theta_{dk} = \frac{\Theta_{dk} + \alpha}{\sum_{k=1}^{K} \Theta_{dk} + K\alpha}, \tag{1}$$

$$\phi_{vk} = \frac{\Phi_{vk} + \beta}{\Psi_k + V\beta}. \tag{2}$$

In each iteration, the GS algorithm updates the topic assignment $z_{ivd} = k$ of every observed word token $x_{ivd} = 1$ in the training corpus by randomly drawing a topic $z_{ivd} = k$ from the collapsed posterior distribution,

$$P(z_{ivd} = k, x_{ivd} = 1 \mid \boldsymbol{z}_{\neg ivd}, \boldsymbol{x}_{\neg ivd}, \alpha, \beta) \propto \frac{\Phi_{vk}^{\neg ivd} + \beta}{\Psi_k^{\neg ivd} + V\beta} \left(\Theta_{dk}^{\neg ivd} + \alpha\right), \tag{3}$$

where $\neg ivd$ means that the corresponding word token and topic $z_{ivd} = k$ is excluded from the count matrices. After the new topic assignment $z_{ivd} = k$ is sampled, the corresponding elements in the count matrices $\Psi_k$, $\Phi_{vk}$, and $\Theta_{dk}$ are updated immediately. SparseLDA divides Equation (3) into three parts:

$$\begin{aligned} P(z_{ivd} = k, x_{ivd} = 1 \mid \boldsymbol{z}_{\neg ivd}, \boldsymbol{x}_{\neg ivd}, \alpha, \beta) \propto\ & \frac{\alpha\beta}{\Psi_k^{\neg ivd} + V\beta} + \frac{\Theta_{dk}^{\neg ivd}\beta}{\Psi_k^{\neg ivd} + V\beta} \\ & + \frac{(\alpha + \Theta_{dk}^{\neg ivd})\Phi_{vk}^{\neg ivd}}{\Psi_k^{\neg ivd} + V\beta}. \end{aligned} \tag{4}$$

Due to sparsity of the topic posterior probability $P(z_{ivd} = k, x_{ivd} = 1 \mid \boldsymbol{z}_{\neg ivd}, \boldsymbol{x}_{\neg ivd}, \alpha, \beta)$, randomly sampling these three parts does not need to calculate $K$ times. As a result, SparseLDA has a time complexity insensitive to the number of topics $K$. Moreover, it

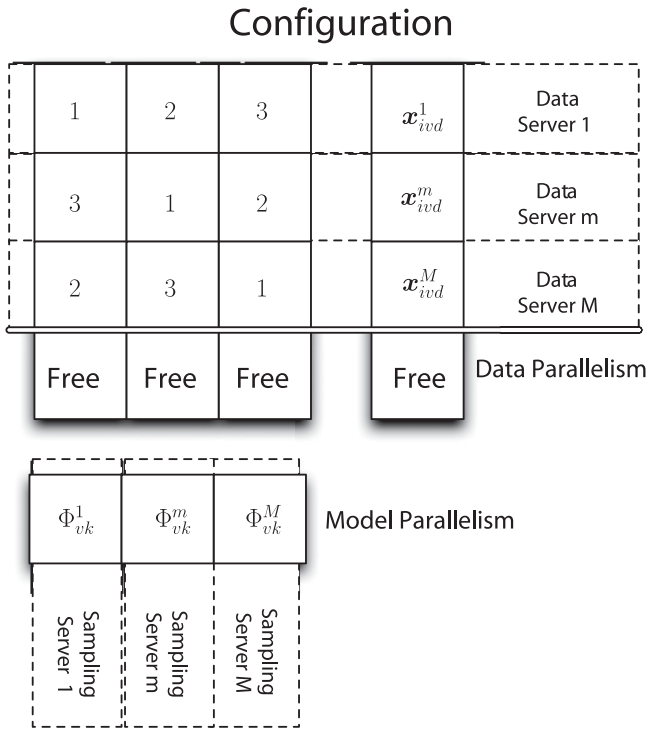


Fig. 2. One configuration in the first layer of the hierarchical distributed architecture.

has a low space complexity because it stores only the topic assignment vector $\boldsymbol{z}$ rather than the matrix $\Theta_{D\times K}$ in memory, where the size of $\boldsymbol{z}$ is equal to the total number of word tokens in corpus, which is irrelevant with the number of topics $K$. SparseLDA reorganizes the corresponding $\boldsymbol{z}_{ivd}$ into the document-specific vector $\Theta_{dk}$ on the fly. If a vocabulary word $v$ has a total of $N \ll K$ occurrences in training corpus, the $v$th row of the parameter matrix $\Phi_{vk}$ only needs to store up to $N$ rather than $K$ values.

### 3.2. Hierarchical Distributed Architecture

The key challenge is to store the word tokens $\boldsymbol{x}$, the topic assignment vector $\boldsymbol{z}$, and the large LDA parameter matrix $\Phi_{V\times K}$, when $V \geq 10^5$, $K \geq 10^5$, and $D \geq 10^9$ in industrial applications. For example, the count matrix $\Phi_{V\times K}$ alone takes at least tens of gigabytes when learning $10^5$ topics, whereas modern computer clusters are composed of commodity computers [Dean and Ghemawat 2008] with few gigabytes of memory (e.g., 2GB memory). Therefore, we propose the hierarchical distributed architecture to solve this large-scale problem, which contains a configuration of servers to handle both big data and the big LDA model in Figure 2. More specifically, the distributed GS algorithm can be executed by a configuration of the following servers:

(1) *Model parallelism*: We partition the parameter $\Phi_{V\times K}$ matrix by rows $1 \leq v \leq V$ into $1 \leq m \leq M$ model shards, $\{\Phi^1_{vk}, \ldots, \Phi^m_{vk}, \ldots, \Phi^M_{vk}\}$. We use *$M$ sampling servers*, where the $m$th sampling server maintains the $\Phi^m_{vk}$ shard and local copies of $\Psi^m_k$ and $\alpha^m_k$ in memory. The value of $M$ should make each model shard $\Phi^m_{vk}$ small enough to fit in the memory of a sampling server. Each sampling server runs the SparseLDA algorithm to update the topic assignments $\boldsymbol{z}^m_{ivd}$ in training blocks sent from the data servers. The sampling algorithm will update the $\Phi^m_{vk}$ shard at the $m$th sampling

server. At the end of the training process, all sampling servers output their $\Phi^m_{vk}$ model shards.

(2) *Data parallelism*: We partition the word tokens $\boldsymbol{x}_{D\times V}$ and their topic assignments $\boldsymbol{z}$ by rows into $1 \le m \le M$ shards, $\{\boldsymbol{x}^1_{idv}, \boldsymbol{z}^1_{idv}, \ldots, \boldsymbol{x}^m_{idv}, \boldsymbol{z}^m_{idv}, \ldots, \boldsymbol{x}^M_{idv}, \boldsymbol{z}^M_{idv}\}$. We use $M$ *data servers*, where each loads a data shard $\boldsymbol{x}^m_{idv}$ and the corresponding $\boldsymbol{z}^m_{idv}$ shard in memory. The value of $M$ should make each data shard and its corresponding $\boldsymbol{z}$ shard small enough to fit in the memory of a data server. The data servers send word tokens and their topic assignments shards to the corresponding sampling servers, which update the topic assignments $\boldsymbol{z}^m_{idv}$ and the model parameters $\Phi^m_{vk}$. After processing each segment of data shards, the sampling servers send back the changed topic assignments $\Delta\boldsymbol{z}^m_{idv}$ to data servers, which write $\Delta\boldsymbol{z}^m_{idv}$ back to disks for fault recovery.

