Journal of Machine Learning Research x (2009) x-xx Submitted 6/08; Published xx/xx

Distributed Algorithms for Topic Models

David Newman NEWMAN@UCI.EDU
Department of Computer Science
University of California, Irvine
Irvine, CA 92697, USA

Arthur Asuncion ASUNCION@ICS.UCI.EDU
Department of Computer Science
University of California, Irvine
Irvine, CA 92697, USA

Padhraic Smyth SMYTH@ICS.UCI.EDU
Department of Computer Science
University of California, Irvine
Irvine, CA 92697, USA

Max Welling WELLING@ICS.UCI.EDU
Department of Computer Science
University of California, Irvine
Irvine, CA 92697, USA

Editor:

Abstract

We describe distributed algorithms for two widely-used topic models, namely the Latent Dirichlet Allocation (LDA) model, and the Hierarchical Dirichet Process (HDP) model. In our distributed algorithms the data is partitioned across separate processors and inference is done in a parallel, distributed fashion. We propose two distributed algorithms for LDA. The first algorithm is a straightforward mapping of LDA to a distributed processor setting. In this algorithm processors concurrently perform Gibbs sampling over local data followed by a global update of topic counts. The algorithm is simple to implement and can be viewed as an approximation to Gibbs-sampled LDA. The second version is a model that uses a hierarchical Bayesian extension of LDA to directly account for distributed data. This model has a theoretical guarantee of convergence but is more complex to implement than the first algorithm. Our distributed algorithm for HDP takes the straightforward mapping approach, and merges newly-created topics either by matching or by topic-id. Using five real-world text corpora we show that distributed learning works well in practice. For both LDA and HDP, we show that the converged test-data log probability for distributed learning is indistinguishable from that obtained with single-processor learning. Our extensive experimental results include learning topic models for two multi-million document collections using a 1024-processor parallel computer.

Keywords: Topic Models, Latent Dirichlet Allocation, Hierarchical Dirichlet Processes, Distributed Parallel Computation

1. Introduction

Very large data sets, such as collections of images or text documents, are becoming increasingly common, with examples ranging from collections of online books at Google and Amazon, to the large collection of images at Flickr. These data sets present major opportunities for machine learning, such as the ability to explore richer and more expressive models than previously possible, and provide new and interesting domains for the application of learning algorithms.

However, the scale of these data sets also brings significant challenges for machine learning, particularly in terms of computation time and memory requirements. For example, a text corpus with one million documents, each containing one thousand words, will require approximately eight GBytes of memory to store the billion words. Adding the memory required for parameters, which usually exceeds memory for the data, creates a total memory requirement that exceeds that available on a typical desktop computer. If one were to assume that a simple operation, such as computing a probability vector over categories using Bayes rule, takes on the order of 10^{-6} seconds per word, then a full pass through the billion words would take 1000 seconds. Thus, algorithms that make multiple passes through the data, for example clustering and classification algorithms, will have run times in days for this sized corpus. Furthermore, for small to moderate sized document sets where memory is not an issue, it would be useful to have algorithms that could take advantage of desktop multiprocessor/multicore technology to learn models in near real-time.

An obvious approach for addressing these time and memory issues is to distribute the learning algorithm over multiple processors. In particular, with P processors, it is somewhat trivial to address the memory issue by distributing $\frac{1}{P}$ of the total data to each processor. However, the computation problem remains non-trivial for a fairly large class of learning algorithms, namely how to combine local processing on each processor to arrive at a useful global solution.

In this general context we investigate distributed algorithms for two widely-used unsupervised learning models: the Latent Dirichlet Allocation (LDA) model, and the Hierarchical Dirichet Process (HDP) model. LDA and HDP models are arguably among the most successful recent learning algorithms for analyzing discrete data such as bags of words from a collection of text documents. However, they can take days to learn for large corpora, and thus, distributed learning would be particularly useful.

The rest of the paper is organized as follows: In Section 2 we review the standard derivation of LDA and HDP. Section 3 presents our two distributed algorithms for LDA and one distributed algorithm for HDP. Empirical results are provided in Section 4. Scalability results are presented in Section 5, and further analysis of distributed LDA is provided in Section 6. A comparison with related models is given in Section 7. Finally, Section 8 concludes the paper.

2. Latent Dirichlet Allocation and Hierarchical Dirichlet Process Model

We start by reviewing the LDA and HDP models. Both LDA and HDP are generative probabilistic models for discrete data such as bags of words from text documents – in this context these models are often referred to as topic models. To illustrate the notation, we refer the reader to the graphical models for LDA and HDP shown in Figure 1.

LDA models each of D documents in a collection as a mixture over K latent topics, with each topic being a multinomial distribution over a vocabulary of W words. For document j, we first draw a mixing proportion θ_j from a Dirichlet with parameter α. For the i^{th} word in the document, a topic $z_{ij} = k$ is drawn with probability $\theta_{k|j}$. Word x_{ij} is then drawn from topic z_{ij}, with x_{ij} taking on value w with probability $\phi_{w|z_{ij}}$. A Dirichlet prior with parameter β is placed on the word-topic distributions ϕ_k.

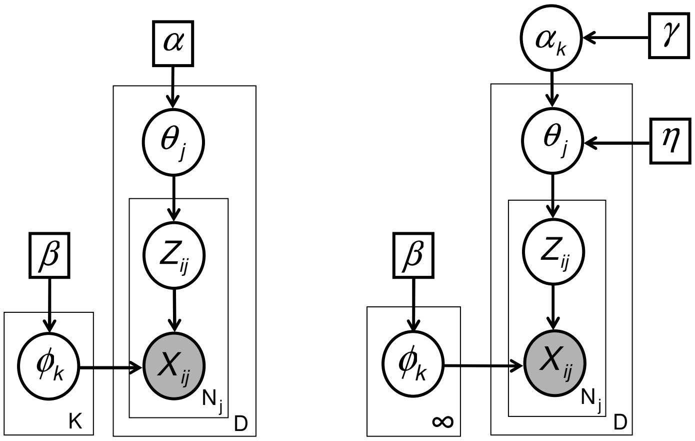

Figure 1: Graphical models for LDA (left) and HDP (right). Observed variables (words) are shaded, and hyperparameters are shown in squares.

Thus, the generative process for LDA is given by

$$\theta_j \sim \mathcal{D}[\alpha] \quad \phi_k \sim \mathcal{D}[\beta] \quad z_{ij} \sim \theta_j \quad x_{ij} \sim \phi_{z_{ij}}. \tag{1}$$

To avoid clutter we denote sampling from a Dirichlet $\theta_j \sim \mathcal{D}[\alpha]$ as shorthand for $[\theta_{1|j}, \ldots, \theta_{K|j}] \sim \mathcal{D}[\alpha, \ldots, \alpha]$, and likewise for ϕ. In this paper, we use symmetric Dirichlet priors for simplicity, unless specified otherwise. The full joint distribution over all parameters and variables is

$$p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta) = \prod_j \frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K} \prod_k \theta_{k|j}^{N_{kj}+\alpha-1} \prod_k \frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \prod_w \phi_{w|k}^{N_{wk}+\beta-1}, \tag{2}$$

where $N_{wkj} = \#\{i : x_{ij} = w, z_{ij} = k\}$, and we use the convention that missing indices are summed out. $N_{kj} = \sum_w N_{wkj}$ and $N_{wk} = \sum_j N_{wkj}$ are the two primary count arrays used in computations, representing the number of words assigned to topic k in document j, and the number of times word w is assigned to topic k in the corpus, respectively. For ease of reading we list commonly used variables in Table 1.

Given the observed words $\mathbf{x} = \{x_{ij}\}$, the task of Bayesian inference for LDA is to compute the posterior distribution over the latent topic assignments $\mathbf{z} = \{z_{ij}\}$, the mixing proportions θ_j, and the topics ϕ_k. Approximate inference for LDA can be performed either using variational methods (Blei et al., 2003) or Markov chain Monte Carlo methods (Griffiths and Steyvers, 2004). In this paper we focus on Markov chain Monte Carlo algorithms for approximate inference. MCMC is widely used as an inference method for a variety of topic models, for example Rosen-Zvi et al. (2004), Li and McCallum (2006), and Chemudugunta et al. (2007) all use MCMC for inference. In the MCMC context, the usual procedure is to integrate out the mixtures $\boldsymbol{\theta}$ and topics $\boldsymbol{\phi}$ in (2) – a procedure called collapsing – and just sample the latent variables $\mathbf{z}$. Given the current state of all but one

	Description
D	Number of documents in collection
W	Number of distinct words in vocabulary
N	Total number of words in collection
K	Number of topics
x_{ij}	i^{th} observed word in document j
z_{ij}	Topic assigned to x_{ij}
N_{wk}	Count of word assigned to topic
N_{kj}	Count of topic assigned in document
ϕ_k	Probability of word given topic k
θ_j	Probability of topic given document j

Table 1: Description of commonly used variables.

variable z_{ij}, the conditional probability of z_{ij} is

$$p(z_{ij} = k|\mathbf{z}^{\neg ij}, \mathbf{x}, \alpha, \beta) \propto \frac{N_{wk}^{\neg ij} + \beta}{\sum_w N_{wk}^{\neg ij} + W\beta} \left(N_{kj}^{\neg ij} + \alpha \right), \tag{3}$$

where the superscript $\neg ij$ means that the corresponding word is excluded in the counts.

