Asynchronous Distributed Estimation of Topic Models for Document Analysis

Arthur Asuncion, Padhraic Smyth, Max Welling

Department of Computer Science

University of California, Irvine

Abstract

Given the prevalence of large data sets and the availability of inexpensive parallel computing hardware, there is significant motivation to explore distributed implementations of statistical learning algorithms. In this paper, we present a distributed learning framework for Latent Dirichlet Allocation (LDA), a well-known Bayesian latent variable model for sparse matrices of count data. In the proposed approach, data are distributed across P processors, and processors independently perform inference on their local data and communicate their sufficient statistics in a local asynchronous manner with other processors. We apply two different approximate inference techniques for LDA, collapsed Gibbs sampling and collapsed variational inference, within a distributed framework. The results show significant improvements in computation time and memory when running the algorithms on very large text corpora using parallel hardware. Despite the approximate nature of the proposed approach, simulations suggest that asynchronous distributed algorithms are able to learn models that are nearly as accurate as those learned by the standard non-distributed approaches. We also find that our distributed algorithms converge rapidly to good solutions.

Key words: Topic Model, Distributed Learning, Parallelization, Gibbs sampling

1. Introduction

The emergence of the Internet over the past decade has significantly increased the amount of information available to end users. For instance, the Medline database contains millions of scientific publications, the Netflix Prize data set has over 100 million movie ratings, and Google indexes over a billion web pages. Statistical analysis with complex models on data sets of this scale is often difficult on a single computer—the data may not fit in main memory or the time to perform the analysis may take on the order of days or weeks.

Fortunately, parallel and distributed computing resources are becoming relatively inexpensive and widely available. New computers are typically equipped with multi-core processors, and clusters of computers can be deployed with relative ease. The increasing availability of multiprocessor and grid computing technologies provides a practical motivation to develop statistical learning algorithms that are able take advantage of such computational resources.

In this paper, we focus on the specific problem of developing distributed learning algorithms for the Latent Dirichlet Allocation model [1]. LDA is a Bayesian model for sparse high dimensional matrices of counts, such as word frequencies in text documents or feature counts in images.

In the machine learning community, LDA is widely referred to as the "topic model" since this model facilitates the learning of low dimensional representations, or "topics", from the data. LDA has been primarily applied to text corpora for the purpose of performing automated document analysis and multiple variants have already been proposed. Hierarchical Dirichlet Processes (HDP) are a non-parametric analogue of LDA which allow the number of topics to vary [2]. The Author-Topic model builds upon LDA by incorporating the notion of an author [3]. Models that seek to learn correlations between topics include the Correlated Topic Model [4] and Pachinko Allocation [5]. Applications of LDA include information retrieval [6], entity resolution [7], web spam filtering [8], software artifact analysis [9], and computer vision tasks [10].

There is growing interest in applying these techniques to very large data sets. In order to scale LDA to these large data sets, several distributed topic modeling algorithms have recently been developed [11, 12, 13, 14]. Distributed computation in this context provides two distinct benefits: (1) parallelization across many machines can significantly speed up inference; (2) distributed computing increases the total amount of collective memory, allowing corpora with billions of words to be processed efficiently.

While synchronous distributed algorithms for topic models have been proposed in earlier work, our primary contribution is the introduction of *asynchronous* distributed algorithms for LDA, based on collapsed Gibbs sampling and collapsed variational inference. Fully asynchronous algorithms provide several computational advantages over their synchronous counterparts: (1) there does not exist the computational bottleneck of global synchronization across all processors; (2) the system is fault-tolerant due to its decentralized nature; (3) heterogeneous machines with different processor speeds and memory capacities can be used; (4) new processors and new data can be incorporated into the system at any time.

We employ an asynchronous "gossip-based" framework [15] which only uses pairwise interactions between random pairs of processors. The distributed framework we propose can provide substantial memory and time savings over single-processor computation, since each processor only needs to store and perform Gibbs sampling sweeps over $\frac{1}{P}$th of the data, where P is the number of processors. Furthermore, the asynchronous approach can scale to large corpora and large numbers of processors, since no global synchronization steps are required.

In the proposed framework, local inference on individual processors is based on a noisy inexact view of the global topics. As a result, our distributed collapsed sampling algorithm is not sampling from the proper global posterior distribution. Likewise, the distributed collapsed variational inference algorithm we propose is not optimizing the true variational bound. Nonetheless, as we will show in our experiments, these algorithms are empirically very robust and converge rapidly to high-quality solutions. In most applications of LDA, one is often most interested in discovering good modes in the posterior rather than fully analyzing the posterior shape. We find that our algorithms are particularly well-suited for this task.

We first review collapsed Gibbs sampling and collapsed variational inference for LDA. We also briefly review the general use of distributed computing in statistics and machine learning. Then we describe the details of our distributed algorithms. Finally, we present accuracy, convergence, and speedup results for our algorithms when applied to text data and conclude with directions for future work.

2. A Review of Latent Dirichlet Allocation

Dimensionality reduction and the discovery of latent relationships between variables are important problems which have prompted the development of statistical decomposition techniques

such as factor analysis and related approaches. A well-known dimensionality reduction technique is Principal Components Analysis (PCA), which allows for the extraction of principal components from data through the eigenvalue decomposition of the data covariance matrix [16]. Latent Semantic Analysis (LSA) can be viewed as the application of PCA to documents [17]. When applying LSA to text corpora, each document is represented as a vector of frequencies of word counts for the document. The ensuing matrix of word-document counts is decomposed via singular value decomposition, allowing documents to be mapped to a lower dimensional space.

Probabilistic Latent Semantic Analysis (PLSA) improves upon LSA by introducing a probabilistic model for this decomposition [18]. In turn, Latent Dirichlet Allocation (LDA) was proposed as a generalization of PLSA, casting the model within a generative Bayesian framework, and in the process avoiding some of the overfitting issues that were observed with PLSA [1]. LDA also bears similarities to other statistical models, such as admixture models [19] and mixed-membership models [20]. A general review of the similarities between PCA, LSA, PLSA, LDA, and other models can be found in Buntine and Jakulin [21].

In LDA, each document j in the corpus is modeled as a mixture over K topics, and each topic k is a discrete distribution, ϕ_{wk}, over a vocabulary of W words[1]. Each topic, ϕ_{wk}, is drawn from a Dirichlet distribution with parameter η. In order to generate a new document, the document's mixture, θ_{kj}, is first sampled from a Dirichlet distribution with parameter α. For each token i in that document, a topic assignment z_{ij} is sampled from θ_{kj}, and the specific word x_{ij} is drawn from $\phi_{wz_{ij}}$. The graphical model for LDA is shown in Figure 1, and the generative process is below:

$$\theta_{k,j} \sim D[\alpha] \qquad \phi_{w,k} \sim D[\eta] \qquad z_{ij} \sim \theta_{k,j} \qquad x_{ij} \sim \phi_{w,z_{ij}} \; .$$

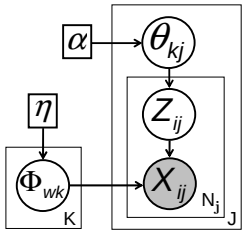

Figure 1: LDA model. Plates denote replication over indices, boxes denote parameters, circles denote hidden variables, shaded circles denote observed variables, and arrows denote dependencies.