As a result, the entire data has been partitioned into $M \times M$ blocks and each data server stores $M$ blocks. If the sampling server simultaneously obtains the same column of data segment sent by all data servers, it will read and write the topic assignments of the same vocabulary word under race conditions. For example in Figure 2, if the sampling server 1 receives simultaneously three data blocks $\{1, 3, 2\}$ sent by three data servers, it has a higher likelihood to change the topic assignments of the same vocabulary words, which causes serious read/write locks and I/O delays. To solve this problem, we design a lock-free parallel strategy similar to Yan et al. [2009], Gemulla et al. [2011], and Zhuang et al. [2013]. The sampling servers process blocks on the main diagonal sent by the data servers. Since only $\Phi^1_{vk}$ and $\{\boldsymbol{x}^1_{ivd}, \boldsymbol{z}^1_{ivd}\}$ are required for processing the first data block, $\Phi^2_{vk}$ and $\{\boldsymbol{x}^2_{ivd}, \boldsymbol{z}^2_{ivd}\}$ for the second, and $\Phi^3_{vk}$ and $\{\boldsymbol{x}^3_{ivd}, \boldsymbol{z}^3_{ivd}\}$ for the third, these three blocks can be processed simultaneously without conflicts in accessing $\Phi_{vk}$ and $\{\boldsymbol{x}_{ivd}, \boldsymbol{z}_{ivd}\}$. It is analogous to processing the data blocks on the second and the third diagonals. Because the global parameter $\Psi_k$ is needed for all block computation, we store a local copy of $\Psi^m_k$ in each sampling server and synchronize $\Psi^m_k$ by a coordinator server after processing each diagonal of blocks. We refer to each of nonconflicting $M$ shards as a *segment*. Figure 2 shows three segments, $\{1, 1, 1\}$, $\{2, 2, 2\}$, and $\{3, 3, 3\}$, when $M = 3$.

The scalability of one configuration in Figure 2 is limited. Increasing the number of sampling servers $M$ indicates the increasing of vertical partitions of the training corpus $\boldsymbol{x}_{D\times V}$ as well as the model parameters $\Phi_{V\times K}$ in rows. Thus, the number of data servers $M$ would be less than the size of vocabulary $V$. However, the size of vocabulary words in one sampling server should be larger than a value (e.g., $\ge 10^3$) for a better efficiency. In this case, $M$ cannot be very large in practice. When $V \ll D$, it is difficult to use the limited $M$ data servers to store big data in one configuration. As a result, we need to build multiple configurations and use a set of $M$ *aggregation servers* to synchronize the global model parameter $\Phi_{vk}$ from different configurations.

Figure 3 shows the hierarchical distributed architecture containing two layers. In layer 1, there are $1 \le c \le C$ configurations as shown in Figure 2. For simplicity, we do not illustrate the data servers in each configuration. In layer 2, there are $M$ aggregation servers to connect corresponding sampling servers. At the end of each GS iteration, all sampling servers in all configurations report their model parameter change $\Delta\Phi^m_{vk}$ to aggregation servers [Newman et al. 2007; Ahmed et al. 2012]. Notice that the $m$th sampling server in the layer 1 configuration reports only to the $m$th aggregation server in layer 2. After the aggregation from all configurations, the aggregation servers distribute the updated global model $\Phi_{V\times K}$ to all configurations in layer 1. In practice, the aggregation does not have to be done in every iteration. Each configuration can run independently for several iterations before the model aggregation in layer 2. This strategy is analogous to the Robbins-Monro stochastic optimization [Robbins and Monro 1951],

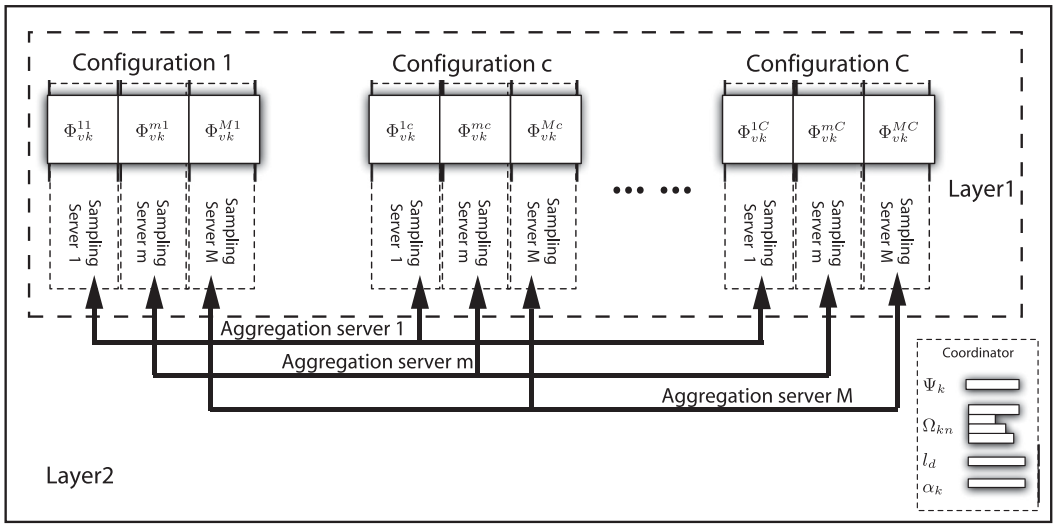


Fig. 3. The hierarchical distributed architecture in Peacock. The first layer contains multiple configurations in Figure 2. The second layer contains $M$ aggregation servers and one coordinator server for global parameter synchronization and asymmetric prior estimation. This architecture can readily scale up to hundreds of machines having thousands of cores to learn at least $10^5$ topics from $10^9$ search queries.

where a minibatch estimate of model parameters can be viewed as a stochastic approximation to them. Therefore, this asynchronous parameter update method in Peacock is likely to work because Robbins-Monro stochastic optimization works. Previous results [Newman et al. 2007; Yan et al. 2014] also confirm that the asynchronous and delayed synchronization of the global parameters will not degrade LDA accuracy very much.

In layer 2, we also use a *coordinator* to control the cooperative work of sampling, data, and aggregation servers. The coordinator is also responsible for updating and redistributing the global copy of $\Psi_k$ to sampling servers and optimizing the asymmetric Dirichlet prior $\alpha_k$. To optimize $\alpha_k$, the coordinator has to maintain a histogram of document lengths, $l_d$, and collects a matrix, $\Omega_{kn}$, storing the number of documents in which topic $k$ appears $n$ times [Wallach et al. 2009a] from the data servers. Likewise to the synchronization of $\Phi_{vk}^{mc}$ in different configurations, we estimate these global parameters in the coordinator and distribute them to sampling servers in the asynchronous manner. For example, after all configurations run a few iterations, the coordinator aggregates $\Psi_k^m$ (the aggregation cost is small due to a simple sum operation) and broadcasts the updated global parameter to all sampling servers. All of these servers are developed using Google's Go programming language—a compiled concurrent system programming language. The parallel machine learning systems could benefit a lot from the native support of concurrent programming and convenient implementation of remote procedure calls (RPCs) provided by Go.

*3.2.1. Distributed Gibbs Sampling Algorithm.* Figure 4 shows the distributed GS algorithm executed by the coordinator, where the dot symbol denotes an RPC call. For example, in line 6, the coordinator calls procedure SetAlpha, which is exposed and executed by a sampling server, where the sampling follows Equation (4). The **par-for** in Figure 4 denotes the concurrent version of the commonly used control structure **for**. The **par-for** flattens the loop body and executes it in parallel. The **par-for** can be implemented using Go language elements of *channel* and *goroutine*, as shown by the open source project

```
1: procedure RunGibbsIteration(M, M)
2:   for segment ← 1..⌈M⌉ do
3:     SampleSegment(segment)
4:   α_k ← OptimizeHyperparameters(α_k, l_d, Ω_kn)
5:   par-for m ← 1..M
6:     sampling-server[m]. SetAlpha(α_k)

7: procedure SampleSegment(segment)
8:   par-for m ← 1..M
9:     data-server[m]. LoadShard(segment, m)
10:  for dig ← 1..M do                          ▷ dig indices diagonals
11:    par-for m ← 1..M
12:      data-server[(m+dig)%M]. WorkWithSampler(m) as follows:
13:      for each token x^m_ivd do
14:        Update z^m_ivd by sampling one topic from Equation (4)
15:    par-for m ← 1..M
16:      Ψ_k ← Ψ_k+ sampling-server[m]. GetDiffNt
17:    par-for m ← 1..M
18:      sampling-server[m].SetNt(Ψ_k)
19:  par-for m ← 1..M
20:    data-server[m].SaveShard(segment, m)
21:  par-for m ← 1..M
22:    Ω_kn ← Ω_kn+ data-server[m].CountNtn
```

Fig. 4. The distributed GS algorithm.

https://github.com/wangkuiyi/parallel, which is used in Peacock. C/C++ programmers can use the **par-for** provided by OpenMP.