HDP is a collection of Dirichlet Processes which share the same topic distributions and can be viewed as the non-parametric extension of LDA. The advantage of HDP is that the number of topics is determined by the data. The HDP model is obtained by taking the following model in the limit as K goes to infinity. Let α_k be top level Dirichlet variables sampled from a Dirichlet with parameter γ/K. The topic mixture for each document, θ_j, is drawn from a Dirichlet with parameters $\eta\alpha_k$. The word-topic distributions ϕ_k are drawn from a base Dirichlet distribution with parameter β. As in LDA, z_{ij} is sampled from θ_j, and word x_{ij} is sampled from the corresponding topic $\phi_{z_{ij}}$. The generative process is given by

$$\alpha_k \sim \mathcal{D}[\gamma/K] \qquad \theta_j \sim D[\eta\alpha_k] \qquad \phi_k \sim D[\beta] \qquad z_{ij} \sim \theta_j \qquad x_{ij} \sim \phi_{z_{ij}}. \tag{4}$$

The posterior distribution is sampled using the direct assignment sampler for HDP described in Teh et al. (2006). As was done for LDA, both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are integrated out, and z_{ij} is sampled from the following conditional distribution:

$$p(z_{ij} = k|\mathbf{z}^{\neg ij}, \mathbf{x}, \alpha, \beta, \eta) \propto \begin{cases} \frac{N_{wk}^{\neg ij} + \beta}{\sum_w N_{wk}^{\neg ij} + W\beta} \left(N_{kj}^{\neg ij} + \eta\alpha_k \right), & \text{if } k \text{ previously used} \\ \frac{\eta\alpha_{\text{new}}}{W}, & \text{if } k \text{ is new.} \end{cases} \tag{5}$$

The sampling scheme for α_k is also detailed in Teh et al. (2006). Note that a small amount of probability mass proportional to α_{new} is reserved for the instantiation of new topics. While HDP is defined to have infinitely many topics, the sampling algorithm only instantiates topics as needed.

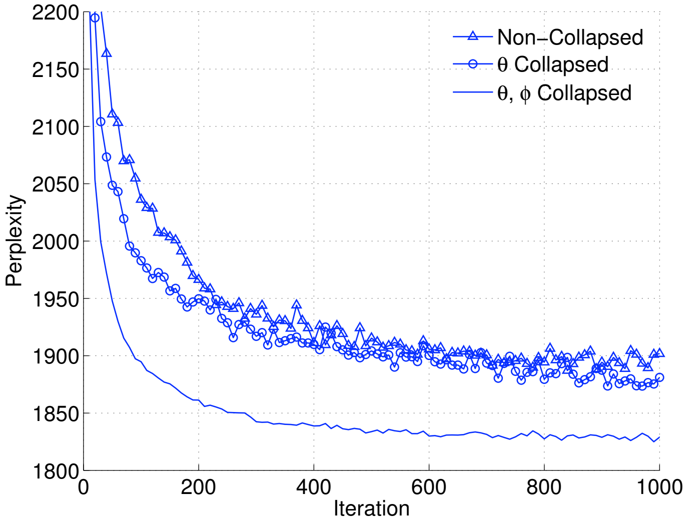

Figure 2: On the NIPS dataset using $K = 20$ topics, the fully collapsed Gibbs sampler (solid line) converges faster than the partially collapsed (circles) and non-collapsed (triangles) samplers.

Need for Distributed Algorithms: One could argue that it is trivial to distribute non-collapsed Gibbs sampling, because sampling of z_{ij} can happen independently given θ_j and ϕ_k, and therefore can be done concurrently. In the non-collapsed Gibbs sampler, one samples z_{ij} given θ_j and ϕ_k, and then samples θ_j and ϕ_k given z_{ij}. Furthermore, if individual documents are not spread across different processors, one can marginalize over just θ_j, since θ_j is processor-specific. In this partially collapsed scheme, the latent variables z_{ij} on each processor can be concurrently sampled, where the concurrency is over processors.

Unfortunately, the non-collapsed and partially collapsed Gibbs samplers exhibit slow convergence due to the strong dependencies between the parameters and latent variables. Generally, we expect faster mixing as more variables are collapsed (Liu et al., 1994; Casella and Robert, 1996). Figure 2 shows, using one of the data sets used throughout our paper, that the log probability of test data (measured as perplexity, which is defined in Section 4) of the non-collapsed and partially collapsed samplers converges more slowly than the fully collapsed sampler.

The slow convergence of partially collapsed and non-collapsed Gibbs samplers motivates the need to devise distributed algorithms for fully collapsed Gibbs samplers. In the following section we introduce distributed topic modeling algorithms that take advantage of the benefits of collapsing both θ and ϕ.

3. Distributed Algorithms for Topic Models

We introduce algorithms for LDA and HDP where the data, parameters, and computation are distributed over distinct processors. We distribute the D documents over P processors, with approximately $D_P = \frac{D}{P}$ documents on each processor. Documents are randomly assigned to processors, although as we will see later, the assignment of documents to processors – ranging from random to highly non-random or adversarial – appears to have little influence on the results. This indifference

Algorithm 1 AD-LDA

```
repeat
  for each processor $p$ in parallel do
    Copy global counts: $N_{wkp} \leftarrow N_{wk}$
    Sample $\mathbf{z}_p$ locally: LDA-Gibbs-Iteration($\mathbf{x}_p$, $\mathbf{z}_p$, $N_{kjp}$, $N_{wkp}$, $\alpha$, $\beta$)
  end for
  Synchronize
  Update global counts: $N_{wk} \leftarrow N_{wk} + \sum_p (N_{wkp} - N_{wk})$
until termination criterion satisfied
```

is somewhat understandable given that converged results from Gibbs sampling are independent of sampling order.

We partition the words from the D documents into $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_p, \ldots, \mathbf{x}_P\}$ and the corresponding topic assignments into $\mathbf{z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_p, \ldots, \mathbf{z}_P\}$, where processor p stores $\mathbf{x}_p$, the words from documents $j = (p-1)D_P + 1, \ldots, pD_P$, and $\mathbf{z}_p$, the corresponding topic assignments. Topic-document counts N_{kj} are likewise distributed as N_{kjp}. The word-topic counts N_{wk} are also distributed, with each processor keeping a separate local copy N_{wkp}.

3.1 Approximate Distributed Latent Dirichlet Allocation

The difficulty of distributing and parallelizing over Gibbs sampling updates (3) lies in the fact that Gibbs sampling is a strictly sequential process. To asymptotically sample from the posterior distribution, the update of any topic assignment z_{ij} can not be performed concurrently with the update of any other topic assignment $z_{i'j'}$. But given the typically large number of word tokens compared to the number of processors, to what extent will the update of one topic assignment z_{ij} depend on the update of any other topic assignment $z_{i'j'}$? Our hypothesis is that this dependence is weak, and therefore we should be able to relax the requirement of sequential sampling of topic assignments and still learn a useful model. One can see this weak dependence in the following common situation. If two processors are concurrently sampling, but sampling different words in different documents (i.e. $w_{ij} \neq w_{i'j'}$), then concurrent sampling will be very close to sequential sampling because the only term affecting the order of operations is the total count of topics $\sum_w N_{wk}$ in the denominator of (3).

The pseudocode for our Approximate Distributed LDA (AD-LDA) algorithm is shown in Algorithm 1. After distributing the data and parameters across processors, AD-LDA performs simultaneous LDA Gibbs sampling on each of the P processors. After processor p has swept through its local data and updated topic assignments $\mathbf{z}_p$, the processor has modified count arrays N_{kjp} and N_{wkp}. The topic-document counts N_{kjp} are distinct because of the document index, j, and will be consistent with the topic assignments $\mathbf{z}$. However, the word-topic counts N_{wkp} will in general be different on each processor, and not globally consistent with $\mathbf{z}$. To merge back to a single and consistent set of word-topic counts, we perform a reduce operation on N_{wkp} across all processors to update the global counts. After the synchronization and update operations, each processor has the same values in the N_{wkp} array which are consistent with the global vector of topic assignments $\mathbf{z}$. Note that N_{wkp} is not the result of P separate LDA models running on separate data. In particular, each word-topic count array reflects all the counts, not just those local to that processor, so for every processor $\sum_{wk} N_{wkp} = N$, where N is the total number of words in the corpus. As in LDA, the

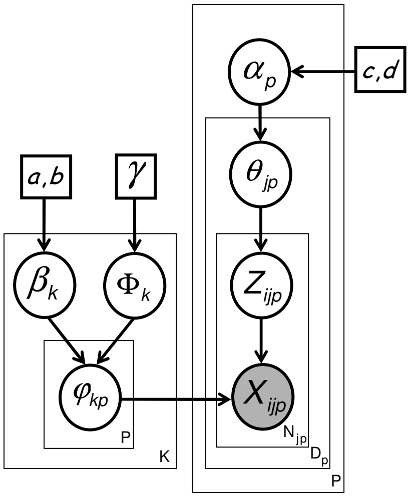

Figure 3: Graphical model for Hierarchical Distributed Latent Dirichlet Allocation.

algorithm can terminate either after a fixed number of iterations, or based on some suitable MCMC convergence metric.

We chose the name *Approximate* Distributed LDA because in this algorithm we are no longer asymptotically sampling from the true posterior, but to an approximation of the true posterior. Nonetheless, we will show in our experimental results that the approximation made by Approximate Distributed LDA works very well.

3.2 Hierarchical Distributed Latent Dirichlet Allocation

In AD-LDA we constructed an algorithm where each processor is independently computing an LDA model, but at the end of each sweep through a processor's data, a consistent global array of topic counts N_{wk} is reconstructed. This global array of topic counts could be thought of as a parent topic distribution, from which each processor draws its own local topic distribution.