Given observed data, it is possible to infer the posterior distribution of the latent variables. For LDA, a particularly simple and accurate inference technique is collapsed Gibbs sampling (CGS) [22], in which θ_{kj} and ϕ_{wk} are integrated out and sampling of the topic assignments is performed sequentially in the following manner,

$$P(z_{ij} = k | z^{\neg ij}, x_{ij} = w) \propto \frac{N_{wk}^{\neg ij} + \eta}{\sum_w N_{wk}^{\neg ij} + W\eta} \left(N_{jk}^{\neg ij} + \alpha \right) . \qquad (1)$$

N_{wk} denotes the number of word tokens of type w assigned to topic k, while N_{jk} is the number of tokens in document j assigned to topic k. $N^{\neg ij}$ denotes the count with token ij removed.

Once an LDA model is learned, the topics, ϕ, can be used in a variety of ways. The high probability words of each topic are informative of the semantic nature of the topic. For instance, if the high probability words in topic 1 are "Student School Study Grades Teacher", we would interpret topic 1 to be about academics. Examples of actual LDA topics learned on the NIPS corpus are shown later in the paper (Table 3). Moreover, each document's θ gives us the distribution of topics within that document. Thus, LDA provides an automatic way to summarize the semantic content of a corpus (through topics) and discover the topical content within each document.

[1]To avoid notational clutter, we write ϕ_{wk} or θ_{kj} to denote the set of all components, i.e. $\{\phi_{wk}\}$ or $\{\theta_{kj}\}$. Similarly, when sampling from a Dirichlet distribution, we write $\theta_{kj} \sim D[\alpha]$ instead of $[\theta_{1,j}, ..\theta_{K,j}] \sim D[\alpha, .., \alpha]$. We use symmetric Dirichlet priors for simplicity in this paper.

Furthermore, similar documents can be clustered together based on the similarity between their θ's, allowing for applications such as document retrieval and classification.

An alternative approximate inference technique for LDA is variational Bayesian inference (VB) [1], where the true posterior distribution is approximated by a fully factorized posterior distribution in order to simplify inference. In VB, the negative marginal log likelihood is upper bounded by Jensen's inequality, leading to a quantity known as the variational free energy. The objective is to minimize this free energy with respect to the variational parameters. As with the collapsed Gibbs sampler, θ_{kj} and ϕ_{wk} can be integrated out within the variational inference framework, yielding collapsed variational Bayesian (CVB) inference [23]. In CVB, each token ij has an underlying discrete distribution over topics $\{\gamma_{ij1}...\gamma_{ijK}\}$, and the product of these distributions forms the fully factorized posterior. Optimizing with respect to these variational parameters yields a relatively efficient deterministic update,

$$\gamma_{ijk} \propto \frac{N_{wk}^{\neg ij} + \eta}{\sum_w N_{wk}^{\neg ij} + W\eta} \left(N_{jk}^{\neg ij} + \alpha\right) \exp\left(-\frac{V_{jk}^{\neg ij}}{2(N_{jk}^{\neg ij} + \alpha)^2} - \frac{V_{wk}^{\neg ij}}{2(N_{wk}^{\neg ij} + \eta)^2} + \frac{\sum_w V_{wk}^{\neg ij}}{2(\sum_w N_{wk}^{\neg ij} + W\eta)^2}\right). \tag{2}$$

In CVB, N_{jk} denotes the expected number of tokens in document j assigned to topic k, and can be calculated from γ as follows: $N_{jk}^{\neg ij} = \sum_{i' \neq i} \gamma_{i'jk}$. There is also a variance associated with each count: $V_{jk}^{\neg ij} = \sum_{i' \neq i} \gamma_{i'jk}(1 - \gamma_{i'jk})$. This CVB update is fully derived in Teh et al. [23]. The distributed CVB algorithm that we introduce in this paper utilizes a hybrid technique which performs CGS on singleton tokens (where the count in the word-document matrix is one) and CVB updates on non-singleton tokens [24].

3. Related work on distributed learning

The use of parallel and distributed computing in statistics and machine learning has garnered significant interest in recent years. Kontoghiorghes has compiled a lengthy handbook of the use of parallel computing in statistics, including parallel optimization methods, parallel algorithms for linear algebra techniques, and parallel Bayesian computation [25]. Distributed expectation maximization (EM) algorithms have been investigated in both the synchronous case [26] and asynchronous case [27]. Rossini et al. propose a parallel programming framework in the R language for *embarrassingly parallel* problems, i.e. problems which can be decomposed into independent subtasks [28]. Chu et al. recast various machine learning algorithms, such as logistic regression, PCA, EM, and SVD, within Google's distributed Map-Reduce framework [29]

Parallelized Gibbs sampling techniques for the specific case of LDA models have been proposed by Newman et al. [13] and Mimno and McCallum [11]. In other work, Nallapati et al. [12] parallelize the variational expectation maximization algorithm for LDA by taking advantage of the independence of variational parameters between documents. Similarly, Wolfe et al. [14] parallelize both the E and M steps of the variational LDA algorithm, under various computer network topologies. In a similar vein, other related models, such as PLSA, have also been parallelized [30]. The primary distinctions between the ideas in this paper and earlier work are that (a) our algorithms use purely asynchronous communication rather than a global synchronous scheme, and (b) we have incorporated both collapsed Gibbs sampling and collapsed variational inference within our framework.

Figure 2: Sampling two variables in parallel, conditioned on the opposing value in the previous time step, produces two independent chains and incorrect results [34].

More generally, Gibbs sampling can be parallelized in relatively simple ways: (1) samplers with different initial seeds can be run in parallel to obtain multiple samples, (2) non-interacting sets of variables can be sampled in parallel [25]. In this paper, we are more concerned with the problem of parallelizing the generation of a single sample, since many applications of LDA only require the discovery of a good posterior mode, e.g., the discovery and interpretation of specific word-topic probability distributions. If variables are dependent, parallelizing within a single run of Gibbs sampling is difficult to perform due to the sequential nature of MCMC. Consider Gibbs sampling with two dependent variables, as shown in Figure 2. If both variables are sampled concurrently given the values of the opposing variables at the previous iteration, two independent Markov chains would form, and the sample (x^t, y^t) for any time t would never be simulated from the correct distribution. Approximate parallel sampling for LDA makes use of the same concurrent sampling technique [13]. However, in LDA, because there can be millions of variables (the latent topic assignments for the words), and these variables tend to be only weakly dependent on each other, it is reasonable to believe that local sampling will be approximately correct. Empirical studies have supported this intuition, by showing that approximations resulting from performing Gibbs sampling on LDA models in parallel appear to be very slight [13, 31]. Furthermore, it is possible to derive a bound for the approximation error for certain parallel LDA samplers [32].