In each iteration, the procedure RunGibbsIteration invokes SampleSegment to update the topic assignments segment by segment, where SampleSegment coordinates the $M$ data servers and the $M$ sampling servers to run the parallel SparseLDA algorithm. Each sampling server keeps a local $\Phi^m_{vk}$ and a global $\Psi_k$, and updates the topic assignment $z^m_{ivd}$ sent by the data server in line 14. SampleSegment also collects the matrix, $\Omega_{kn}$, which records the number of documents in which the topic assignment $k$ occurs $n$ times. This matrix is used by the procedure OptimizeHyperparams, which is described in Wallach et al. [2009a], to optimize the asymmetric prior, $\alpha_k$, at the end of each GS iteration. In addition to $\Omega_{kn}$, OptimizeHyperparameters also requires $l_d$, the vector recording the document lengths, which is counted in the first iteration and saved in the coordinator for later use.

*3.2.2. Pipeline for Efficient Communication.* Figure 4 shows that all network communications in Peacock happen in the form of RPCs. The largest fraction of *communication cost* lies in WorkWithSampler, where the data servers send data blocks to the sampling servers and wait for responses containing the updated topic assignment $\boldsymbol{z}^m_{ivd}$. We reduce the communication cost using the *pipeline* technique. To avoid the overflow of the network communication buffer, the data server sends just a few document fragments known as a package rather than sending a block in an RPC to the sampling server. Instead of waiting for the response from the sampling server before sending the next package, the data server sends $T$ packages concurrently. On the sampling server, there are multiple *goroutines*, a kind of light-weighted thread scheduled by the Go runtime system, to process these packages and respond to the data server. The data server maintains a data structure with $T$ slots, and each keeps track of an outgoing package. The data server clears a slot after receiving a response of the corresponding package,

Table II. Parameters of the Communication Pipeline and the Corresponding Communication Time

| $T$ | $L$ (KB) | Time (minutes) |
|---|---|---|
| 200,000 | 1 | 48.1 |
| 20,000 | 10 | 45.3 |
| 2,000 | 100 | 43.5 |
| 200 | 1,000 | 43.3 |
| 40 | 5,000 | 43.4 |
| 20 | 10,000 | 43.5 |
| 10 | 20,000 | 44.1 |
| 1 | 200,000 | 49.8 |

```
1: procedure SAMPLESEGMENT(segment)
2:   par-for m ← 1..M
3:     data-server[m].LOADSHARD(segment, m)
4:   par-for m ← 1..M
5:     for dig ← 1..M do                          ▷ dig indices diagonals
6:       data-server[(m+dig)%M].WORKWITHSAMPLER(m)
7:   par-for m ← 1..M
8:     Ψ_k ← Ψ_k+ sampling-server[m].GETDIFFNT
9:   par-for m ← 1..M
10:    sampling-server[m].SETNT(Ψ_k)
11:  par-for m ← 1..M
12:    data-server[m].SAVESHARD(segment, m)
13:  par-for m ← 1..M
14:    Ω_kn ← Ω_kn+ data-server[m].COUNTNTN
```

Fig. 5. The Faster sampling of corpus segments.

or getting a timeout. Once there are empty slots and packages to be processed, the data server would continue sending packages. In general, this pipeline optimizes the throughput by overlaps the sending, processing, and responding of packages.

Maximizing the throughput depends on finding the optimal configuration of two parameters: package size $L$ and pipeline capacity $T$. The product, $L \times T$, is proportional to the size of memory used as communication buffer. In practice, there would be an upper limit of buffer size, $y$, and we would make full use of it to get the maximum throughput. This can be written as the constraint function, $y = L \times T$, which is a curve on the two-dimensional space $L$ and $T$. The best configuration would be a point on this curve. In our computing environment, a practical $y$ value is 200MB. The measure of time consumption with respect to the curve $y = L \times T$ is shown in Table II, where the time consumption is larger at both ends of this curve and the optimal configuration lies in the middle of the curve. The variance is too small to be shown.

*3.2.3. Lock-Free Synchronization.* After issuing parallel executions of the loop body in Figure 4, the **par-for** does a synchronization operation that waits for the completions of all executions of sampling servers called the *synchronization lock problem*. We address this problem by three lock-free strategies. First, to avoid data skewness in each data block, we randomly shuffle data $\boldsymbol{x}_{D\times V}$ by rows and columns so that each block contains almost equal number of word tokens in practice [Zhuang et al. 2013].

Second, we can further reduce the synchronization cost of the **par-for** on line 4 of Figure 5 by balancing the workload of sampling servers. Because each sampling server processes a row of corpus blocks, it is desirable that the block rows contain similar word frequencies. This can be achieved using a pretraining scheduler, which assigns vocabulary words to $\Phi^m_{vk}$ shards. As with PLDA+ [Liu et al. 2011], we use the

weighted round-robin method for this word assignment. We first sort vocabulary words in descending order by their frequency, then pick the word with the largest frequency and assign it to the $\Phi^m_{vk}$ shard with the accumulative word frequency. Next, we update the accumulated word frequency of $\Phi^m_{vk}$. This placement process is repeated until all words have been assigned. Weighted round robin has been empirically shown to achieve a balanced load with a high probability [Berenbrink et al. 2008].

Finally, in Figure 4, the nested loop starting from line 10 invokes the **par-for** many times and introduces many waits. The problem can be relieved by swapping the inner and outer loop, as shown in Figure 5. In this change, two **par-for** structures at line 13 and line 15 are moved one upper level, which relaxes the aggregation and redistribution of vector $\Psi_k$ from a per-diagonal granularity to a per-segment granularity. This relaxed aggregation does not affect the correctness of the distributed GS algorithm analogous to the stochastic optimization framework [Robbins and Monro 1951]. Indeed, similar asynchronous optimization methods for updating model parameters have been confirmed to work well in lock-free parallel stochastic gradient descent algorithms [Niu et al. 2011; Johnson et al. 2013]. After swapping the **par-for** at line 11 with its outer loop at line 10, a data server might work with more than one sampling server simultaneously. However, this would introduce conflicts in accessing the $\boldsymbol{z}^m_{ivd}$ shard maintained by the data server. We address this conflict problem by two methods. First, we use $M + 1$ data servers as shown in Figure 2 denoted by "Free." These Free data servers provide additional conflict-free data blocks for computing without waiting for the completion of other sampling servers. The coordinator will schedule the finished sampling server to those conflict-free data blocks having the minimum number of visits. In this way, we ensure that all data blocks will have almost equal number of visits. Second, each data server maintains two copies of the $\boldsymbol{z}_{ivd}$ shard: $\boldsymbol{z}^{old}_{ivd}$ and $\boldsymbol{z}^{new}_{ivd}$ without conflicts. After receiving the response of updated package from the sampling server, the data server applies the difference between the response and $\boldsymbol{z}^{old}_{ivd}$ to $\boldsymbol{z}^{new}_{ivd}$.