Using this intuition, we created a Bayesian model reflecting this structure, as shown in Figure 3. Our Hierarchical Distributed LDA model (HD-LDA) places a hierarchy over word-topic distributions, with Φ_k being the global or parent word-topic distribution and φ_{kp} the local word-topic distributions on each processor. The local word-topic distributions φ_{kp} are drawn from Φ_k according to a Dirichlet distribution with a topic-dependent strength parameter β_k, for each topic $k = 1 \ldots K$. The model on each processor is simply an LDA model. The generative process is given by:

$$\begin{gathered}\beta_k \sim \mathcal{G}[a,b] \qquad \alpha_p \sim \mathcal{G}[c,d] \qquad \theta_{jp} \sim \mathcal{D}[\alpha_p] \qquad \Phi_k \sim \mathcal{D}[\gamma] \qquad \varphi_{kp} \sim \mathcal{D}[\beta_k \Phi_k] \\ z_{ijp} \sim \theta_{jp} \qquad x_{ijp} \sim \varphi_{z_{ijp}p}\end{gathered} \tag{6}$$

From this generative process, we derive Gibbs sampling equations for HD-LDA. The derivation is based on the Teh et al. (2006) sampling schemes for Hierarchical Dirichlet Processes. As was done for LDA, we start by integrating out φ and θ. The collapsed distribution of $\mathbf{z}_p$ and $\mathbf{x}_p$ on processor p is given by:

$$p(\mathbf{z}_p, \mathbf{x}_p | \alpha_p, \beta, \Phi) = \prod_j \left[\frac{\Gamma(K\alpha_p)}{\Gamma(N_{jp} + K\alpha_p)} \prod_k \frac{\Gamma(N_{kjp} + \alpha_p)}{\Gamma(\alpha_p)} \right] \prod_k \left[\frac{\Gamma(\beta_k)}{\Gamma(N_{kp} + \beta_k)} \prod_w \frac{\Gamma(N_{wkp} + \beta_k \Phi_{w|k})}{\Gamma(\beta_k \Phi_{w|k}))} \right]. \tag{7}$$

From this we derive the conditional probability for sampling a topic assignment z_{ijp}. Unlike AD-LDA, the topic assignments on any processor are now conditionally independent of the topic assignments on the other processors given Φ, thus allowing each processor to sample $\mathbf{z}_p$ concurrently. The conditional probability of z_{ijp} is

$$p(z_{ijp} = k | \mathbf{z}_p^{\neg ijp}, \mathbf{x}, \alpha_p, \beta, \Phi) = (N_{kjp}^{\neg ijp} + \alpha_p) \frac{(N_{wkp}^{\neg ijp} + \beta_k \Phi_{w|k})}{(N_{kp}^{\neg ijp} + \beta_k)}. \tag{8}$$

The full derivation of the Gibbs sampling equations for HD-LDA is provided in Appendix A, which lists the complete set of sampling equations for α_p, β_k, and Φ_k.

The pseudocode for our Hierarchical Distributed LDA algorithm is given in Algorithm 2. Each variable in this model is either local or global, depending on whether inference for the variable is computed locally on a processor or globally, requiring information from all processors. Local variables include α, θ, φ, z, and x. Global variables include β and Φ. Each processor uses Gibbs sampling to sample its local variables concurrently. After each sweep through the processor's data, the global variables are sampled. Note that, unlike AD-LDA, HD-LDA is performing strictly correct sampling for its model.

HD-LDA can be viewed as a mixture model with P LDA mixture components with equal mixing weights. In this view the data have been hard-assigned to their respective clusters (i.e. processors), and the parameters of the clusters are generated from a shared prior distribution.

Algorithm 2 HD-LDA

```
repeat
  for each processor $p$ in parallel do
    Sample $\mathbf{z}_p$ locally: LDA-Gibbs-Iteration($\mathbf{x}_p$, $\mathbf{z}_p$, $N_{kjp}$, $N_{wkp}$, $\alpha_p$, $\beta_k\Phi_k$)
    Sample $\alpha_p$ locally
  end for
  Synchronize
  Sample: $\beta_k$, $\Phi_k$
  Broadcast: $\beta_k$, $\Phi_k$
until termination criterion satisfied
```

3.3 Approximate Distributed Hierarchical Dirichlet Processes

Our third distributed algorithm, Approximate Distributed HDP, takes the same approach as AD-LDA. Processors concurrently run HDP for a single sweep through their local data. After all of the processors sweep through their data, a synchronization and update step is performed to create a

Algorithm 3 AD-HDP

```
repeat
  for each processor $p$ in parallel do
    Sample $z_p$ locally: HDP-Gibbs-Iteration($x_p$, $z_p$, $N_{kjp}$, $N_{wkp}$, $\alpha_{kp}$, $\beta$, $\gamma$, $\eta$)
    Report $N_{wkp}$, $\alpha_{kp}$ to master node
  end for
  Synchronize
  Update global counts (and merge new topics): $N_{wk} \leftarrow N_{wk} + \sum_p (N_{wkp} - N_{wk})$
  $\alpha_k \leftarrow (\sum_p \alpha_{kp})/P$
  Sample: $\eta$, $\alpha_k$, $\gamma$
  Broadcast: $N_{wk}$, $\alpha_k$, $\gamma$, $\eta$
until termination criterion satisfied
```

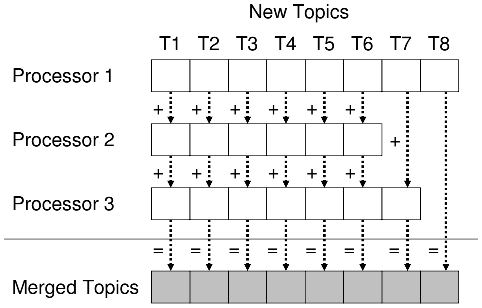

Figure 4: The simplest method to merge new topics in AD-HDP is by integer topic label.

single set of globally-consistent word-topic counts N_{wk}. We refer to the distributed version of HDP as AD-HDP, and provide the pseudocode in Algorithm 3.

Unlike AD-LDA, which uses a fixed number of topics, individual processors in AD-HDP may instantiate new topics during the sampling phase, according to the HDP sampling equation (5). During the synchronization and update step, instead of treating each processor's new topics as distinct, we merge new topics that were instantiated on different processors. Merging new topics helps limit unnecessary growth in the total number of topics and allows AD-HDP to produce more of a global model.

There are several ways to merge newly created topics on each processor. A simple way – inspired by AD-LDA – is to merge new topics based on their integer topic label. A more complicated way is to match new topics across processors based on topic similarity.

In the first merging scheme, new topics are merged based on their integer topic label. For example, assume that we have three processors, and at the end of a sweep through the data, processor one has 8 new topics, processor two has 6 new topics, and processor three has 7 new topics. Then during synchronization, all these new topics would be aligned by topic label and their counts summed, producing 8 new global topics, as shown in Figure 4.

While this merging of new topics by topic-id may seem suboptimal, it is computationally simple and efficient. We will show in the next section that this merging generally works well in practice, even when processors only have a small amount of data. We suggest that even if the merging by

Algorithm 4 Greedy Matching of New Topics for AD-HDP

```
Initialize global set of new topics, $G$, to be processor 1's set of new topics
for $p$ = 2 to P do
  for topic $t$ in processor p's set of new topics do
    Initialize score array
    for topic $g$ in $G$ do
      score[$g$] = symmetric-KL-divergence($t$,$g$)
    end for
    if min(score) $<$ threshold then
      Add $t$'s counts to the topic in $G$ corresponding to min(score)
    else
      Augment $G$ with the new topic $t$
    end if
  end for
end for
```

topic-id is initially quite random, the subsequent dynamics align the topics in a sensible manner. We will also show that AD-HDP ultimately learns models with similar perplexity to HDP, irrespective of how new topics are merged.

We also investigate more complicated schemes for merging new topics in AD-HDP, beyond the simple approach of merging by topic-id. Instead of aligning new topics based topic-id it is possible to align new topics using a similarity metric such as symmetric Kullback-Leibler divergence. However, finding the optimal matching of topics in the case where $P > 2$ is NP-hard (Burkard and Çela, 1999). Thus, we consider approximate schemes: bipartite matching using a reference processor, and greedy matching.

In the bipartite matching scheme, we select a reference processor and perform bipartite matching between every processor's new topics and the set of new topics of the reference processor. The bipartite match is computed using the Hungarian algorithm, which runs in $O(T^3)$, producing an overall complexity of $O(PT^3)$ where T is the maximum number of new topics on a processor. We implemented this scheme but did not find any improvement over AD-HDP with merging by topic-id.

In the greedy matching scheme, new topics on each processor are sequentially compared to a global set of new topics. This global set is initialized to the first processor's set of new topics. If a new topic is sufficiently different from every topic in the global set, the number of topics in the global set is incremented; otherwise, the counts for that new topic are added to those from the closest match in the global set. A threshold is used to determine whether a new topic is sufficiently different from another topic. The worst case complexity of this algorithm is $O(P^2T^2)$ – this is the case where every new topic is found to be different from every other new topic in the global set. Increasing this threshold will make it more likely for new topics to merge with the topics already in the global set (instead of incrementing the set), causing the expected running time of this merging algorithm to be linear in the number of processors. The pseudocode of this greedy matching scheme is shown in Algorithm 4. This algorithm is run after each iteration of AD-HDP to produce a global set of new topics. We show in the next section that this greedy matching scheme significantly improves the rate of convergence for AD-HDP.

	KOS	NIPS	WIKIPEDIA	PUBMED	NEWSGROUPS
D_{train}	3,000	1,500	2,051,929	8,200,000	19500
W	6,906	12,419	120,927	141,043	27,059
N	467,714	2,166,058	344,941,756	737,869,083	2,057,207
D_{test}	430	184	-	-	498

Table 2: Characteristics of data sets used in experiments.

4. Experiments

The purpose of our experiments is to investigate how our distributed topic model algorithms, AD-LDA, HD-LDA and AD-HDP, perform when compared to their sequential counterparts, LDA and HDP. We are interested in two aspects of performance: the quality of the model learned, measured by log probability of test data; and the time taken to learn the model. Our primary data sets for these experiments were KOS blog entries, from dailykos.com, and NIPS papers, from books.nips.cc. We chose these relatively small data sets to allow us to perform a large number of experiments. Both data sets were split into a training set and a test set. Size parameters for these data sets are shown in Table 2. For each corpus, D is the number of documents, W is the vocabulary size and N is the total number of words. Two larger data sets WIKIPEDIA, from en.wikipedia.org, and PUBMED, from pubmed.gov were used for speedup experiments, described in Section 5. For precision-recall experiments we used the NEWSGROUPS data set, taken from the UCI Machine Learning Repository. All the data sets used in this paper can be downloaded from the UCI Machine Learning Repository (Asuncion and Newman, 2007).