Exact parallel Gibbs samplers have been shown to exist by making use of periodic synchronous random fields [33]; however, a method of construction for such a sampler is not available. Brockwell [34] presents a pre-fetching parallel algorithm for MCMC, and Winkler [35] describes general parallel sampling schemes which rely on an annealing schedule. These techniques appear to be impractical for large-scale sampling for LDA. Newman et al. [13] modify the LDA model slightly to obtain a model in which collapsed inference is exactly parallelizable. This particular technique does not seem to be applicable to the asynchronous distributed setup. Other related sampling techniques are parallel particle filtering [36], and parallel tempering [37].

There also exists a large body of prior work on gossip algorithms (e.g., [15]), such as Newscast EM, a gossip algorithm for performing EM on Gaussian mixture models [27]. The techniques we present in this paper are also related to work on distributed averaging (e.g. [38]), where each node has an initial value, and the goal is to calculate the average of these values across the network of nodes in a distributed fashion. In our case, each node (or processor) has a matrix of LDA word-topic counts, and the goal of each processor is to estimate the sum of these matrices across all nodes in the network (in order to obtain the global topic information needed for Gibbs sampling), with the added complication that these counts are dynamically changing on each processor, due to Gibbs sampling.

4. Asynchronous distributed learning algorithms for LDA

We consider the task of learning an LDA model with K topics in a distributed fashion where J documents are distributed across P processors. Each processor p stores the following local variables: w_{ij}^p contains the word type for each token i in document j in the processor, and z_{ij}^p contains the assigned topic for each token. $N_{wk}^{\neg p}$ is the global word-topic count matrix stored at the processor—this matrix stores counts of other processors gathered during the communication step and does not include the processor's local counts. N_{kj}^p is the local document-topic count matrix (derived from z^p), N_w^p is the simple word count on a processor (derived from w^p), and N_{wk}^p is the local word-topic count matrix (derived from z^p and w^p) which only contains the counts of data on the processor. In the CVB case, instead of storing a scalar topic assignment z_{ij}^p for each token, a distribution over topics is stored as $\{\gamma_{pij1}...\gamma_{pijK}\}$, and variance counts ($V_{wk}^{\neg p}$, V_{wk}^p and V_{kj}^p) are also stored. In the following sections, we describe the details of our distributed algorithms.

4.1. Async-CGS: Asynchronous collapsed Gibbs sampling

We begin by describing a synchronous parallel version of LDA based on collapsed Gibbs sampling (which we call Parallel-CGS) introduced by Newman et al. [13]. In Parallel-CGS, each processor has responsibility for $\frac{1}{P}$ of the documents in the corpus, and the z's are globally initialized. Each iteration of the algorithm is composed of a Gibbs sampling step followed by a synchronization step. In the sampling step, each processor samples its local z^p by using the global topics of the previous iteration. In the synchronization step, the local counts N_{wk}^p on each processor are aggregated to produce a global set of word-topic counts N_{wk}. This process is repeated for a fixed number of iterations or until a convergence criterion is satisfied.

While Parallel-CGS provides substantial memory and time savings over the standard single processor algorithm, it is a fully synchronous algorithm which requires global synchronization at each iteration. In some applications, a global synchronization step may not be desirable or feasible. Some processors may be unavailable, while other processors may be in the middle of a long Gibbs sweep, due to differences in processor speeds or different amounts of data on each processor. To obtain the benefits of asynchronous computing, we introduce an asynchronous distributed version of LDA based on collapsed Gibbs sampling (Async-CGS) that follows a similar two-step process to that above. Each processor performs a local collapsed Gibbs sampling step followed by a step of communicating with another randomly selected processor.

In each iteration of Async-CGS, the processors perform a full sweep of collapsed Gibbs sampling over their local topic assignment variables z^p according to the following conditional distribution, in a manner directly analogous to equation 1,

$$P(z_{pij} = k | z_p^{\neg ij}, w_p) \propto \frac{(N^{\neg p} + N^p)_{wk}^{\neg ij} + \eta}{\sum_w (N^{\neg p} + N^p)_{wk}^{\neg ij} + W\eta} \left(N_{pjk}^{\neg ij} + \alpha\right). \tag{3}$$

The sum of $N_{wk}^{\neg p}$ and N_{wk}^p is used in the sampling equation. Recall that $N_{wk}^{\neg p}$ represents processor p's belief of the counts of all the other processors with which it has already communicated (not including processor p's local counts), while N_{wk}^p is the processor's local word-topic counts. Thus, the sampling of the z^p's is based on the processor's "noisy view" of the global set of topics.

Once the inference of z^p is complete (and N_{wk}^p is updated), the processor finds another finished processor and initiates communication[2]. We are generally interested in the case where memory

[2]We do not discuss in general the details of how processors might identify other processors that have finished their iteration, but we imagine that a standard network protocol could be used, like peer-to-peer.

Algorithm 1 Async-CGS

for each processor p in parallel **do**
  repeat
    Sample z^p locally (Equation 3)
    Receive N^g_{wk} from random proc g
    Send N^p_{wk} to proc g
    if p has met g before **then**
      Sample $\tilde{N}^g_{wk}$ (Equation 4)
      $N^{\neg p}_{wk} \leftarrow N^{\neg p}_{wk} - \tilde{N}^g_{wk} + N^g_{wk}$
    else
      $N^{\neg p}_{wk} \leftarrow N^{\neg p}_{wk} + N^g_{wk}$
    end if
  until convergence
end for

Algorithm 2 Async-CVB

for each processor p in parallel **do**
  repeat
    Update γ^p locally (Equations 3, 5)
    Receive N^g_{wk}, V^g_{wk} from random proc g
    Send N^p_{wk}, V^p_{wk} to proc g
    if p has met g before **then**
      Calculate $\tilde{N}^g_{wk}$, $\tilde{V}^g_{wk}$ (Equation 6)
      $N^{\neg p}_{wk} \leftarrow N^{\neg p}_{wk} - \tilde{N}^g_{wk} + N^g_{wk}$
      $V^{\neg p}_{wk} \leftarrow V^{\neg p}_{wk} - \tilde{V}^g_{wk} + V^g_{wk}$
    else
      $N^{\neg p}_{wk} \leftarrow N^{\neg p}_{wk} + N^g_{wk}$
      $V^{\neg p}_{wk} \leftarrow V^{\neg p}_{wk} + V^g_{wk}$
    end if
  until convergence
end for

and communication bandwidth are both limited. We also assume in the simplified gossip scheme that a processor can establish communication with every other processor. In the more general case these assumptions can be relaxed.

In the communication step, if two processors p and g have never met before, the processors would simply exchange their local N^p_{wk}'s (their local contribution to the global topic set), and processor p would add N^g_{wk} to its $N^{\neg p}_{wk}$, and vice versa.