*3.2.4. Fault Recovery.* Data parallelism also helps fault recovery, which is critical in large-scale machine learning. Considering that a parallel learning job may take days or even weeks, it is very likely that some workers fail or are preempted during the period. If the system cannot recover when it fails, we would have to restart the job from the beginning. Since the restart might fail again, the learning job would never finish. As the configurations in the layer 1 work independently within every few iterations, it is straightforward to restart any failed configuration based on the independent checkpoint on hard disks. The restart can be implemented simply using the SSH command, or sophisticated cluster management systems like Apache YARN. This architecture is similar to Google's architecture for parallel deep learning [Dean et al. 2012], which refers to configurations as models. More than achieving fine-grained fault recovery, this design also improves the parallelism and makes Peacock highly scalable.

### 3.3. Real-Time Prediction

In online applications such as search engine and online advertising systems, it is critical to predict latent semantics of new user queries in real time based on the large number of topics. In typical Internet services, the response time of backend servers is measured in milliseconds. Given LDA models with at least $10^5$ topics, few inference algorithms are efficient enough to do real-time prediction. Hence, we propose a real-time inference algorithm, RT-LDA, especially for prediction of new queries.

The basic idea of RT-LDA is to replace the sampling operation in Equation (4) by the max operation. This makes RT-LDA a hill climbing or coordinate ascent algorithm whose search path consists of line segments aligned with axes of the topic space, which is similar to the coordinate ascent using a one-dimensional Newton step [Yuan et al.

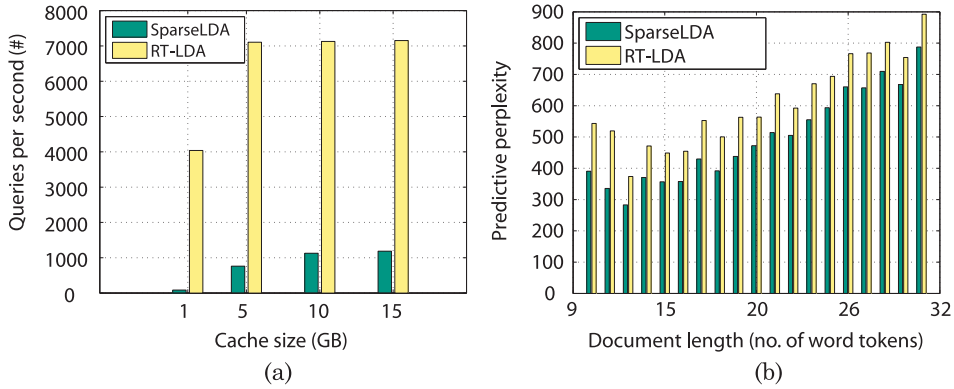


Fig. 6. Comparisons between RT-LDA and SparseLDA [Yao et al. 2009] in speed (a) and accuracy (b).

2010] widely used in learning regression models. The max operation in RT-LDA can be optimized using a cache-based technique. According to Equation (3), the max operator in RT-LDA is

$$\begin{aligned}&\max_{k\in[1,K]} P(z_{ivd}=k, x_{ivd}=1 \mid \boldsymbol{z}_{\neg ivd}, \boldsymbol{x}_{\neg ivd}, \alpha, \beta)\\ &= \max_{k\in[1,K]} \phi_{vk}(\Theta_{dk}+\alpha_k)\\ &= \max_{k\in[1,K]} \phi_{vk}\Theta_{dk} + \phi_{vk}\alpha_k,\end{aligned} \tag{5}$$

where $\phi_{vk}$ is the empirical probability matrix computed by Equation (2). In prediction, $\phi_{vk}$ and $\alpha_k$ are constants, whereas $\Theta_{dk}$ changes with the updating of topic assignments. This makes it viable to precompute $\max_k \phi_{vk}\alpha_k$, whose result is a sparse matrix $R$,

$$R_{vk} = \begin{cases}\phi_{vk}\alpha_k & \text{if } k = \max_{k'} \phi_{vk'}\alpha_{k'},\\ 0 & \text{otherwise.}\end{cases} \tag{6}$$

We save $R$ in a compact data structure that contains only $V$ nonzero elements. The compact $R$ is an approximation to Equation (5), where the error of the approximation is caused by nonzero elements in $\Theta_{dk}$. We rewrite Equation (5) as

$$\begin{aligned}&\max_{k\in[1,K]} P(z_{ivd}=k, x_{ivd}=1 \mid \boldsymbol{z}_{\neg ivd}, \boldsymbol{x}_{\neg ivd}, \alpha, \beta)\\ &= \max_k \left[ R^*_{vk}, \max_{\substack{k\in[1,K]\\ s.t.\Theta_{dk}>0}} \phi_{vk}(\Theta_{dk}+\alpha_k) \right],\end{aligned} \tag{7}$$

where $R^*_{vk}$ denotes the nonzero element in the column of $R$ corresponding to the vocabulary word $v$. Different from the max operation in Equation (5), which iterates over $k \in [1, K]$, the first max operation in Equation (7) compares two values, and the second max operation visits only nonzero elements in $\Theta_{dk}$. Suppose that the maximum number of nonzero elements in a query $d$ is the length of the query; Equation (7) makes RT-LDA significantly faster than SparseLDA when the number of topics $K \geq 10^5$.

Figure 6(a) compares the prediction speed between RT-LDA and SparseLDA. We use the the number of query-per-second (QPS) of RT-LDA and SparseLDA on a real backend inference server with 100% CPU load. The $x$-axis is the cache size. Generally, the larger cache leads to faster prediction, but the performance reaches the upper bound with the increase of cache size in gigabytes (GB). This setting implies that the response

time of prediction is the reciprocal of the QPS. We see that RT-LDA is about an order of magnitude faster than SparseLDA. Figure 6(b) compares RT-LDA and SparseLDA for their topic modeling accuracy measured in the predictive perplexity [Zeng et al. 2013], which is a standard performance measure for topic modeling accuracy. The lower perplexity means a higher topic modeling accuracy on new test data set. This experiment uses 1,200 Wikipedia titles by clustering them into 20 groups with various lengths. For all of these groups, we randomly select 10% as the test dataset and retain the remaining 90% as the training set. We see that the accuracy of the two algorithms, measured in perplexity, are very close. RT-LDA loses some tolerable topic modeling accuracy to get a faster speed compared to SparseLDA. We can further improve the effectiveness of RT-LDA by running multiple line searches in parallel and then averaging results of all of these parallel trails. Because RT-LDA is much more efficient than SparseLDA, the Peacock system can afford many parallel trails to extract topic features from new queries.

### 3.4. Topic De-Duplication

Although topic duplication has rarely been discussed in previous literature, it becomes a main challenge in topic feature engineering. As noted in Wallach et al. [2009a], when learning LDA, frequent words often dominate more than one topic, and the learned topics are similar to each other. We refer to these similar topics as *duplicates*. Generally, when learning $\geq 10^5$ topics, 20% to 40% of topics have duplicates in practice. If a query $d$ is 60% about topic $A$ and 40% about topic $B$, then the query $d$ is mainly about topic $A$. However, if topic $A$ has three duplicates, $A1$, $A2$, and $A3$, the GS algorithm would follow Equation (3) to scatter the 60% weight of $A$ in $d$ to $A1$, $A2$, and $A3$. If each duplicate gets a third of the original 60% of topic $A$, the interpretation of the query would mainly become about topic $B$, although the truth is that the query is mainly about topic $A$.

We remove topic duplicates by two methods. First, we learn the asymmetric Dirichlet prior $\alpha_k$ over the document-topic distributions [Wallach et al. 2009a], which substantially increases the robustness of LDA to variations in the number of topics and to the highly skewed word frequency distributions common in natural language. Asymmetric priors over document-topic distributions automatically combine similar topics into one large topic rather than splitting topics more uniformly by symmetric priors. In practice, we can set a very large $K = 10^6$ value at initial learning time and prune duplicates by the asymmetric Dirichlet prior. Those topics with very small Dirichlet priors would be automatically weighted trivial by RT-LDA (Section 3.3) at serving time. We find that this approach prevents common words from dominating many topics and thus leaves room for long-tail topics. Second, we cluster topic duplicates if their $L_1$-distance is below a threshold. The lower $L_1$-distance threshold means that we would remove more duplicates from a large number of topics.