Using the KOS and NIPS data sets, we computed test set perplexities for a range of topics K, and for numbers of processors, P, ranging from 1 to 3000. The distributed algorithms were initialized by first randomly assigning topics to words in $\mathbf{z}$, then counting topics in documents, N_{kjp}, and words in topics, N_{wkp}, for each processor. For each run of LDA, AD-LDA, and HD-LDA, a sample was taken at 500 iterations of the Gibbs sampler, which is well after the typical burn-in period of the initial 200-300 iterations. For each run of HDP and AD-HDP, we allow the Gibbs sampler to run for 3000 iterations, to allow the number of topics to grow. In our perplexity experiments, multiple processors were simulated in software by separating data, running sequentially through each processor, and simulating the global synchronization and update steps. For the speedup experiments, computations were run on 64 to 1024 processors on a 2000+ processor parallel supercomputer.

The following set of hyperparameters was used for the experiments, where hyperparameters are shown as variables in squares in the graphical models in Figures 1 and 3. For AD-LDA we set $\alpha = 0.1$ and $\beta = 0.01$. For AD-HDP we set $\beta = 0.01$, $\eta \sim$Gamma$(2, 1)$ and $\gamma \sim$Gamma$(10, 1)$. While η and γ could have also been fixed, resampling these hyperparameters allows for more robust topic growth, as described by Teh et al. (2006). For LDA and AD-LDA we fixed the hyperparameters α and β, but these priors could also be learned using sampling.

Selection of hyperparameters for HD-LDA was guided by our experience with AD-LDA. For AD-LDA, $\sum_w N_{wkp} \approx \frac{N}{K}$, but for HD-LDA $\sum_w N_{wkp} \approx \frac{N}{PK}$, so we choose a and b to make the mode of $\beta_k = \frac{(P-1)N}{PK}$ to simulate the inclusion of global counts in N_{wkp} as is done in AD-LDA. We set $\gamma = 2/K$, because it is important to scale γ by the number of topics to prevent oversmoothing when the counts are spread thinly among many topics. Finally, we choose c and d to make the mode of $\alpha_p = 0.1$, matching the value of α used in our LDA and AD-LDA experiments. Specifically, we set: $a = \frac{(P-1)N}{PK}$, $b = 1$, $c = 0.1 * 10 + 1$ and $d = 0.1$.

To systematically evaluate our distributed topic model algorithms, AD-LDA, HD-LDA and AD-HDP, we measured performance using test set perplexity, which is computed as $\text{Perp}(\mathbf{x}^{\text{test}}) = \exp(-\frac{1}{N^{\text{test}}} \log p(\mathbf{x}^{\text{test}}))$. For every test document, half the words at random are designated for fold-in, and the remaining words are used as test. The document mixture θ_j is learned using the fold-in part, and log probability of the test words is computed using this mixture, ensuring that the test words are not used in estimation of model parameters. For AD-LDA, the perplexity computation exactly follows that of LDA, since a single set of topic counts N_{wk} are saved when a sample is taken. In contrast, all P copies of N_{wkp} are required to compute perplexity for HD-LDA. Except where stated, perplexities are computed for all algorithms using $S = 10$ samples from the posterior from ten independent chains using

$$\log p(\mathbf{x}^{\text{test}}) = \sum_{j,w} N_{jw}^{\text{test}} \log \frac{1}{S} \sum_s \sum_k \theta_{k|j}^s \phi_{w|k}^s \qquad \theta_{k|j}^s = \frac{\alpha + N_{kj}^s}{K\alpha + N_j^s} \qquad \phi_{w|k}^s = \frac{\beta + N_{wk}^s}{W\beta + N_k^s}. \tag{9}$$

This perplexity computation follows the standard practice of averaging over multiple chains when making predictions with LDA models trained via Gibbs sampling, as discussed in Griffiths and Steyvers (2004). Averaging over ten samples significantly reduces perplexity compared to using a single sample from one chain. While we perform averaging over multiple samples to improve the estimate of perplexity, we have also observed similar relative results across our algorithms when we use a single sample to compute perplexity.

Analogous perplexity calculations are used for HD-LDA and AD-HDP. With HD-LDA we additionally compute processor-specific responsibilities, since test documents do not belong to any particular processor, unlike the training documents. Each processor learns a document mixture θ_{jp} using the fold-in part for each test document. For each processor, the likelihood is calculated over the words in the fold-in part in a manner analogous to (9), and these likelihoods are normalized to form the responsibilities, r_p. To compute perplexity, we compute the likelihood over the test words, using a responsibility-weighted average of probabilities over all processors:

$$\log p(\mathbf{x}^{\text{test}}) = \sum_{j,w} N_{jw}^{\text{test}} \log \sum_p \frac{r_p}{S} \sum_s \sum_k \theta_{k|jp}^s \phi_{w|kp}^s \tag{10}$$

$$\text{where} \quad \theta_{k|jp}^s = \frac{\alpha_p + N_{kjp}^s}{K\alpha_p + N_{jp}^s}, \qquad \phi_{w|kp}^s = \frac{\beta_k \Phi_{w|k} + N_{wkp}^s}{\beta_k + N_{kp}^s}.$$

Computing perplexity in this manner prevents the possibility of seeing or using test words during the training and fold-in phases.

4.1 Perplexity

The perplexity results for KOS and NIPS in Figure 5 clearly show that the model perplexity is essentially the same for the distributed models AD-LDA and AD-HDP at $P = 10$ and $P = 100$ as their single-processor versions at $P = 1$. The figures show the test set perplexity, versus number of processors, P, for different numbers of topics K for the LDA-type models, and also for the HDP-models which learn the number of topics. The $P = 1$ perplexity is computed by LDA (circles) and HDP (triangles), and we use our distributed algorithms – AD-LDA (crosses), HD-LDA (squares), and AD-HDP (stars) – to compute the $P = 10$ and $P = 100$ perplexities. The variability in perplexity as a function of the number of topics is much greater than the variability due to the number of processors. Note that there is essentially no perplexity difference between AD-LDA and HD-LDA.

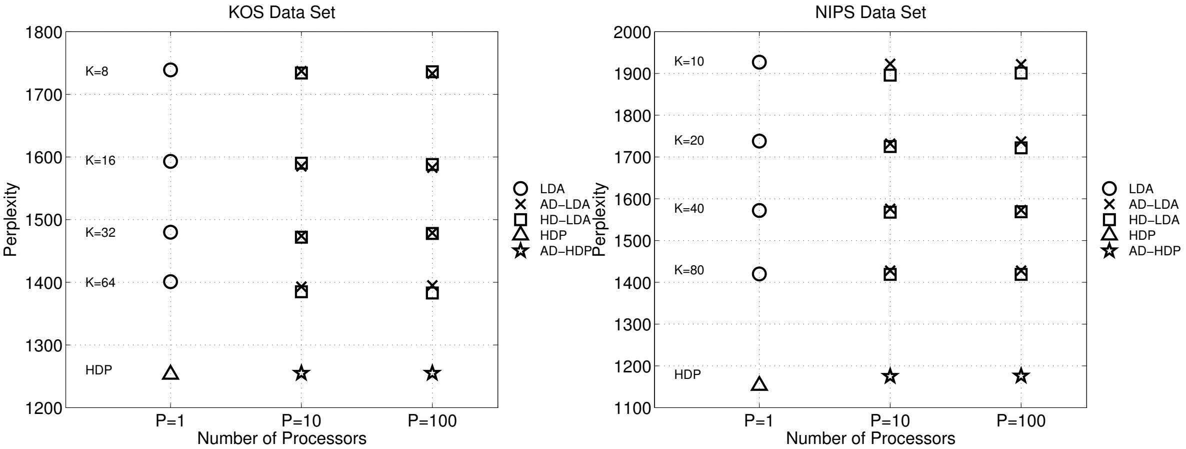

Figure 5: Test perplexity on KOS (left) and NIPS (right) data versus number of processors P. $P = 1$ corresponds to LDA and HDP. At $P = 10$ and $P = 100$ we show AD-LDA, HD-LDA and AD-HDP.

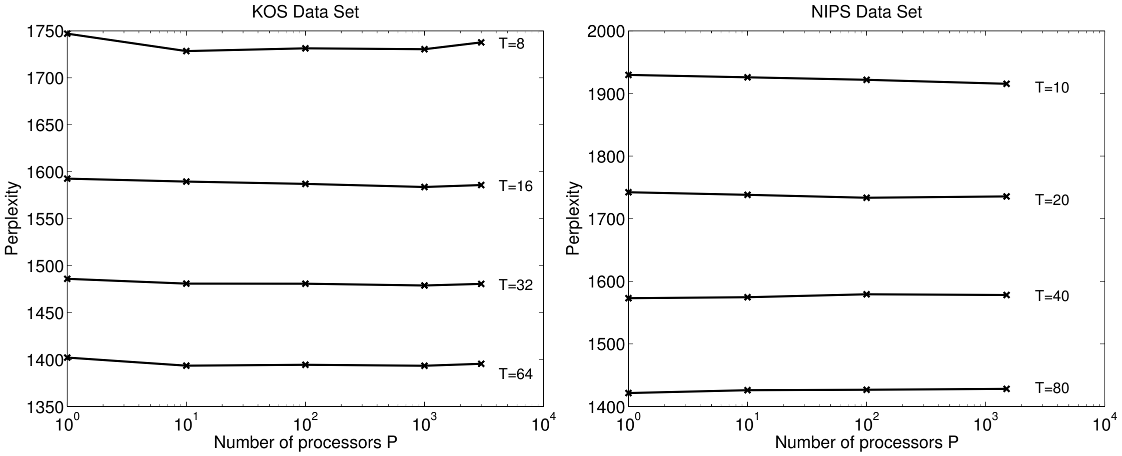

Figure 6: AD-LDA test perplexity versus number of processors up to the limiting case of number of processors equal to number of documents in collection. Left plot shows perplexity for KOS and right plot shows perplexity for NIPS.