When the two processors meet again, the synchronization is a little more complex. The processors should not simply swap and add their local counts again; rather, each processor should first remove from $N^{\neg p}_{wk}$ the previous influence of the other processor during their previous encounter, in order to prevent processors that frequently meet from over-influencing each other. In the general case, we can assume that processor p does not store in memory the previous counts of all the other processors that processor p has already met. Since the previous local counts of the other processor were already absorbed into $N^{\neg p}_{wk}$ and are thus not retrievable, we must take a different approach. In Async-CGS, the processors exchange their N^p_{wk}'s, from which the count of words on each processor, N^p_w can be derived. Using processor g's N^g_w, processor p creates a proxy set of counts, $\tilde{N}^g_{wk}$, by sampling N^g_w topic values randomly without replacement from collection $\{N^{\neg p}_{wk}\}$. We can imagine that there are $\sum_k N^{\neg p}_{wk}$ colored balls, with $N^{\neg p}_{wk}$ balls of color k, from which we pick N^g_w balls uniformly at random without replacement. This process is equivalent to sampling from a multivariate hypergeometric (MH) distribution. $\tilde{N}^g_{wk}$ acts as a substitute for the N^g_{wk} that processor p received during their previous encounter. Since all knowledge of the previous N^g_{wk} is lost, this method makes use of Laplace's principle of indifference (or the principle of maximum entropy). Finally, we update $N^{\neg p}_{wk}$ by subtracting $\tilde{N}^g_{wk}$ and adding the current N^g_{wk}:

$$N^{\neg p}_{wk} \leftarrow N^{\neg p}_{wk} - \tilde{N}^g_{wk} + N^g_{wk} \quad \text{where} \quad \tilde{N}^g_{w,k} \sim \text{MH}\left[N^g_w; N^{\neg p}_{w,1}, .., N^{\neg p}_{w,K}\right]. \tag{4}$$

Pseudocode for Async-CGS is shown in the display box for Algorithm 1. The assumption of limited memory can be relaxed by allowing processors to cache previous counts of other processors. In this case, the cached N^g_{wk} would replace $\tilde{N}^g_{wk}$.

A limitation of our merging scheme is that processors would need to visit all other processors to obtain all the counts in the system. In simple gossip-based problems, such as computing the global average of numbers stored on different processors, pairs of processors would exchange their numbers and then compute and store the average of those numbers. This averaging procedure allows information to quickly propagate through the network. However, the concept of

	Time (per iteration)	Space	Communication
Async-CGS	$\frac{1}{P}(NK)$	$2WK + \frac{1}{P}(N + JK)$	WK
Async-CVB	$\frac{1}{P}(MK)$	$4WK + \frac{1}{P}(MK + 2JK)$	$2WK$

Table 1: Time, space, and communication complexities for Async-CGS and Async-CVB.

averaging counts does not appear to work well in the Async-CGS case. We have investigated several asynchronous algorithms which average the global topic counts between two processors and we found that averaging performs worse than the Async-CGS algorithm we have described.

One way to overcome this limitation is to relax the assumption of limited bandwidth. Processor p could forward its individual cached counts (from other processors) to g, and vice versa, to quicken the dissemination of information. In fixed topologies where the network is not fully connected, forwarding is necessary to propagate the counts across the network.

4.2. Async-CVB: Asynchronous collapsed variational Bayesian inference

As an alternative to collapsed Gibbs sampling, we now consider an asynchronous distributed variational algorithm for LDA, which we will refer to as Async-CVB. The general scheme is essentially the same as Async-CGS, except that collapsed variational updates are performed, in a manner directly analogous to equation 2,

$$\gamma_{pijk} \propto \frac{(N^{\neg p} + N^{p})^{\neg ij}_{wk} + \eta}{\sum_w (N^{\neg p} + N^{p})^{\neg ij}_{wk} + W\eta} \left(N^{\neg ij}_{pjk} + \alpha\right) \exp\left(-\frac{V^{\neg ij}_{pjk}}{2(N^{\neg ij}_{pjk} + \alpha)^2} - \frac{(V^{\neg p} + V^{p})^{\neg ij}_{wk}}{2((N^{\neg p} + N^{p})^{\neg ij}_{wk} + \eta)^2} + \frac{\sum_w (V^{\neg p} + V^{p})^{\neg ij}_{wk}}{2(\sum_w (N^{\neg p} + N^{p})^{\neg ij}_{wk} + W\eta)^2}\right). \quad (5)$$

Pseudocode for Async-CVB is found in the display box for Algorithm 2. Just like Async-CGS, the combination of global and local topic counts is used in the update equation. The corresponding global and local variance counts are also used in the equation. Recall that Async-CVB makes use of the improved hybrid technique [24], which performs CGS on singleton tokens (eq. 3) and CVB updates on non-singleton tokens (eq. 5) . When performing CGS for a singleton token, the sampled topic assignment z^p_{ij} can be represented as $\{\gamma_{pij1} ... \gamma_{pijK}\}$ where all the probabilities are zero except for $\gamma_{pijz^p_{ij}}$ which is set to one. Thus, both the CGS and CVB updates yield a vector of variational parameters, and computation for the expected counts remains the same.

One drawback of Async-CVB is that it oftentimes requires more memory than Async-CGS, since Async-CVB needs to store the variance count matrices as well as a variational distribution for each token. Async-CVB also needs to transmit twice the amount of information as Async-CGS, in order to maintain the global variance counts. If memory is limited or network communication costs are expensive, Async-CGS would be preferable to Async-CVB.

Async-CVB does provide several benefits. Unlike MCMC techniques, variational techniques are able to assess convergence by monitoring the change in the variational free energy. Furthermore, instead of sampling $\tilde{N}$ from the MH distribution, the expected value is used, allowing Async-CVB to avoid the costly sampling computation ($\tilde{V}$ is also computed in a similar manner),

$$\tilde{N}^{g}_{w,k} = \left[\frac{N^{g}_{w} * N^{\neg p}_{w,1}}{N^{\neg p}_{w}}, \ldots, \frac{N^{g}_{w} * N^{\neg p}_{w,K}}{N^{\neg p}_{w}}\right], \qquad \tilde{V}^{g}_{w,k} = \left[\frac{N^{g}_{w} * V^{\neg p}_{w,1}}{N^{\neg p}_{w}}, \ldots, \frac{N^{g}_{w} * V^{\neg p}_{w,K}}{N^{\neg p}_{w}}\right]. \quad (6)$$

	KOS	NIPS	NYT	PUBMED
Total number of documents in training set	3,000	1,500	300,000	8,200,000
Size of vocabulary	6,906	12,419	102,660	141,043
Total number of words	410,595	1,932,365	99,542,125	737,869,083
Total number of documents in test set	430	184	–	–

Table 2: Data sets used for experiments

Technically, it is possible for Async-CGS to also make use of these expected values; however, these values should be adjusted to be integers to avoid fractional counts in Async-CGS.