## 4. EMPIRICAL STUDIES IN BIG DATA

We evaluate Peacock's topic modeling performance for big data by three performance measures: (1) speedup—the runtime ratio over a sequential GS algorithm as we increase the number of computing cores available; (2) scalability—the ability to handle a growing number of topics; and (3) accuracy—the held-out log-likelihood of LDA achieved by increasing number of iterations [Smola and Narayanamurthy 2010; Ahmed et al. 2012]. The baseline is the state-of-the-art industrial solution Yahoo!LDA [Smola and Narayanamurthy 2010; Ahmed et al. 2012] with open source codes.[1] Likewise, Yahoo!LDA also distributes SparseLDA [Yao et al. 2009] over multiple machines but using a shared memory environment based on the *memcached* technique. In layer 2 of

[1]https://github.com/sudar/Yahoo_LDA.

Peacock, we use one coordinator server and $M = 125$ aggregation servers (as shown in Figure 3). In layer 1, we set $C = 2$ configurations, each of which contains $M = 125$ sampling servers and $M + 1 = 126$ data servers.

### 4.1. Datasets

For a fair comparison, we use the same publicly available dataset PubMed,[2] which contains 8.2 million documents with an average length of around 90 word tokens each. The vocabulary size of PubMed is $1.4 \times 10^5$. We also compose the training corpus of search queries received in recent months called *SOSO*. The preprocessing of the corpus contains five steps. First, transform each query into word tokens, and count word frequencies. Second, remove those words with low frequency, which are likely typos. Third, remove those words with very high frequency, because common words tend to dominate all topics [Wallach et al. 2009a]. Fourth, de-duplicate queries: if a query appears multiple times, we keep only one appearance in corpus. This allows us to include a large variety of user intentions within a certain amount of training corpus. This also lowers the weight of frequent queries in the corpus. Fifth, remove those queries containing only one word, because single-word queries do not provide word co-occurrence counts, which is a clue used by LDA to infer topics. The processed corpus contains one billion search queries with 4.5 word tokens per query on average and takes 17.2GB of storage space. The vocabulary size of SOSO is around $2.1 \times 10^5$. Obviously, SOSO is about six times larger than PubMed.

### 4.2. Results

Figure 7 shows the speedup performance (fixing $K = 1{,}000$), where the $x$-axis is the number of cores and the $y$-axis is the runtime ratio of 100 cores over that of other number of cores in the $x$-axis. The variance is too small to be shown. We see that on average, Peacock achieves around 4.2 speedup when the number of cores is 1,000, which implies that the communication and synchronization in Peacock take about half of the training time. Yahoo!LDA has a much better speedup than Peacock when the number of cores is small ($\leq$1,000). However, its speedup performance drops significantly when the number of cores increases from 1,000 to 3,000. The possible reason is that Yahoo!LDA does not consider the lock-free synchronization problem. In practice, the very large number of cores will cause longer waiting time when accessing the shared memory based on the *memcached* technique. Peacock scales much better to the large number of cores by pipeline communication and lock-free synchronization (Sections 3.2.2 and 3.2.3). We see that Peacock is slower than Yahoo!LDA when the number of cores is small. The reason is that we use more separate sampling servers for model parallelism leading to the additional communication and synchronization costs, which remains at almost a constant ratio in training time by pipeline techniques.

Figure 7 also shows the training time per iteration with the increasing number of topics (fixing the number of cores to 500), $K = \{10^2, 10^3, 10^4, 10^5\}$. With $K$ increasing from $10^2$ to $10^4$, the corresponding training time increase of Peacock is small. When $K$ increases by 10 times from $10^4$ to $10^5$, the training time is close to the linear growth. The scalability of Peacock with respect to $K$ comes mainly from the model and data parallelism (Section 3.2), resulting in very fast sampling performance with a small memory footprint. From $K = 10^4$ to $K = 10^5$, Peacock consumes significantly more training time because the topic sparseness of each word token becomes lower in SparseLDA. As a comparison, Yahoo!LDA has an "out of memory" problem when the the number of topics is $K \geq 10^4$ because it does not consider storing a big topic-word count matrix $\Phi_{vk}$. Peacock divides this big parameter matrix into small model shards

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

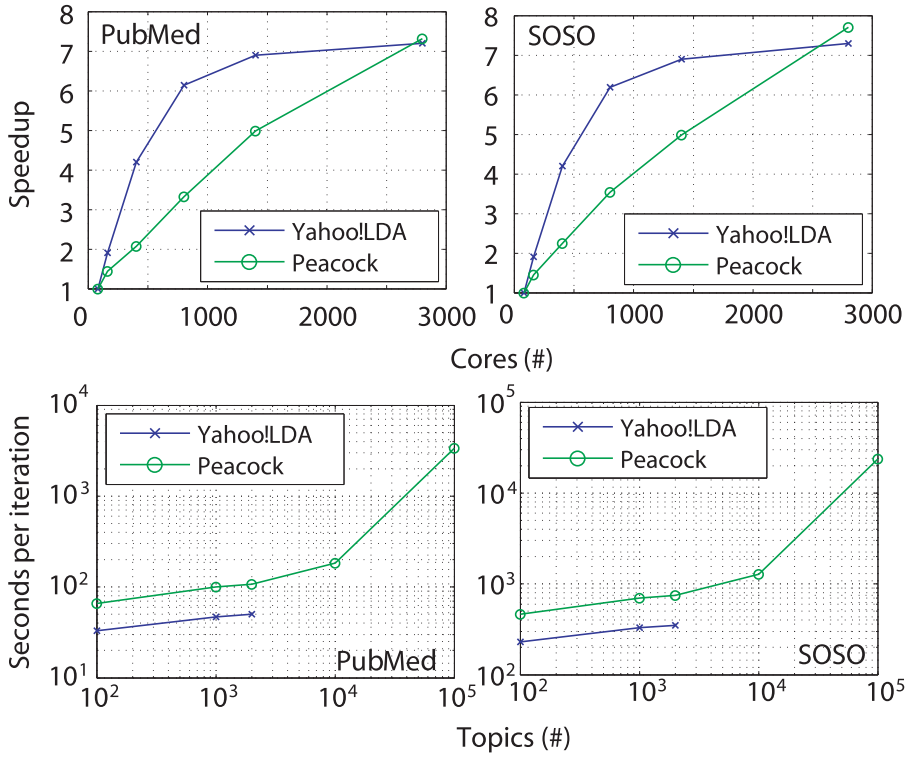


Fig. 7. The speedup and scalability performance.

to handle increasing number of topics. Since Peacock uses additional communication to synchronize more sampling servers, its per-iteration training time is slightly longer than that of Yahoo!LDA.