Even in the limit of a large number of processors, the perplexity for the distributed algorithms matches that for the sequential version. In fact, in the limiting case of just one document per processor, $P = 3000$ for KOS and $P = 1500$ for NIPS, we see that the perplexities of AD-LDA are generally no different to those of LDA, as shown in the rightmost point in each curve in Figure 6.

AD-HDP instantiates fewer topics but produces a similar perplexity to HDP. The average number of topics instantiated by HDP on KOS was 669 while the average number of topics instantiated by AD-HDP was 490 ($P = 10$) and 471 ($P = 100$). For NIPS, HDP instantiated 687 topics while AD-HDP instantiated 569 ($P = 10$) and 569 ($P = 100$) topics. AD-HDP instantiates fewer topics because of the merging across processors of newly-created topics. The similar perplexity results for AD-HDP compared to HDP, despite the fewer topics, is partly due to the relatively small probability mass in many of the topics.

Despite no formal convergence guarantees, the approximate distributed algorithms, AD-LDA and AD-HDP, converged to good solutions in every single experiment (of the more than one hundred) we conducted using multiple real-world data sets. We also tested both our distributed LDA algorithms with adversarial/non-random distributions of topics across processors using synthesized data. One example of an adversarial distribution of documents is where each document only uses a single topic, and these documents are distributed such that processor p only has documents that are about topic p. In this case the distributed topic models have to learn the correct set of P topics, even though each processor only sees local documents that pertain to just one of the topics. We ran multiple experiments, starting with 1000 documents that were hard-assigned to $K = 10$ topics (i.e. each document is only about one topic), and distributing the 1000 documents over $P = 10$ processors, where each processor contained documents belonging to the same topic (an analogy is one processor only having documents about sports, the next processor only having documents about arts, and so on). The perplexity performance of AD-LDA and HD-LDA under these adversarial/non-random distribution of documents was as good as the performance when the documents were distributed randomly, and as good as the performance of single-processor LDA.

To demonstrate that the low perplexities obtained from the distributed algorithms with $P = 100$ processors are not just due to averaging effects, we split the NIPS corpus into one hundred 15-document collections, and ran LDA separately on each of these hundred collections. The test perplexity at $K = 40$ computed by averaging 100-separate LDA models was 2117, significantly higher than the $P = 100$ test perplexity of 1575 for AD-LDA and HD-LDA. This shows that a baseline approach of simple averaging of results from separate processors performs much worse than the distributed coordinated learning algorithms that we propose in this paper.

4.2 Convergence

One could imagine that distributed algorithms, where each processor only sees its own local data, may converge more slowly than single-processor algorithms where the data is global. Consequently, we performed experiments to see whether our distributed algorithms were converging at the same rate as their sequential counterparts. If the distributed algorithms were converging slower, the computational gains of parallelization would be reduced. Our experiments consistently showed that the convergence rate for the distributed LDA algorithms was just as fast as those for the single processor case. As an example, Figure 7 shows test perplexity versus iteration of the Gibbs sampler for the NIPS data at $K = 20$ topics. During burn-in, up to iteration 200, the distributed algorithms are actually converging slightly faster than single processor LDA. Note that one iteration of AD-LDA or

HD-LDA on a parallel multi-processor computer only takes a fraction (at best $\frac{1}{P}$) of the wall-clock time of one iteration of LDA on a single processor computer.

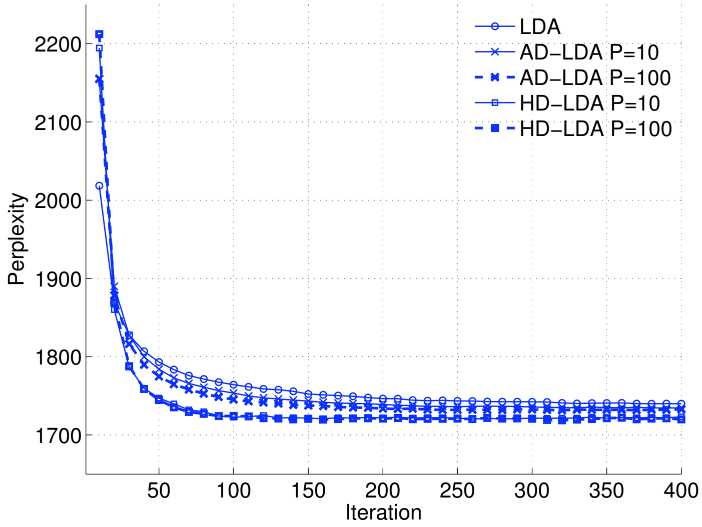

Figure 7: Convergence of test perplexity versus iteration for the distributed algorithms AD-LDA and HD-LDA using the NIPS data set and $K = 20$ topics.

We see slightly different convergence behavior in the non-parametric topic models. AD-HDP converges more slowly than HDP, as shown in Figure 8, due to AD-HDP's heavy averaging of new topics resulting from merging by topic-id (i.e. no matching). This slower convergence may partially be a result of the lower number of topics instantiated. The number of new topics instantiated in one pass of AD-HDP is limited to the maximum number of new topics instantiated on any one processor. For example, in the right plot, after 500 iterations, HDP has instantiated 360 topics, whereas AD-HDP has instantiated 210 ($P = 100$) and 250 ($P = 10$) topics. Correspondingly, at 500 iterations, the perplexity of HDP is lower than the perplexity of AD-HDP. After three thousand iterations, AD-HDP produces the same perplexity as HDP, which is reassuring because it indicates that AD-HDP is ultimately producing a model that has the same predictive ability as HDP. We observe a similar result for the NIPS data set.

One way to accelerate the rate of convergence for AD-HDP is to match newly generated topics by similarity instead of by topic-id. Figure 9 shows that performing the greedy matching scheme for new topics as described in Algorithm 4 significantly improves the rate of convergence for AD-HDP. In this experiment, we used a threshold of 2 for determining topic similarity. The number of topics increases at a faster rate for AD-HDP with matching, since the greedy matching scheme is more flexible in that the number of new topics at each iteration is not limited to the maximum number of new topics instantiated on any one processor. The results show that the greedy matching scheme enables AD-HDP $P = 100$ to converge almost as quickly as HDP. In practice, only a few new topics are generated locally on each processor each iteration, and so the computational overhead of this heuristic matching scheme is minimal relative to the time for Gibbs sampling.

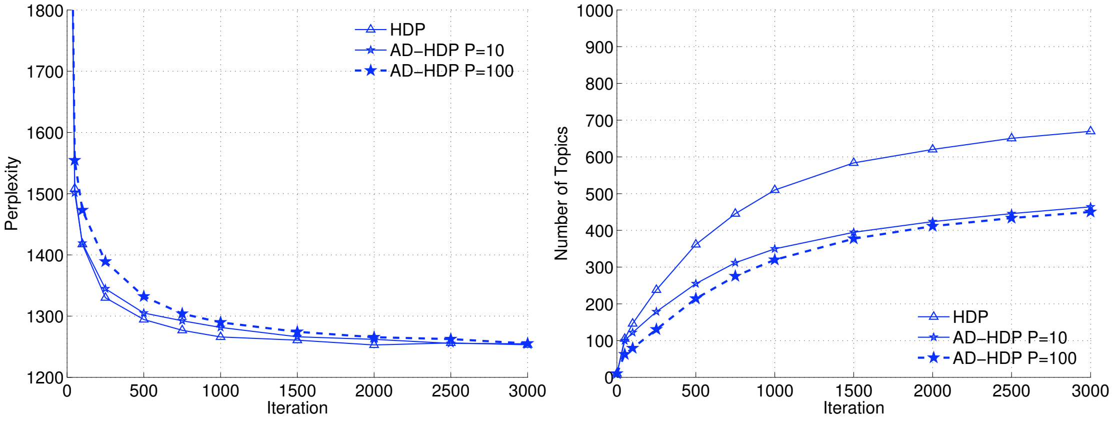

Figure 8: Results for HDP versus AD-HDP with no matching. Left plot shows test perplexity versus iteration for HDP and AD-HDP. Right plot shows number of topics versus iteration for HDP and AD-HDP. Results are for the KOS data set.

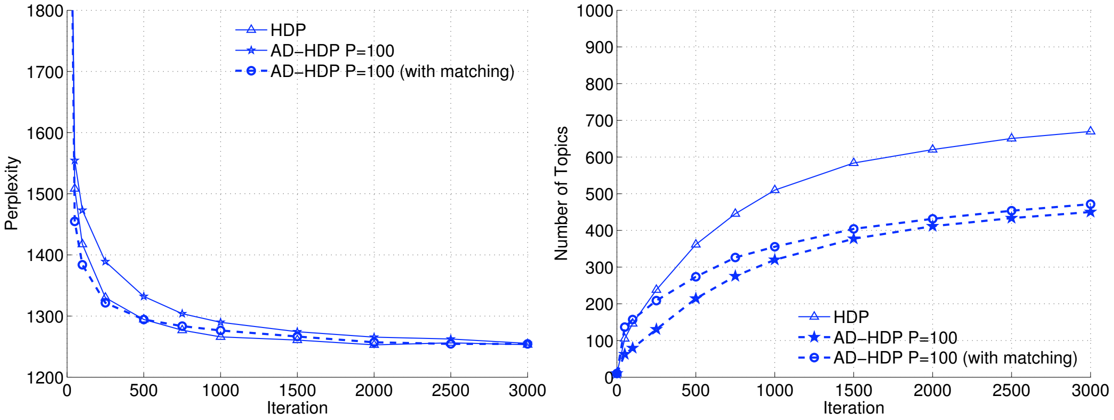

Figure 9: Results for HDP versus AD-HDP with greedy matching. Left plot shows test perplexity versus iteration for HDP and AD-HDP. Right plot shows number of topics versus iteration for HDP and AD-HDP. Results are for the KOS data set.