Another benefit of using Async-CVB is that tokens with the same word-document indices can be clumped and processed together in one update step at each iteration, with little loss in accuracy. Thus, the running time of Async-CVB is linear in the number of non-zero cells (M) in the word-document matrix, while the running time of Async-CGS is linear in the number of tokens (N). Typically, M is much smaller than N. Table 1 shows the time, space, and communication complexities of the algorithms for each processor, assuming all J documents are of equal size. These tradeoffs suggest that one should consider memory limitations, network latencies, and data set characteristics in order to decide which distributed inference scheme to use.

5. Experiments

We use four text data sets for evaluation: KOS, a data set derived from blog entries (dailykos.com); NIPS, a data set derived from NIPS papers (books.nips.cc); NYT, a collection of news articles from the New York Times (nytimes.com); and PUBMED, a large collection of PubMed abstracts (ncbi.nlm.nih.gov/pubmed). The characteristics of these data sets are summarized in Table 2. These data sets are available at the UCI Machine Learning Repository [39].

For the experiments that measure the accuracy of our algorithms, parallel processors were simulated in software and run on smaller data sets (KOS, NIPS), to enable us to test the statistical limits of our algorithms. Actual parallel hardware is used to measure speedup on larger data sets (NYT, PUBMED). The simulations use a gossip scheme over a fully connected network that lets each processor communicate with one other randomly selected processor at the end of every iteration. For instance, with P=100, there are 50 communicating pairs at each iteration.

In our experiments, the data set is separated into training and test sets. We learn our models on the training set, and then we measure the performance of our algorithms on the test set using perplexity, a widely-used metric in the topic modeling community. Perplexity is defined as the exponentiated average per-word log-likelihood and is widely used as a quantitative metric in speech and language modeling [40]. Moreover, perplexity has been shown to correlate well with other performance measures, such as precision/recall metrics [41] and word error rate in speech recognition [42]. For each of our experiments, we perform $S = 5$ different Gibbs runs for stability purposes, with each run lasting 1500 iterations (unless otherwise noted), and we obtain a sample at the end of each of those runs. The 5 samples are averaged when computing perplexity:

$$\log p(\mathbf{x}^{\text{test}}) = \sum_{jw} N_{jw}^{\text{test}} \log \frac{1}{S} \sum_{s} \sum_{k} \hat{\theta}_{jk}^{s} \hat{\phi}_{wk}^{s} \text{ where } \hat{\theta}_{jk}^{s} = \frac{N_{jk}^{s} + \alpha}{N_{j}^{s} + K\alpha}, \; \hat{\phi}_{wk}^{s} = \frac{N_{wk}^{s} + \eta}{N_{k}^{s} + W\eta} \; . \quad (7)$$

After the model is run on the training data, $\hat{\phi}_{wk}^{s}$ is available in sample s. The tokens in each test document are split into two sets of equal size by random selection. To obtain $\hat{\theta}_{jk}^{s}$, we resample

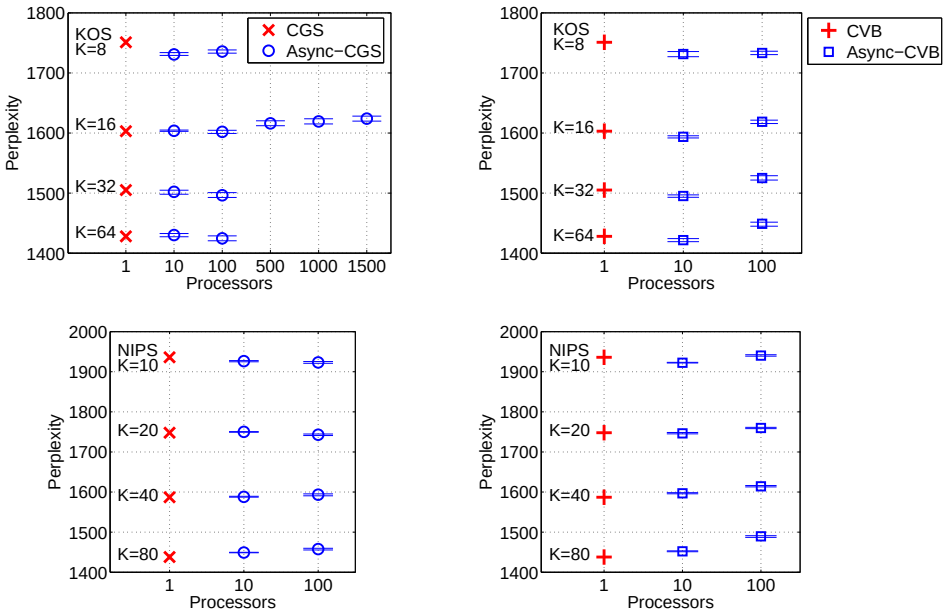

Figure 3: **(a) Top-Left:** Async-CGS perplexities on KOS. **(b) Top-Right:** Async-CVB perplexities on KOS. **(c) Bottom-Left:** Async-CGS perplexities on NIPS. **(d) Bottom-Right:** Async-CVB perplexities on NIPS. For Async-CGS, cache=5 when P≥100, and 3000 iterations were used when P≥500. For Async-CVB, 2000 iterations were used when P=100.

the topic assignments of the first set of tokens of each test document while holding $\hat{\phi}^s_{wk}$ fixed. Perplexity is evaluated on the second set of tokens for each test document, given $\hat{\phi}^s_{wk}$ and $\hat{\theta}^s_{jk}$. This procedure eliminates the possibility of "peeking" at the test data during the training phase.

Since each processor effectively learns a separate local topic model, we can directly compute the perplexity for each processor's local model. In our experiments, we report the average perplexity among processors, and we show error bars denoting the minimum and maximum perplexity among all processors. The variance of perplexities between processors is usually quite small, which suggests that the local topic models learned on each processor are equally accurate.

For KOS and NIPS, we used the same settings for the symmetric Dirichlet priors: $\alpha = 0.1$, $\eta = 0.01$ for CGS, CVB, Async-CGS, and Async-CVB. Note that it is also possible to learn these hyperparameters [41].

5.1. Perplexity results

Figures 3(a,c) show the perplexity results for Async-CGS on the KOS and NIPS data set for varying numbers of topics, K, and varying numbers of processors, P. The variation in perplexities between CGS and Async-CGS is slight and is significantly less than the variation in perplexities as the number of topics K is changed. Figures 3(b,d) show perplexities for Async-CVB. For both CVB and Async-CVB, we use the hybrid technique and we clump tokens with the same word-document indices. There is a slight degradation in perplexity for the P=100 case, since caching was not enabled in Async-CVB. Nonetheless, these results suggest that Async-CGS and Async-CVB converge to solutions of nearly the same quality as CGS and CVB.