We use the iteration annealed importance sampling (iteration-AIS) [Wallach et al. 2009b; Foulds and Smyth 2014] method to evaluate the predictive performance (measured by the held-out log-likelihood) of Peacock. We randomly select $10^4$ documents from PubMed and $10^5$ queries from SOSO as the held-out datasets. Figure 8 shows the held-out log-likelihood as a function of training iterations and time (fixing $K = 1{,}000$ and the number of cores 500); variance is too small to be shown. The higher held-out log-likelihood corresponds to the better model quality. We observe that Peacock has a rise after 900 iterations because we start asymmetric prior optimization and topic de-duplication (Section 3.4), which can improve the topic model quality. Although both Peacock and Yahoo!LDA use SparseLDA, Peacock converges to a higher log-likelihood level. The reason is partly because Yahoo!LDA uses the approximate synchronization to speed up its performance, leading to a slightly worse model quality, which has been also observed in their own work [Smola and Narayanamurthy 2010; Ahmed et al. 2012]. Figure 8 also shows that Peacock uses less training time to achieve a higher held-out log-likelihood than Yahoo!LDA. To see if Peacock can produce the same model quality as that generated by a sequential GS inference on a single machine, we show their held-out log-likelihoods as a function of training iterations in Figure 9; variance is too small to be shown. Since the single machine cannot train the large number of samples due to the memory constraint, we randomly select a subset of the training set $10^6$ queries and select $10^4$ queries as the held-out set. We see that the held-out

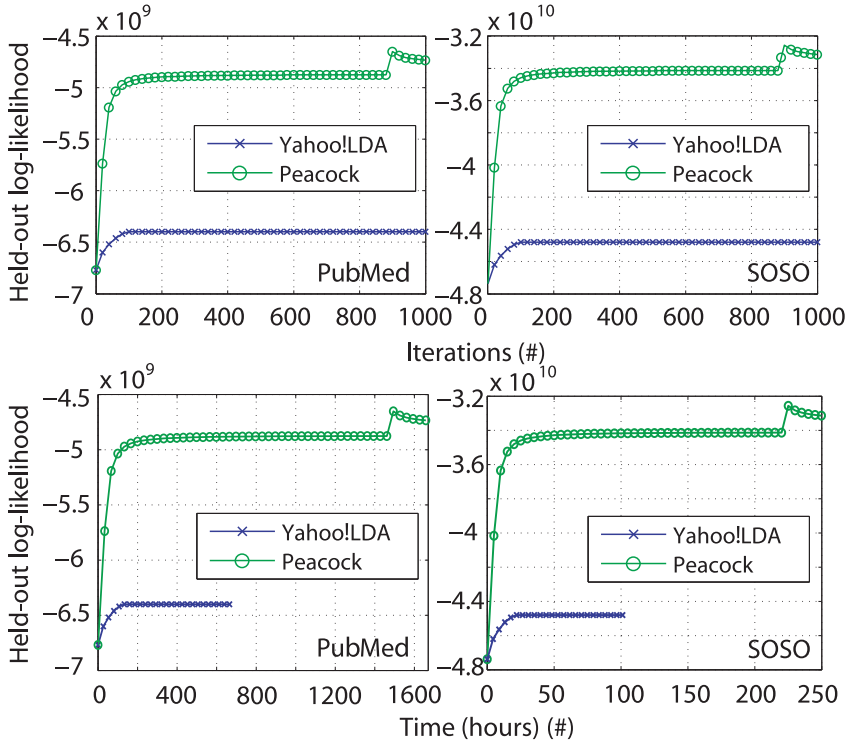


Fig. 8. The topic modeling accuracy and convergence speed.

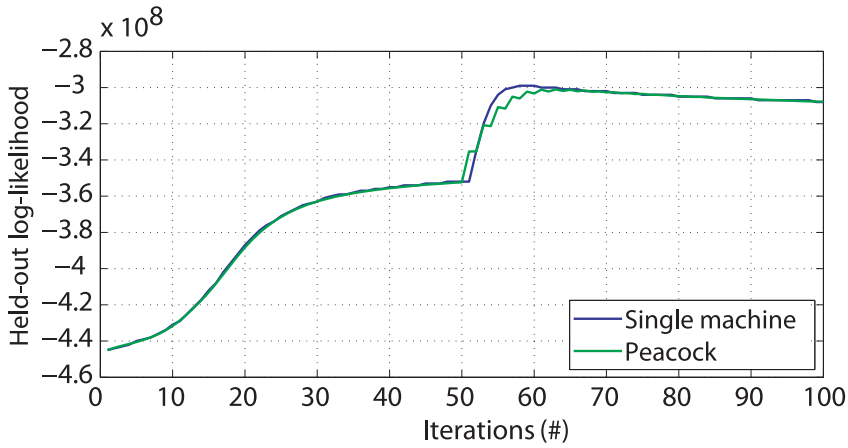


Fig. 9. Comparisons between Peacock and the single machine.

log-likelihood curve generated by Peacock locates closely to that produced by the sequential GS on the single machine. This result confirms that the approximate synchronization techniques used in Peacock do not affect the model quality very much.

To summarize, we see that Yahoo!LDA is more efficient for small-scale ($K \approx 10^3$) topic modeling tasks, whereas Peacock is more suitable for solving large-scale ($K \geq 10^5$) topic modeling problems in industrial applications.

## 5. ONLINE APPLICATIONS

After Peacock learns $K \geq 10^5$ topics from $D \geq 10^9$ queries, we need to integrate the topic features into existing search engine and online advertising systems. We extract topic features from new queries based on the topic distribution over words $\phi_{vk}$ in Equation (2) learned by Peacock. Given the word tokens of a new query $d$, we use RT-LDA to predict its topic distribution $\theta_{dk}$ by fixing $\phi_{vk}$. Employing the Bayes' rule, we calculate the likelihood $P(v|d)$ of a vocabulary word $v$ given a query $d$:

$$P(v|d) = \sum_{k=1}^{K} \phi_{vk}\theta_{dk}. \tag{8}$$

The $V$-length vector $P(v|d)$ is compatible with the standard word vector space model. If we rank $P(v|d)$ in descending order, we obtain the top-likely topic features in the input query.

### 5.1. Experimental Settings

The search engine uses the well-known vector space model in information retrieval and computes cosine similarity between queries and documents in their vector representations. We accelerate this process by using the Weak-AND algorithm [Broder et al. 2003]. Peacock replaces the word vector features of each query by the likely top 30 topic features in Equation (8) (top 30 largest values from $V$-length vector $P(v|d)$) inferred by RT-LDA in the head of each posting list used by the Weak-AND algorithm, which makes the query-document similarity computing efficient enough to be deployed in a real search engine.

Online advertising has been a fundamental financial support of the many free Internet services [Broder and Josifovski 2013]. Most contemporary online advertising systems follow the generalized second price (GSP) auction model [Edelman et al. 2007], which requires that the system is able to predict the click-through rate (pCTR) of an ad, where pCTR is an important clue in GSP to ranking ads and pricing clicks. One of the key questions with the pCTR is the availability of suitable input features or predictor variables that allow accurate CTR prediction for a given ad impression [Richardson et al. 2007]. These features can be grouped into three categories: 1) ad features including bid phrases, ad title, landing page; 2) a hierarchy of advertiser account, campaign, ad group; and 3) ad creative. User features include recent search queries and user behavior data. Context features include display location, geographic location, content of page under browsing, and time. Most of these features are text data in word vector space [Graepel et al. 2010]. We learn an $L_1$-regularized log-linear model [Andrew and Gao 2007] as the baseline for pCTR, which uses a set of text and other features such as ad title, content of page under browsing, content of landing page, ad group id, demographic information of users, categories of ad group, and categories of the page under browsing. As a comparison, Peacock adds all topic features (8)—that is, the $V$-length topic feature vector $P(v|d)$ in baseline text and other features as input to $L_1$-regularized log-linear model [Andrew and Gao 2007].

In both applications, the training dataset of Peacock is SOSO described in Section 4.1. For Peacock, we set $C = 2$ configurations with one coordinator server and $M = 125$ aggregation servers. Each configuration contains $M = 125$ sampling servers and $M + 1 = 126$ data servers.