To further check that the distributed algorithms were performing comparably to their single processor counterparts, we ran experiments to investigate whether the results were sensitive to the number of topics used in the models, in case the distributed algorithms' performance worsens when the number of topics becomes very large. Figure 10 shows the test perplexity computed on the NIPS data set, as a function of the number of topics, for the LDA algorithms and a fixed number of processors $P = 10$ (the results for the KOS data set were quite similar and therefore not shown). The perplexities of the different algorithms closely track each other as number of topics, K, increases. In fact, in some cases HD-LDA produces slightly lower perplexities than those of single processor

LDA. This lower perplexity may be due to the fact that in HD-LDA test perplexity is computed using P sets of topic parameters, thus it has more parameters than AD-LDA to better fit the data.

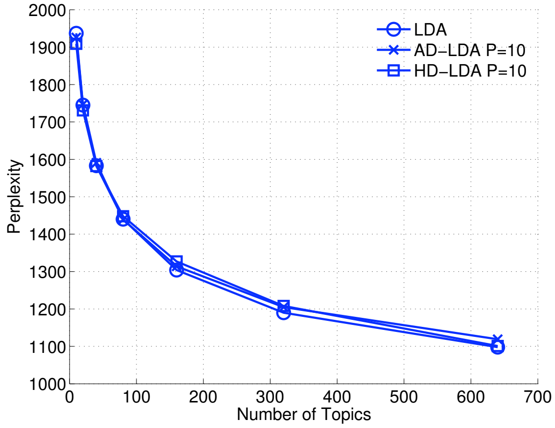

Figure 10: Test perplexity versus number of topics using the NIPS data set (S=5).

4.3 Precision and Recall

In addition to our experiments measuring perplexity, we also performed precision/recall calculations using the NEWSGROUPS data set, where each document's corresponding newsgroup is the class label. In this experiment we use LDA and AD-LDA to learn topic models on the training data. Once the model is learned, each test document can be treated as a ”query”, where the goal is to retrieve relevant documents from the training set. For each test document, the training documents are ranked according to how probable the test document is under each training document's mixture θ_j and the set of topics $\boldsymbol{\phi}$. From this ranking, one can calculate mean average precision and area under the ROC curve.

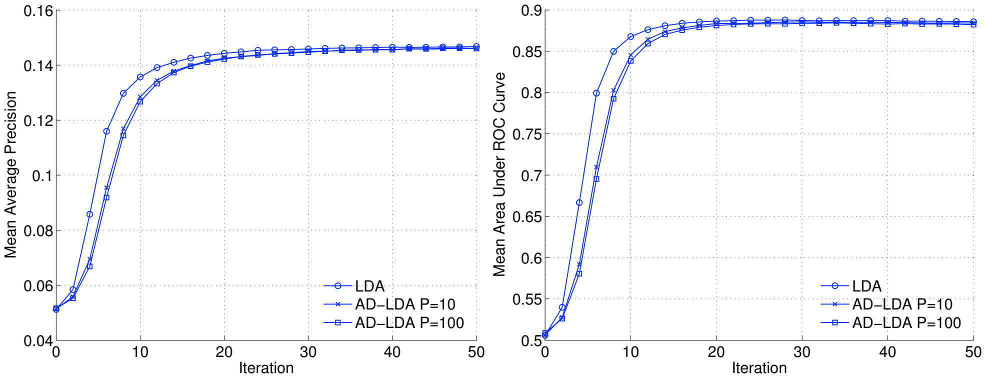

Figure 11: Precision/recall results: (left) Mean average precision for LDA/AD-LDA. (right) Area under the ROC curve for LDA/AD-LDA.

Figure 11 shows the mean average precision and the area under the ROC curve achieved by LDA and AD-LDA, plotted versus iteration. LDA performs slightly better than AD-LDA for the first 20 iterations, but AD-LDA catches up and converges to the same mean average precision and area under the ROC curve as LDA. This again shows that our distributed/parallel version of LDA produces a very similar result to the single-processor version.

5. Scalability

The primary motivation for developing distributed algorithms for LDA and HDP is to have highly scalable algorithms, in terms of memory and computation time. Memory requirements depend on both memory for data and memory for model parameters. The memory for the data scales with N, the total number of words in the corpus. The memory for the parameters is linear in the number of topics K, which is either fixed for the LDA models or learned for the HDP models. The per-processor per-iteration time and space complexity of LDA and AD-LDA are shown in Table 3. AD-LDA's memory requirement scales well as collection sizes grow, because while corpus size (N and D) can get arbitrarily large, which can be offset by increasing the number of processors, P, the vocabulary size W will tend to asymptote, or at least grow more slowly. Similarly the time complexity scales well since the leading order term NK is divided by P.

The communication cost of the reduce operation, denoted by C in the table, represents the time taken to perform the global sum of the count difference $\sum_p (N_{wkp} - N_{wk})$. This is executed in $\log P$ stages and can be implemented efficiently in standard language/protocols such as MPI, the Message Passing Interface. Because of the additional KW term, parallel efficiency will depend on $\frac{N}{PW}$, with increasing efficiency as this ratio increases. Space and time complexity of HD-LDA are similar to that of AD-LDA, but HD-LDA has bigger constants. For a given number of topics, K, we argue that AD-HDP has similar time complexity as AD-LDA.

We performed large-scale speedup experiments with just AD-LDA instead of all three of our distributed topic modeling algorithms because AD-LDA produces very similar results to HD-LDA, but with significantly less computation. We expect that relative speedup performance for HD-LDA and AD-HDP should follow that for AD-LDA.

	LDA	AD-LDA
Space	$N + K(D + W)$	$\frac{1}{P}(N + KD) + KW$
Time	NK	$\frac{1}{P}NK + KW + C$

Table 3: Space and time complexity of LDA and AD-LDA.

We used two multi-million document data sets, WIKIPEDIA and PUBMED, for speedup experiments on a large-scale supercomputer. The supercomputer used was DataStar, a 15.6 TFlop terascale machine at San Diego Supercomputer Center built from 265 IBM P655 8-way compute nodes. We implemented a parallel version of AD-LDA using the Message Passing Interface protocol. We ran AD-LDA on WIKIPEDIA using $K = 1000$ topics and PUBMED using $K = 2000$ topics distributed over $P = 64, 128, 256, 512$ and 1024 processors. The speedup results, shown in Figure 12, show relatively high parallel efficiency, with approximately 700 times speedup for WIKIPEDIA and 800 times speedup for PUBMED when using $P = 1024$ processors, corresponding to parallel efficiencies of approximately 0.7 and 0.8 respectively. This speedup is computed relative to the

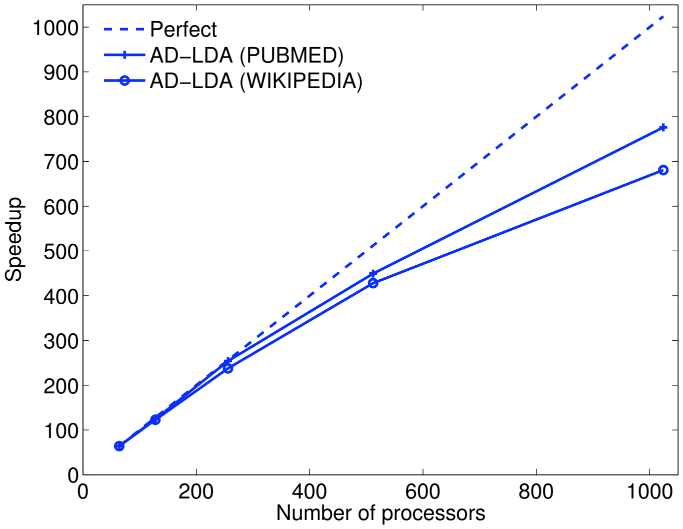

Figure 12: Parallel speedup results for 64 to 1024 processors on multi-million document datasets WIKIPEDIA and PUBMED.

time per iteration when using $P = 64$ processors (i.e. at $P = 64$ processors speedup=64), since it is not possible, due to memory limitations, to run these models on a single processor. Multiple runs were timed for both WIKIPEDIA and PUBMED, and the resulting variation in timing was less than 1%, so error bars are not shown in the figure. We see slightly higher parallel efficiency for PUBMED versus WIKIPEDIA because PUBMED has a larger amount of computation per unit data communicated, $\frac{N}{PW}$.

This speedup dramatically reduces the learning time for large topic models. If we were to learn a $K = 2000$ topic model for PUBMED using LDA on a single processor, it would require over 300 days instead of the 10 hours required to learn the same model using AD-LDA on 1024 processors. In our speedup experiments on these large data sets, we did not directly investigate latency or communication bandwidth effects. Nevertheless, one could expect that if the communication time becomes very long compared to the computation time, then it may be worth doing multiple Gibbs sampling sweeps on a processor's local data before performing the synchronization and global update step. In Section 6 we further examine this question of frequency of synchronizations. The relative time for communication versus computation also effects the weak scaling of parallelization, where the problem size increases linearly with the number of processors. We expect that parallel efficiency will be relatively constant for weak scaling since $\frac{N}{PW}$ is constant.

In addition to the large-scale speedup experiments run on the 1024-processor parallel supercomputer, we also performed small-scale speedup experiments for AD-HDP on an 8-node parallel cluster running MPI. Using the NIPS data set we measured parallel efficiencies of 0.75 and 0.5 for $P = 4$ and $P = 8$. The latter result on 8 processors means that the HDP model for NIPS can be learned four times faster than on a single processor.