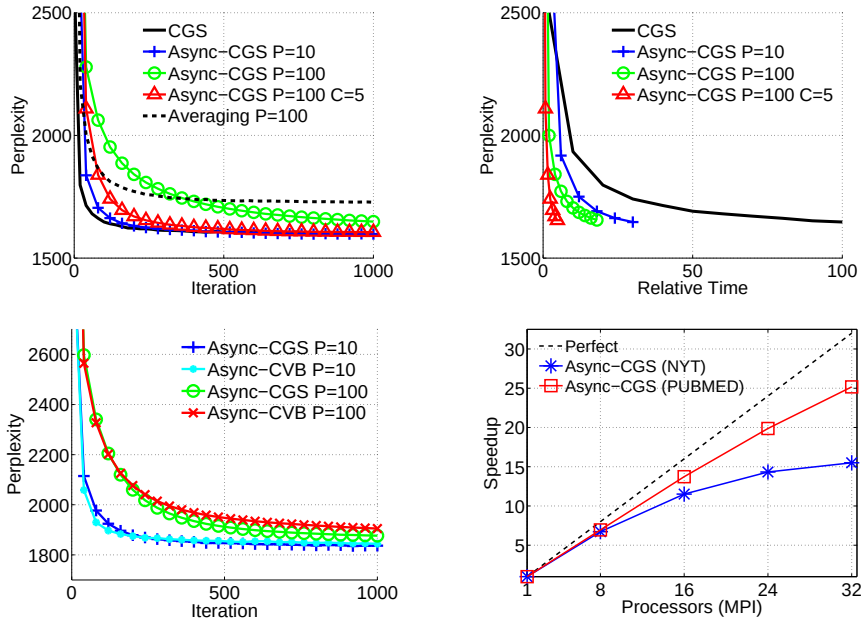

Figure 4: **(a) Top-Left:** Convergence plot for Async-CGS on KOS, K=16. **(b) Top-Right:** Same plot with x-axis as relative time. **(c) Bottom-Left:** Convergence plot comparing Async-CGS to Async-CVB on NIPS, K=20. **(d) Bottom-Right:** Async-CGS speedup results on NYT and PUBMED, K=100, using an MPI cluster.

In Figure 3(a) we stretched the limits of our algorithm by increasing P considerably for the K=16 case. Note that, for P=1500, there are only two documents on each processor. Even in this extreme case, we found that performance was virtually unchanged. As a baseline we ran an experiment where processors never communicate. As the number of processors P was increased from 10 to 1500 the corresponding perplexities increased from 2600 to 5700, notably higher than our Async-CGS algorithm, indicating (unsurprisingly) that processor communication is essential to obtain high quality models.

5.2. Convergence results

Figure 4(a) shows the rate of convergence of Async-CGS. Here, we monitor convergence of the perplexity score on the test data. As the number of processors increases, the rate of convergence slows, since it takes more iterations for information to propagate to all the processors. However, it is important to note that one iteration in real time of Async-CGS is up to P times faster than one iteration of CGS. We show the same curve in terms of estimated real time in Figure 4(b), assuming a parallel efficiency of 0.5, and one can see that Async-CGS converges much more quickly than CGS.

In Figure 4(a), we also show the performance of a baseline asynchronous averaging scheme, where global counts are averaged together: $N_{wk}^{\neg p} \leftarrow (N_{wk}^{\neg p} + N_{wk}^{\neg g})/d + N_{wk}^{g}$. To prevent unbounded count growth, d must be greater than 2, and so we arbitrarily set d to 2.5. While this averaging scheme initially converges quickly, it converges to a final solution that is worse than Async-CGS.

The rate of convergence for Async-CGS P=100 can be significantly improved by letting each processor maintain a cache of previous N_{wk}^g counts of other processors. Figures 4(a,b), C=5, show the improvement made by letting each processor cache the five most recently seen N_{wk}^g's. Note that we still assume a limited bandwidth – each processor does not forward individual cached counts, but instead shares a single matrix of combined cache counts that helps processors achieve faster burn-in time. In this manner, one can elegantly make a tradeoff between time and memory.

We also compare the performance of Async-CVB to Async-CGS on NIPS K=20, without any averaging of samples. On very large corpora, there may be only time to perform one run, and so this experiment simulates a situation of interest. Figure 4(c) reveals that Async-CGS is slightly more accurate than Async-CVB in this case. Both algorithms converge relatively quickly to nearly the same solution, although for the P=10 case, Async-CVB converges at a slightly faster rate than Async-CGS.

Finally, we note that formal MCMC convergence tests are impractical to apply in the case of Async-CGS, given the very large number of parameters in these models. However, one may still assess convergence based on specific measures of interest, such as perplexity and mean entropy of the topic distributions. We conducted several experiments where we ran multiple chains (each governed by our Async-CGS algorithm) initialized at overdispersed starting points, and we calculated the well-known $\hat{R}$ statistic which compares within-chain variance to between-chain variance [43]. We obtained $\hat{R}$ values of 1.1 and 1.2 when using perplexity and mean entropy of the topic distributions as measures, respectively. Since these values are close to 1, they suggest that the chains have essentially converged, relative to our measures of interest. Furthermore, in the case of Async-CVB, one can calculate the variational free energy at each iteration to monitor the progress of the algorithm and assess convergence [23].

5.3. Speedup results

Figure 4(d) shows actual speedup results for Async-CGS on NYT and PUBMED, and the speedups are competitive to those reported for Parallel-CGS [13]. For these experiments, we used the Message Passing Interface (MPI) Library in C to allow processors to communicate with each other. We ran MPI on our cluster of 4 machines, each containing 8 cores. As the data set size grows, the parallel efficiency increases, since communication overhead is dwarfed by the sampling time. These results suggest that substantial speedups can be obtained on large data sets, using our our asynchronous distributed framework.

Since the speedups obtained by our asynchronous algorithms are comparable to the speedups obtained by the synchronous parallel algorithms, a practical issue is determining whether to use synchronous or asynchronous algorithms. Our asynchronous framework provides additional benefits such as fault tolerance, lack of a global bottleneck, pairwise communication, and the use of heterogeneous machines. If none of these asynchronous benefits are needed, we recommend that the synchronous parallel versions be used, since they are easier to implement in practice.