### 5.2. Results

For information retrieval, our testbed is a real search engine, www.soso.com, which ranks the fourth largest in the China market. The test data is used for routinely relevance evaluation, containing 4,818 randomly selected queries and 121,588 query-URL

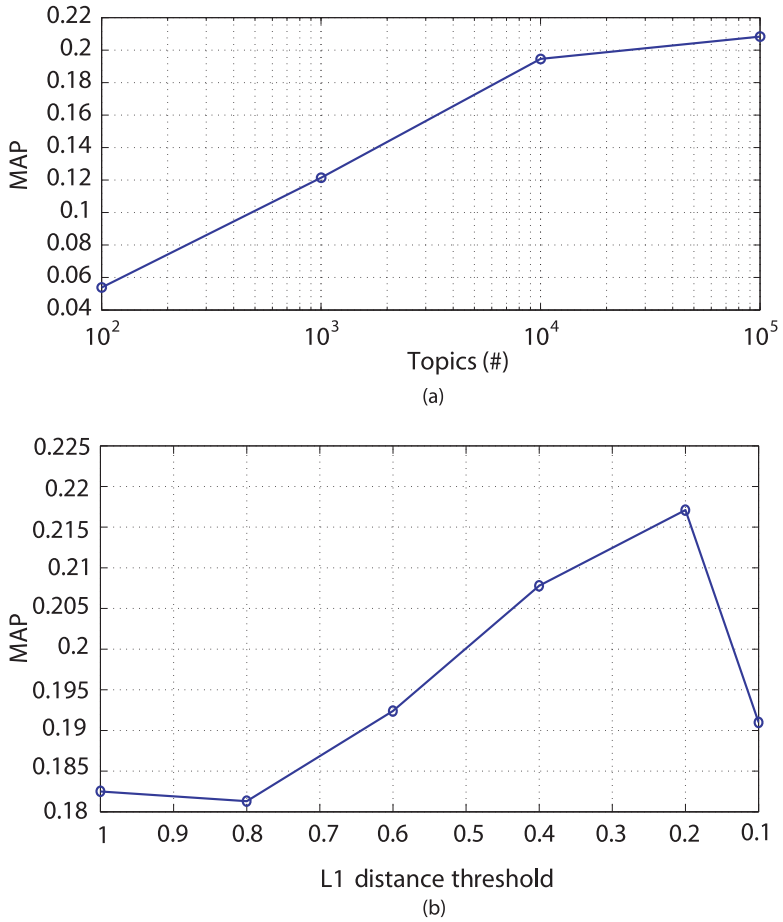


Fig. 10. (a) Topic features improve retrieval in the search engine. (b) Performance improvement in retrieval after topic de-duplication by topic clustering based on $L_1$ distance.

pairs with human labeled relevance rate. Every query-URL pair was rated by three human editors, and the average rate was taken. We compute mean average precision (MAP) using the TREC evaluation tool [Voorhees and Harman 2005]. A higher MAP means a better retrieval performance. Figure 10(a) shows that the topic features improve the relevance measure by MAP with small variance. The relevance improvement grows steadily with the increasing number of topics from $10^2$ to $10^5$. However, the growth of MAP becomes less salient when the number of topics changes from $10^4$ to $10^5$. This is mainly attributed to the problem of *topic duplication*. Figure 10(b) shows that the topic de-duplication method can further improve the relevance of information retrieval. The MAP value of retrieval grows when we prune more duplicated topics (lower $L_1$ distance can prune more similar topics in Section 3.4). Usually, the MAP stops increasing when we prune duplicates from the initial $10^6$ to around $10^5$ topics. This result implies that $10^5$ is a critical number of topics to describe subtle word senses in big query data with $2.1 \times 10^5$ vocabulary words. If the $L_1$ distance is too small (i.e., 0.1), it will degrade the MAP performance by removing more nonduplicate topics.

The online advertising experiment is conducted on a real contextual advertising system (http://e.qq.com). This system logs every ad shown to a particular user in a

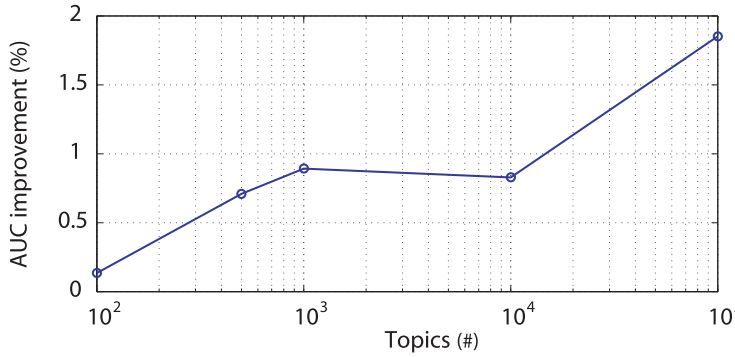


Fig. 11. Topic features improve the pCTR performance in online advertising systems.

particular page view as an *ad impression*. It also logs every click of an ad. By taking each impression as a training instance and labeling it by whether it was clicked, we obtain 9.9 billion training samples and 1.1 billion test samples. We train five *hypothetical models*, whose topic features are extracted using five different LDA models with $10^2$, $5 \times 10^2$, $10^3$, $10^4$, and $10^5$ topics, respectively. Following the judgment rule of Task 2 in KDD Cup 2012, a competition of ad pCTR, we compare our hypothetical models with the baseline by their prediction performance measured in area under the curve (AUC). Figure 11 shows that all hypothetical models gain relative AUC improvement (%) than the baseline (AUC = 0.7439). Variance is too small to be shown. This verifies the value of big LDA models. In addition, the AUC improvement grows with the increase of the number of topics learned by Peacock. The reason the performance of $10^4$ is lower than that of $10^3$ is because of many topic duplicates in $10^4$ topics. After using automatic topic de-duplication by asymmetric Dirchlet prior learning (Section 3.4), the performance of $10^5$ becomes better than that of $10^4$. This result is consistent with those in Figure 10.

## 6. CONCLUSIONS

Topic modeling techniques for big data are needed in many real-world applications. In this article, we confirm that a big LDA model with at least $10^5$ topics inferred from $10^9$ search queries can achieve a significant improvement in industrial applications such as search engine and online advertising systems. We propose a unified solution, Peacock, to do topic modeling for big data. Peacock uses a hierarchical distributed architecture to handle large-scale data as well as LDA parameters. In addition, Peacock addresses some novel problems in big topic modeling, including real-time prediction and topic de-duplication. We show that Peacock is scalable to more topics than the current state-of-the-art industrial solution Yahoo!LDA. Through two online applications, we also obtain the following experiences:

—The good performance is often achieved when the number of topics is approximately equal to or more than the number of vocabulary words. In our experiments, the vocabulary size is $2.1 \times 10^5$ so that the number of topics $K \geq 10^5$. In other industrial applications, the vocabulary size may reach a few millions or even a billion. The Peacock system can do topic feature learning when $K \geq 10^7$ is needed.
—The topic de-duplication method is a key technical component to ensure that $K \geq 10^5$ topics can provide high-quality topic features. Better topic de-duplication techniques remain to be an open research issue.

—The real-time topic prediction method for a large number of topics is also important in industrial applications. If $K \geq 10^7$, faster prediction methods are needed and remain to be a future research issue.

In our future work, we will study how to deploy online LDA algorithms in Peacock and how to implement inference algorithms to learn other topic models, such as HDP [Teh et al. 2004] and ATM [Steyvers et al. 2004].

## ACKNOWLEDGMENTS

This work was supported by National Grant Fundamental Research (973 Program) of China under Grant 2014CB340304, NSFC (Grant No. 61373092 and 61033013), Natural Science Foundation of the Jiangsu Higher Education Institutions of China (Grant No. 12KJA520004), and Innovative Research Team in Soochow University (Grant No. SDT2012B02). This work was partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization.

## REFERENCES

Sungjin Ahn, Babak Shahbaba, and Max Welling. 2014. Distributed stochastic gradient MCMC. In *Proceedings of ICML*. 1044–1052.

Amr Ahmed, Mohamed Aly, Joseph Gonzalez, Shravan M. Narayanamurthy, and Alexander J. Smola. 2012. Scalable inference in latent variable models. In *Proceedings of WSDM*. 123–132.

Galen Andrew and Jianfeng Gao. 2007. Scalable training of $L^1$-regularized log-linear models. In *Proceedings of ICML*. 33–40.

Arthur Asuncion, Max Welling, Padhraic Smyth, and Yee Whye Teh. 2009. On smoothing and inference for topic models. In *Proceedings of UAI*. 27–34.