6. Analysis of Approximate Distributed LDA

Finally, we investigate the dynamics of AD-LDA learning using toy data to get further insight into how AD-LDA is working. While we have shown experimental results showing that AD-LDA produces models with similar perplexity and similar convergence rates to LDA, it is not obvious why this algorithm works so well in practice. Our toy example has $W = 3$ words and $K = 2$ topics. We generated document collections according to the LDA generative process given by (1). We chose a low dimension vocabulary, W, so that we could plot the evolution of the Gibbs sampler on a two-dimensional word-topic simplex. We first generated data, then learned models using LDA and AD-LDA.

The left plot of Figure 13 shows the L_1 distance between the model's estimate of a particular topic-word distribution and the true distribution, as a function of Gibbs iteration, for both single-processor LDA and AD-LDA with $P = 2$. LDA and AD-LDA have qualitatively the same three-phase learning dynamics. The first four or so iterations (labeled *initialize*) correspond to somewhat random movement close to the randomly initialized starting point. In the next phase (labeled *burn-in*) both algorithms rapidly move in parameter space toward the posterior mode. And finally after burn-in (labeled *stationary*) both are sampling around the mode. In the right plot we show the similarity between AD-LDA and LDA samples taken from the equilibrium distribution – here plotted on the two-dimensional planar simplex corresponding to the three-word topic distribution.

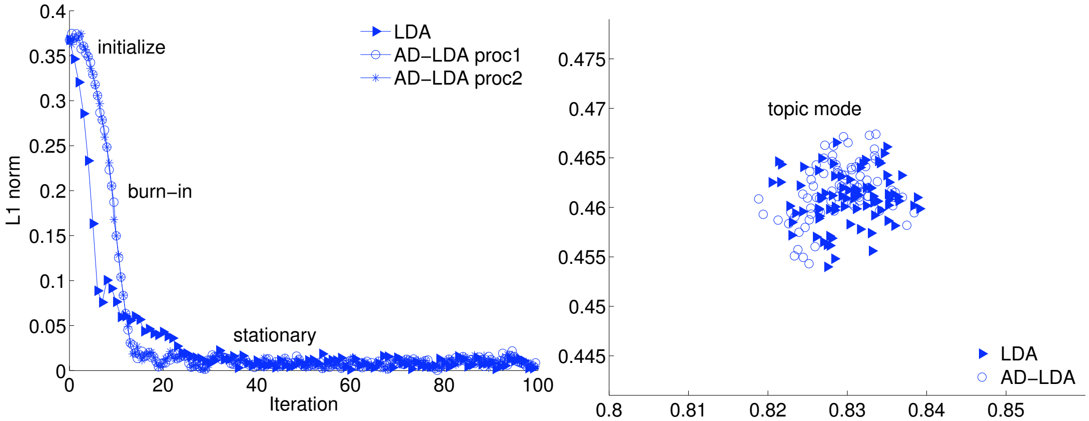

Figure 13: (Left) L_1 distance to the mode for LDA and for $P = 2$ AD-LDA. (Right) Closeup of 50 samples of ϕ (projected onto the topic simplex) taken from the equilibrium distribution, showing the similarity between LDA and $P = 2$ AD-LDA. Note the zoomed scale in this figure.

The left plot of Figure 14 depicts the same trajectory shown in Figure 13 left, projected onto the topic simplex. This plot shows the paths in parameter space of each model, and the same three-phase learning dynamics: taking a few small steps near the starting point, moving up to the true solution, and then sampling near the posterior mode for the rest of the iterations. For each Gibbs iteration, the parameters corresponding to each of the two individual processors, and those parameters after merging, are shown for AD-LDA. One can see the alternating pattern of two separate (but close) parameter estimates on each processor, followed by a merged estimate. We observed that after the initial few iterations, the individual processor steps and the merge step each resulted in a move closer to the mode. One might worry that the AD-LDA algorithm would get trapped close to the initial starting point, e.g., due to repeated label switching or oscillatory behavior of topic labeling across processors. In practice we have consistently observed that the algorithm quickly discards such configurations due to the stochastic nature of the moves and latches onto a consistent and stable labeling that rapidly moves it toward the posterior mode. The figure clearly illustrates that LDA and AD-LDA have qualitatively similar learning dynamics. The right plot in Figure 14 illustrates the same qualitative behavior as in the left plot, but now for $P = 10$ processors.

Interestingly, across a wide range of experiments, we observed that the variance in the AD-LDA word-topic distribution samples is typically only about 70% of the variance in LDA topic samples. Since the samplers are not the same it makes sense that the posterior variance differs (i.e. is underestimated) by the parallel sampler. We expect less variance because AD-LDA ignores fluctuations in the bulk of N_{wk}. Nonetheless, all of our experiments indicate that the posterior mode and means found by the parallel sampler are essentially the same as those found by the sequential sampler.

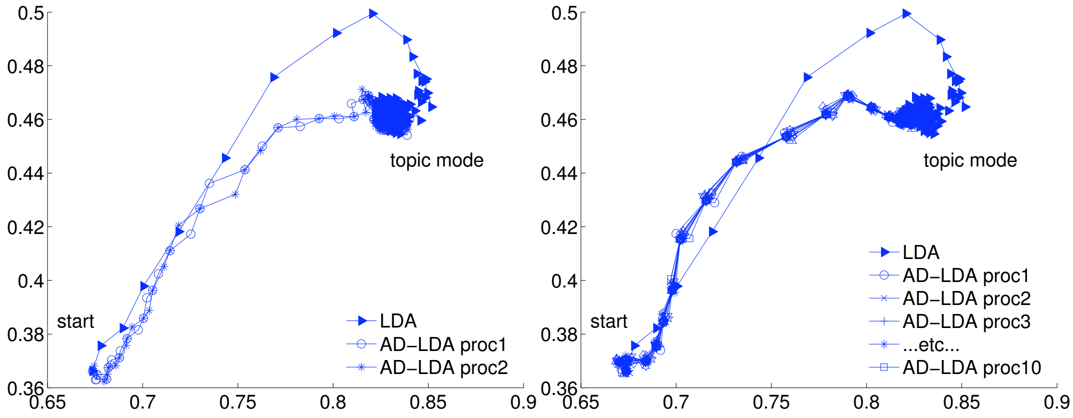

Figure 14: (Left) Projection of topics onto simplex, showing convergence to mode for $P = 2$. (Right) Same as left plot, but with $P = 10$.

Another insight can be gained by thinking of LDA as an approximation to stochastic descent in the space of assignment variables $\mathbf{z}$. On a single processor, one can view Gibbs sampling during burn-in as a stochastic algorithm to move up the likelihood surface. With multiple processors, each processor computes an upward direction in its own subspace, keeping all other directions fixed. The global update step then recombines these directions by vector-addition, in the same way as one would compute a gradient using finite differences. This is expected to be accurate as long as the surface is locally convex or concave, but will break down at saddle-points. We conjecture AD-LDA works reliably because saddle points are unstable and rare because the posterior is usually highly peaked for LDA models and high-dimensional count data sets.

While we see similar perplexities for AD-LDA compared to LDA, we could further ask if the AD-LDA algorithm is producing any bias in its estimates of the model parameters. To test this, we

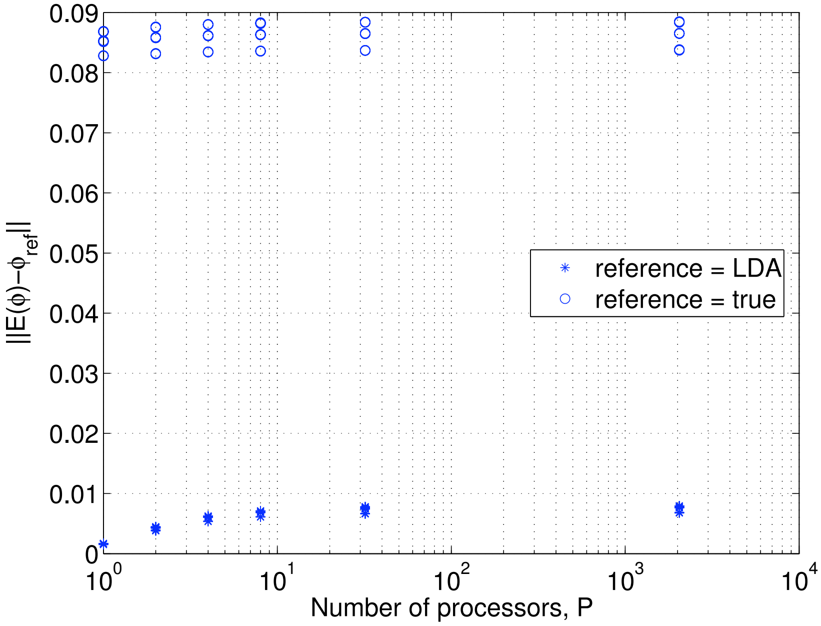

Figure 15: Average L_1 error in word-topic distribution versus P for AD-LDA.

performed a series of experiments where we generated synthetic data sets according to the LDA generative process, with known word-topic distributions ϕ^*. We then learned LDA and AD-LDA models from each of the simulated data sets. We computed the expected value of the AD-LDA topics $E(\phi)$ and compared this to two reference values, ϕ_{ref} one based on the true distribution, $\phi_{\text{ref}} = \phi^*$, the other based on multiple LDA samples, $\phi_{\text{ref}} = E[\phi_{\text{LDA}}]$. Figure 15 shows that AD-LDA is much closer to the LDA topics $E[\phi_{\text{LDA}}]$ than either are to the true topics ϕ^*, telling us that the sampling variation in learning LDA models from finite data sets is much greater than the variation between LDA and AD-LDA on the same data sets.