5.4. Other experiments

In certain applications, it is desirable to learn a topic model incrementally as new data arrives. For instance, consider a news article database that receives new articles daily. In our framework, if new data arrives, we assign the new data to a new processor, and then let that new processor enter the "world" of processors with which it can begin to communicate. Our asynchronous approach requires no global initialization or global synchronization step. We do assume a fixed global vocabulary, but one can imagine schemes which allow the vocabulary to grow as well. We

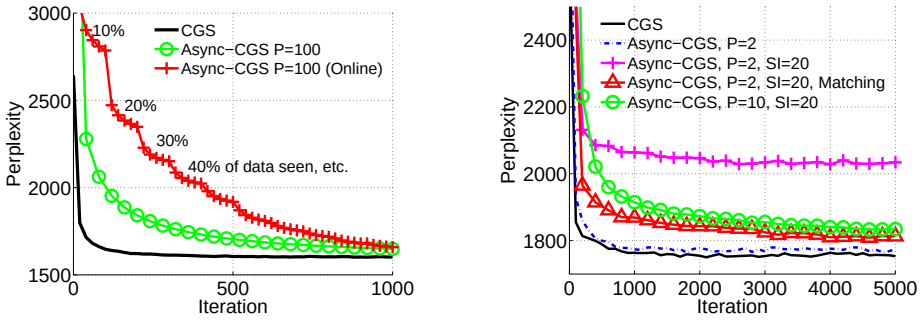

Figure 5: **(a) Left:** Online learning on KOS, K=16 via Async-CGS. **(b) Right:** When the synchronization interval, SI, is increased, Async-CGS converges to a suboptimal solution when P=2.

performed an experiment for Async-CGS where we introduced 10 new processors (each carrying new data) every 100 iterations. In the first 100 iterations, only 10% of the KOS data is known, and every 100 iterations, an additional 10% of the data is added to the system through new processors. Figure 5(a) shows that perplexity decreases as more processors and data are added. After 1000 iterations, the Async-CGS perplexity has converged to the standard LDA perplexity. Thus, in this experiment, learning in an online fashion does not adversely affect the final model.

We also note that it is possible to perform online learning with a fixed number of processors. For instance, one can introduce new "logical" processors with new data, where each actual processor would handle multiple threads, each representing a logical processor. Alternatively, one may augment the document-topic matrix N_{pjk} with the new documents and also add the new counts to N^p_{wk}; however, this approach may require complicated bookkeeping of the topic counts to avoid inconsistencies between processors.

In Figure 5(b), we show a case where our asynchronous algorithms perform suboptimally. In this experiment, we introduced a synchronization interval of 20, i.e. processors only communicate after 20 iterations of local Gibbs sampling. We did not perform averaging of samples in this experiment. In the P=2 case, Async-CGS converges to a suboptimal solution because processor 1's topics start to drift far from processor 2's topics as the synchronization interval increases, causing label switching to potentially occur. The effects of drifting can be mitigated by performing bipartite matching between processor 1's topics and processor 2's topics, via the Hungarian algorithm for bipartite matching [44]. In this experiment, Async-CGS with topic matching performs significantly better than without matching. Note that as P increases, drifting becomes less of an issue, since each processor can only modify $\frac{1}{P}$ of the total topic assignments. Async-CGS P=10 (without matching), performs significantly better than the P=2 case.

In Table 3, we show representative topics learned by CGS, Async-CGS P=10, and Async-CVB P=10 on NIPS, K=20. Note that not all the topics are exactly alike; for instance, in the fourth cell of Table 3, Async-CVB's topic seems to be a hybrid of a "signal processing" topic and a "tree" topic. Nonetheless, the general semantic coherence and similarity between the topics learned by different algorithms suggest that our distributed approach is able to learn high-quality models.

Other situations of interest, such as distributed learning with load imbalance and distributed learning of the HDP model (the non-parametric analogue of LDA), are described in earlier work [31].

CGS	model distribution data gaussian probability parameter likelihood mixture
Async-CGS	model distribution data probability gaussian parameter likelihood mean
Async-CVB	model data distribution gaussian parameter probability likelihood mean
CGS	neuron cell model spike synaptic firing input potential
Async-CGS	neuron cell model synaptic input firing spike activity
Async-CVB	neuron model input cell synaptic spike activity firing
CGS	function bound algorithm theorem threshold number set result
Async-CGS	function bound theorem result number threshold neural set
Async-CVB	function bound algorithm learning number theorem set result
CGS	signal filter frequency noise component channel information source
Async-CGS	signal information frequency channel filter noise component source
Async-CVB	information node component tree nodes signal algorithm independent

Table 3: High probability words of topics learned by CGS, Async-CGS, Async-CVB on NIPS

6. Conclusions

We have presented two different algorithms for distributed learning for LDA. Our perplexity and speedup results suggest that latent variable models such as LDA can be learned in a scalable asynchronous fashion for a wide variety of situations. One can imagine these algorithms being performed by a large network of idle processors, in an effort to mine the terabytes of text information available on the Internet.

A practical issue is determining the number of processors to use for a given data set, based on the tradeoff between the quality of approximation and the savings in processing time. As we have seen in our experiments, there is very little degradation in the quality of solution achieved, even when the number of processors is dramatically increased. Thus, we generally advocate using as many processors as are available. However, if the communication time between processors dominates the time it takes to perform local CGS/CVB sweeps over the data, then perhaps the data set is not large enough to justify using many processors. Another practical issue is the use of count caching to improve the rate of convergence of the algorithm. We generally recommend that caching be performed according to the memory capacities of each processor.

Although processors perform inference locally based on inexact global counts, the asynchronous algorithms proposed in this paper nonetheless produce solutions that are almost identical to those produced by standard single-processor algorithms, but that can be obtained in a fraction of the time. The only case where we found that the distributed algorithms break down is when the processors do not synchronize on a regular basis. Providing a theoretical justification for these distributed topic model algorithms is still an open area of research. More generally, parallelization within a single MCMC/Gibbs sampling run has yet to be fully explored.

Acknowledgments

A shorter version of the material in this paper was presented by the authors at the Neural Information Processing Systems (NIPS) Conference, Vancouver, December 2008. This material is based upon work supported in part by NSF under Award IIS-0083489 (PS, AA), IIS-0447903 and IIS-0535278 (MW), and by an NSF graduate fellowship (AA). PS was also supported by a Google Research Award and by ONR under grant N00014-08-1-1015. MW was also supported by ONR under grant 00014-06-1-073.

References

[1] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent Dirichlet allocation, Journal of Machine Learning Research 3 (2003) 993–1022.

[2] Y. W. Teh, M. I. Jordan, M. J. Beal, D. M. Blei, Hierarchical Dirichlet processes, Journal of the American Statistical Association 101 (476) (2006) 1566–1581.

[3] M. Rosen-Zvi, T. Griffiths, M. Steyvers, P. Smyth, The author-topic model for authors and documents, in: Proceedings of the 20th Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-04), AUAI Press, Arlington, Virginia, 2004, pp. 487–494.

[4] D. Blei, J. Lafferty, Correlated topic models, in: Y. Weiss, B. Schölkopf, J. Platt (Eds.), Advances in Neural Information Processing Systems 18, MIT Press, Cambridge, MA, 2006, pp. 147–154.

[5] W. Li, A. McCallum, Pachinko allocation: DAG-structured mixture models of topic correlations, in: ICML '06: Proceedings of the 23rd International Conference on Machine Learning, ACM, New York, NY, USA, 2006, pp. 577–584.

[6] X. Wei, W. B. Croft, LDA-based document models for ad-hoc retrieval, in: SIGIR '06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, 2006, pp. 178–185.

[7] D. Newman, C. Chemudugunta, P. Smyth, Statistical entity-topic models, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM New York, NY, USA, 2006, pp. 680–686.