Arthur U. Asuncion, Padhraic Smyth, and Max Welling. 2008. Asynchronous distributed learning of topic models. In *Proceedings of NIPS*. 81–88.

Petra Berenbrink, Tom Friedetzky, Zengjian Hu, and Russell Martin. 2008. On weighted balls-into-bins games. *Theoretical Computer Science* 409, 3, 511–520.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research* 3, 993–1022.

Andrei Broder and Vanja Josifovski. 2013. *Lecture Introduction to Computational Advertising*. Stanford University, Computer Science, Online Lecture Notes.

Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proceedings of CIKM*. 426–434.

Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C. Wilson, and Michael I. Jordan. 2013. Streaming variational Bayes. In *Proceedings of NIPS*. 1727–1735.

Wray L. Buntine and Aleks Jakulin. 2005. Discrete component analysis. In *Proceedings of SLSFS*. 1–33.

N. de Freitas and K. Barnard. 2001. *Bayesian Latent Semantic Analysis of Multimedia Databases*. Technical Report. University of British Columbia.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large scale distributed deep networks. In *Proceedings of NIPS*. 1232–1240.

Jeffery Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *Communications of the ACM* 51, 1, 107–113.

Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. 2007. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American Economic Review* 97, 1, 242–259.

James R. Foulds, Levi Boyles, Christopher DuBois, Padhraic Smyth, and Max Welling. 2013. Stochastic collapsed variational Bayesian inference for latent Dirichlet allocation. In *Proceedings of KDD*. 446–454.

James R. Foulds and Padhraic Smyth. 2014. Annealing paths for the evaluation of topic models. In *Proceedings of UAI*.

Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of KDD*. 69–77.

Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. 2010. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In *Proceedings of ICML*. 13–20.

David Graff and Christopher Cieri. 2003. English Gigaword. Linguistic Data Consortium.

Thomas Griffiths and Mark Steyvers. 2004. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America* 101, 5228–5235.

Matthew D. Hoffman, David M. Blei, and Francis R. Bach. 2010. Online learning for latent Dirichlet allocation. In *Proceedings of NIPS*. 856–864.

Matthew D. Hoffman, David M. Blei, Chong Wang, and John William Paisley. 2013. Stochastic variational inference. *Journal of Machine Learning Research* 14, 1, 1303–1347.

Matthew Johnson, James Saunderson, and Alan Willsky. 2013. Analyzing hogwild parallel Gaussian Gibbs sampling. In *Proceedings of NIPS*.

Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. 2008. PFP: Parallel FP-growth for query recommendation. In *Proceedings of RecSys*. 107–114.

Zhiyuan Liu, Yuzhou Zhang, Edward Y. Chang, and Maosong Sun. 2011. PLDA+: Parallel latent Dirichlet allocation with data placement and pipeline processing. *ACM Transactions on Intelligent Systems and Technology* 2, 3, 26.

David M. Mimno, Matthew D. Hoffman, and David M. Blei. 2012. Sparse stochastic inference for latent Dirichlet allocation. In *Proceedings of ICML*.

Thomas P. Minka and John D. Lafferty. 2002. Expectation-propogation for the generative aspect model. In *Proceedings of UAI*. 352–359.

Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA.

David Newman, Arthur U. Asuncion, Padhraic Smyth, and Max Welling. 2007. Distributed inference for latent Dirichlet allocation. In *Proceedings of NIPS*.

David Newman, Jey Han Lau, Karl Grieser, and Timothy Baldwin. 2010. Automatic evaluation of topic coherence. In *Proceedings of HLT-NAACL*. 100–108.

Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. 2011. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of NIPS*. 693–701.

Sam Patterson and Yee Whye Teh. 2013. Stochastic gradient Riemannian Langevin dynamics on the probability simplex. In *Proceedings of NIPS*. 3102–3110.

Ian Porteous, David Newman, Alexander T. Ihler, Arthur U. Asuncion, Padhraic Smyth, and Max Welling. 2008. Fast collapsed Gibbs sampling for latent Dirichlet allocation. In *Proceedings of KDD*. 569–577.

Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks: Estimating the click-through rate for new ads. In *Proceedings of WWW*. 521–530.

Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *Annals of Mathematical Statistics* 22, 3, 400–407.

Issei Sato and Hiroshi Nakagawa. 2012. Rethinking collapsed variational Bayes inference for LDA. In *Proceedings of ICML*.

Mark W. Schmidt, Nicolas Le Roux, and Francis Bach. 2013. Minimizing finite sums with the stochastic average gradient. *CoRR* abs/1309.2388.

Alexander J. Smola and Shravan M. Narayanamurthy. 2010. An architecture for parallel topic models. *Proceedings of the VLDB Endowment* 3, 1, 703–710.

Mark Steyvers, Padhraic Smyth, Michal Rosen-Zvi, and Thomas L. Griffiths. 2004. Probabilistic author-topic models for information discovery. In *Proceedings of KDD*. 306–315.

Yee Whye Teh, Michael Jordan, Matthew Beal, and David Blei. 2004. Hierarchical Dirichlet processes. *Journal of the American Statistical Association* 101, 476, 1566–1581.

Yee Whye Teh, David Newman, and Max Welling. 2006. A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation. In *Proceedings of NIPS*. 1353–1360.

Ellen M. Voorhees and Donna K. Harman (Eds.). 2005. *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, Cambridge MA.

Hanna M. Wallach, David M. Mimno, and Andrew McCallum. 2009a. Rethinking LDA: Why priors matter. In *Proceedings of NIPS*. 1973–1981.

Hanna M. Wallach, Iain Murray, Ruslan Salakhutdinov, and David M. Mimno. 2009b. Evaluation methods for topic models. In *Proceedings of ICML*. 1105–1112.

Yi Wang, Hongjie Bai, Matt Stanton, Wen-Yen Chen, and Edward Y. Chang. 2009. PLDA: Parallel latent Dirichlet allocation for large-scale applications. In *Proceedings of AAIM*. 301–314.

Feng Yan, Ningyi Xu, and Yuan Qi. 2009. Parallel inference for latent Dirichlet allocation on graphics processing units. In *Proceedings of NIPS*. 2134–2142.

Jian-Feng Yan, Jia Zeng, Yang Gao, and Zhi-Qiang Liu. 2014. Communication-efficient algorithms for parallel latent Dirichlet allocation. *Soft Computing* 19, 1, 3–11.

Jian-Feng Yan, Jia Zeng, Zhi-Qiang Liu, and Yang Gao. 2013. Towards big topic modeling. arXiv:1311.4150.

Limin Yao, David M. Mimno, and Andrew McCallum. 2009. Efficient methods for topic model inference on streaming document collections. In *Proceedings of KDD*. 937–946.

Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. 2010. A comparison of optimization methods and software for large-scale $L^1$-regularized linear classification. *Journal of Machine Learning Research* 11, 3183–3234.

Jia Zeng. 2012. A topic modeling toolbox using belief propagation. *Journal of Machine Learning Research* 13, 2233–2236.

Jia Zeng, William K. Cheung, and Jiming Liu. 2013. Learning topic models by belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 5, 1121–1134.

Jia Zeng, Zhi-Qiang Liu, and Xiao-Qin Cao. 2012a. A new approach to speeding up topic modeling. arXiv:1204.0170 [cs.LG].

Jia Zeng, Zhi-Qiang Liu, and Xiao-Qin Cao. 2012b. Online belief propagation for topic modeling. arXiv:1210.2179 [cs.LG].

Ke Zhai, Jordan L. Boyd-Graber, Nima Asadi, and Mohamad L. Alkhouja. 2012. Mr. LDA: A flexible large scale topic modeling package using variational inference in MapReduce. In *Proceedings of WWW*. 879–888.

Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. 2013. A fast parallel SGD for matrix factorization in shared memory systems. In *Proceedings of RecSys*. 249–256.

Received May 2014; revised October 2014; accepted December 2014