[8] H. Misra, O. Cappé, F. Yvon, Using LDA to detect semantically incoherent documents, in: CoNLL '08: Proceedings of the 12th Conference on Computational Natural Language Learning, Association for Computational Linguistics, Morristown, NJ, USA, 2008, pp. 41–48.

[9] H. Asuncion, A. Asuncion, R. Taylor, Software traceability with topic modeling, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE), ACM, New York, NY, USA, 2010.

[10] X. Wang, E. Grimson, Spatial latent Dirichlet allocation, in: J. Platt, D. Koller, Y. Singer, S. Roweis (Eds.), Advances in Neural Information Processing Systems 20, MIT Press, Cambridge, MA, 2008, pp. 1577–1584.

[11] D. Mimno, A. McCallum, Organizing the OCA: Learning faceted subjects from a library of digital books, in: Joint Conference on Digital Libraries, ACM, New York, NY, USA, 2007, pp. 376–385.

[12] R. Nallapati, W. Cohen, J. Lafferty, Parallelized variational EM for latent Dirichlet allocation: An experimental evaluation of speed and scalability, in: ICDM Workshop On High Performance Data Mining, 2007.

[13] D. Newman, A. Asuncion, P. Smyth, M. Welling, Distributed inference for latent Dirichlet allocation, in: J. Platt, D. Koller, Y. Singer, S. Roweis (Eds.), Advances in Neural Information Processing Systems 20, MIT Press, Cambridge, MA, 2008, pp. 1081–1088.

[14] J. Wolfe, A. Haghighi, D. Klein, Fully distributed EM for very large datasets, in: ICML '08: Proceedings of the 25th International Conference on Machine Learning, ACM, New York, NY, USA, 2008, pp. 1184–1191.

[15] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Gossip algorithms: Design, analysis and applications, in: INFOCOM, 2005, pp. 1653–1664.

[16] I. Jolliffe, Principal component analysis, Springer New York, 2002.

[17] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, R. Harshman, Indexing by latent semantic analysis, Journal of the American Society for Information Science 41 (6) (1990) 391–407.

[18] T. Hofmann, Unsupervised learning by probabilistic latent semantic analysis, Machine Learning 42 (1) (2001) 177–196.

[19] J. Pritchard, M. Stephens, P. Donnelly, Inference of population structure using multilocus genotype data, Genetics 155 (2) (2000) 945–959.

[20] E. Erosheva, S. Fienberg, J. Lafferty, Mixed-membership models of scientific publications, Proceedings of the National Academy of Sciences of the United States of America 101 (Suppl 1) (2004) 5220–5227.

[21] W. Buntine, A. Jakulin, Discrete component analysis, Lecture Notes in Computer Science 3940 (2006) 1–33.

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

[23] Y. W. Teh, D. Newman, M. Welling, A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation, in: B. Schölkopf, J. Platt, T. Hoffman (Eds.), Advances in Neural Information Processing Systems 19, MIT Press, Cambridge, MA, 2007, pp. 1353–1360.

[24] M. Welling, Y. W. Teh, H. Kappen, Hybrid variational/Gibbs collapsed inference in topic models, in: Proceedings of the 24th Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-08), AUAI Press, Corvallis, Oregon, 2008, pp. 587–594.

[25] E. Kontoghiorghes, Handbook of Parallel Computing and Statistics, CRC Press, 2006.

[26] G. Forman, B. Zhang, Distributed data clustering can be efficient and exact, SIGKDD Explor. Newsl. 2 (2) (2000) 34–38.

[27] W. Kowalczyk, N. Vlassis, Newscast EM, in: L. K. Saul, Y. Weiss, L. Bottou (Eds.), Advances in Neural Information Processing Systems 17, MIT Press, Cambridge, MA, 2005, pp. 713–720.
[28] A. Rossini, L. Tierney, N. Li, Simple parallel statistical computing in R, Journal of Computational & Graphical Statistics 16 (2) (2007) 399.
[29] Chu, Kim, Lin, Yu, Bradski, Ng, Olukotun, Map-reduce for machine learning on multicore, in: B. Schölkopf, J. Platt, T. Hoffman (Eds.), Advances in Neural Information Processing Systems 19, MIT Press, Cambridge, MA, 2007, pp. 281–288.
[30] A. S. Das, M. Datar, A. Garg, S. Rajaram, Google news personalization: Scalable online collaborative filtering, in: WWW '07, ACM, New York, NY, USA, 2007, pp. 271–280.
[31] A. Asuncion, P. Smyth, M. Welling, Asynchronous distributed learning of topic models, in: D. Koller, D. Schuurmans, Y. Bengio, L. Bottou (Eds.), Advances in Neural Information Processing Systems 21, Cambridge, MA, 2009.
[32] A. Ihler, D. Newman, Bounding sample errors in approximate distributed latent Dirichlet allocation, Large Scale Machine Learning Workshop, NIPS. UCI ICS Technical Report # 09-06 (2009).
URL http://www.ics.uci.edu/~ihler/papers/tr09-06.pdf
[33] L. Younes, Synchronous random fields and image restoration, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (4) (1998) 380–390.
[34] A. Brockwell, Parallel Markov chain Monte Carlo simulation by pre-fetching, Journal of Computational & Graphical Statistics 15, No. 1 (2006) 246–261.
[35] G. Winkler, Image Analysis, Random Fields and Markov Chain Monte Carlo Methods: A Mathematical Introduction (Stochastic Modelling and Applied Probability), Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[36] O. Brun, V. Teuliere, J.-M. Garcia, Parallel particle filtering, Journal of Parallel and Distributed Computing 62 (7) (2002) 1186 – 1202.
[37] D. Earl, M. Deem, Parallel tempering: Theory, applications, and new perspectives, Phys. Chem. Chem. Phys. 7 (2005) 3910–3916.
[38] L. Xiao, S. Boyd, Fast linear iterations for distributed averaging, Systems & Control Letters 53 (1) (2004) 65–78.
[39] A. Asuncion, D. Newman, UCI machine learning repository (2007).
URL http://www.ics.uci.edu/~mlearn/MLRepository.html
[40] F. Jelinek, R. Mercer, L. Bahl, J. Baker, Perplexity – a measure of the difficulty of speech recognition tasks, The Journal of the Acoustical Society of America 62 (1977) S63.
[41] A. Asuncion, M. Welling, P. Smyth, Y. Teh, On smoothing and inference for topic models, in: Proceedings of the 25th Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-09), AUAI Press, Corvallis, Oregon, 2009, pp. 27–34.
[42] D. Klakow, J. Peters, Testing the correlation of word error rate and perplexity, Speech Communication 38 (1-2) (2002) 19–28.
[43] A. Gelman, D. Rubin, Inference from iterative simulation using multiple sequences, Statistical science 7 (4) (1992) 457–472.
[44] H. Kuhn, The Hungarian method for the assignment problem, Naval Research Logistics Quarterly 2 (1955) 83–97.