When Does AD-LDA Fail? In all of our experiments thus far, we have seen that our distributed algorithms learn models with equivalent predictive power as their non-distributed counterparts. However, when global synchronizations are done less frequently (i.e., when the synchronization step is performed after multiple Gibbs sampling sweeps through local data), the distributed algorithms may converge to suboptimal solutions.

When the synchronization interval is increased dramatically, it is possible for AD-LDA to converge to a suboptimal solution. This can happen because the topics (with the same integer label) on each processor can drift far apart, so that topic k on one processor diverges from topic k on another processor. In Figure 16, we show the results of an experiment on KOS where synchronizations only occur once every 100 iterations. For $P = 2$ processors, AD-LDA performs significantly worse than LDA. The $P = 2$ processor case is the worst case for AD-LDA, since one half of the total words on each processor have the freedom to drift. In contrast, when $P = 100$ processors, each processor can only locally modify 1/100th of the topic assignments, and so the topics on each processor can not drift far from the global set of topic counts at the previous iteration. Bipartite matching significantly improves the perplexity in the $P = 2$ processor case, suggesting that the lack of communication has indeed caused the topics to drift apart. Fortunately, topic drifting becomes less of a problem as more processors are used, and can be eliminated by frequent synchronization. It is also important to note that AD-LDA $P = 2$, where processors synchronize after every iteration, gives essentially identical results as LDA. Our recommendation in practice is to perform the synchronization and count updates after each iteration of the Gibbs sampler. As shown earlier in the paper, this leads to performance that is essentially indistinguishable from LDA. Since most multi-processor computing hardware will tend to have communication bandwidth matched to processor speed (i.e. faster and/or more processors usually come with a faster communication network), synchronizing after each iteration of the Gibbs sampler will usually be the optimal strategy.

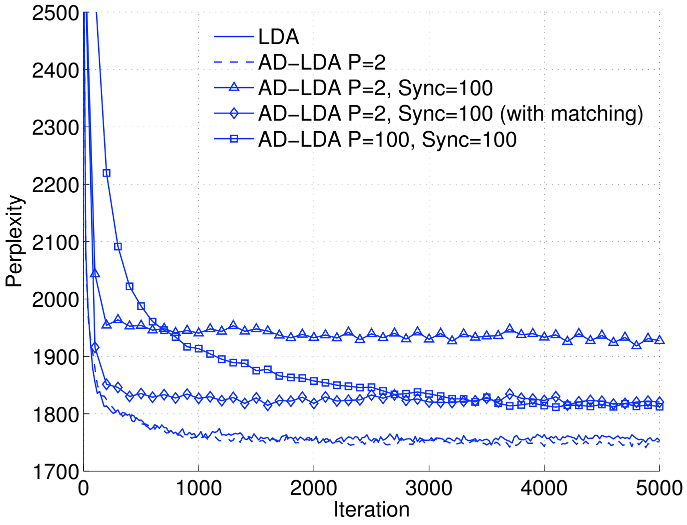

Figure 16: Test perplexity versus iteration where synchronizations between processors only occur every 100 iterations, KOS, $K = 16$.

7. Related Work

Approximate inference for topic models such as LDA and HDP can be carried out using a variety of methods, the most common being variational methods and Markov chain Monte Carlo methods. Previous efforts to parallelize these algorithms have focused on variational methods, which are often straightforward to cast in a distributed framework. For example, Blei et al. (2002) and Nallapati et al. (2007) describe distributed variational EM methods for LDA. In their distributed variational approach, the computationally expensive E-step is easily parallelized because the document-specific variational parameters are independent. Wolfe et al. (2008) investigate the parallelization of both the E and M-steps of variational EM for LDA, under a variety of computer network topologies. In these cases the distributed version of LDA produces identical results to the sequential version of the algorithm. However, memory for variational inference in LDA scales as MK, where M is the number of distinct document-word pairs in the corpus. For typical English-language corpora, the total number of words in the corpus is less than twice the number of distinct document-word pairs

($N < 2M$), so M can be considered on the order of N. Since M is usually much larger than the number of documents, D, this memory requirement of MK is not nearly as scalable as that the memory requirement of $N + DK$ for MCMC methods.

Parallelized versions of various machine learning algorithms have also been developed. Forman and Zhang (2000) describe a parallel k-means algorithm, and W. Kowalczyk and N. Vlassis (2005) describe an asynchronous parallel EM algorithm for Gaussian mixture learning. A parallel EM algorithm for Probabilistic Latent Semantic Analysis, implemented using Google's MapReduce framework, was described in Das et al. (2007). A review of how to parallelize an array of standard machine learning algorithms using MapReduce was presented by Chu et al. (2007). Rossini et al. (2007) presents a framework for statisticians that allows for the parallel computing of independent tasks within the R language.

While many of these EM algorithms are readily parallelizable, Gibbs sampling of dependent variables (such as topic assignments) is fundamentally sequential and therefore difficult to parallelize. One way to parallelize Gibbs sampling is to run multiple independent chains in parallel to obtain multiple samples; however, this multiple-chain approach does not address the fact that the burn-in within each chain may take a long time. Furthermore, for some applications, one is not interested in multiple samples from independent chains. For example, if we wish to learn topics for a very large document collection, one is usually satisfied with mean values of word-topic distributions taken from a single chain.

One can parallelize a single MCMC chain by decomposing the variables into independent non-interacting blocks that can be sampled concurrently (Kontoghiorghes, 2005). However, when the variables are not independent, sampling variables in parallel is not possible. Brockwell (2006) presents a general parallel MCMC algorithm based on pre-fetching, but it is not practical for learning topic models because it discards most of its computations which makes it relatively inefficient. It is possible to construct partially parallel Gibbs samplers, in which the samples are independently accepted with some probability. In the limit as this probability goes to zero, this sampler will approach the sequential Gibbs sampler, as explained in P. Ferrari et al. (1993). However, this method is also not practical when learning topic models because it is computationally inefficient. Younes (1998) shows the existence of exact parallel samplers that make use of periodic synchronous random fields. However there is no known method for constructing such a sampler.

Our HD-LDA model is similar to the DCM-LDA model presented by Mimno and McCallum (2007). There the authors perform topic modeling on a collection of books by learning a different topic model for each book and then clustering these learned topics together to find global topics. In this model, the concept of a book is directly analogous to our concept of a processor. DCM-LDA uses Stochastic EM along with agglomerative clustering to learn topics, while our HD-LDA follows a fully Bayesian approach for inference. HD-LDA also differs from other topic hierarchies found in the literature. The Hierarchical Dirichlet Process model of Teh et al. (2006) places a deeper hierarchical prior on the topic mixture, instead of on the word-topic distributions. The Pachinko Allocation Model presented by Li and McCallum (2006) deals with a document-specific hierarchy of topic-assignments. These types of hierarchies do not directly facilitate proper parallel Gibbs sampling as is done in HD-LDA.

8. Conclusions

We have proposed three different algorithms for distributing across multiple processors Gibbs sampling for LDA and HDP. With our approximate distributed algorithm, AD-LDA, we sample from an approximation to the posterior distribution by allowing different processors to concurrently sample topic assignments on their local subsets of the data. Despite having no formal convergence guarantees, AD-LDA works very well empirically and is easy to implement. With our hierarchical distributed model, HD-LDA, we adapt the underlying LDA model to map to the distributed processor architecture. This model is more complicated than AD-LDA, but it inherits the usual convergence properties of Markov chain Monte Carlo. We discovered that careful selection of hyperparameters was critical to making HD-LDA work well, but this selection was clearly informed by AD-LDA. Our distributed algorithm AD-HDP followed the same approach as AD-LDA, but with an additional step to merge newly instantiated topics.

Our proposed distributed algorithms learn LDA models with predictive performance that is no different than single-processor LDA. On each processor they burn-in and converge at the same rate as LDA, yielding significant speedups in practice. For HDP, our distributed algorithm eventually produced the same perplexity as the single-processor version of HDP. Prior to reaching the converged perplexity result, AD-HDP had higher perplexity than HDP since the merging of new topics by label slows the rate of topic growth. We also discovered that matching new topics by similarity significantly improves AD-HDP's rate of convergence.

The space and time complexity of these distributed algorithms make them scalable to run very large data sets, for example, collections with billions to trillions of words. Using two multi-million document datasets, and running computations on a 1024-processor parallel supercomputer, we showed how one can achieve a 700-800 times reduction in wall-clock time by using our distributed approach.

There are several potentially interesting research directions that can be pursued using the algorithms proposed here as a starting point. One research direction is to use more complex schemes that allow data to adaptively move from one processor to another. The distributed schemes presented in this paper can also be used to parallelize topic models that are based on or derived from LDA and HDP, and beyond that a potentially larger class of graphical models.

Acknowledgments

This material is based upon work supported by the National Science Foundation. DN and PS were supported by NSF grants SCI-0225642, CNS-0551510, and IIS-0083489. DN was also supported by a Google Research Award. PS was also supported by ONR grant N00014-18-1-101 and a different Google Research Award. AA was supported by an NSF graduate fellowship. MW was supported by NSF grants IIS-0535278 and IIS-0447903, and ONR grant 00014-06-1-073. Computations were performed at SDSC under an MRAC Allocation.

Appendix A.

The auxiliary variable method explained in Escobar and West (1995) and Teh et al. (2006) is used to sample α, β, and Φ. To derive Gibbs sampling equations, we use the following expansions:

$$\frac{\Gamma(u)}{\Gamma(u+n)} = \frac{1}{\Gamma(n)} B(u,n) = \frac{1}{\Gamma(n)} \int_0^1 t^{u-1}(1-t)^{n-1}\, dt \tag{11}$$

$$\frac{\Gamma(u+n)}{\Gamma(u)} = \sum_{s=0}^{n} S(n,s)(u)^s \qquad \text{(S is Stirling number of first kind)} \tag{12}$$

The first expansion follows from the definition of the Beta function, and the second expansion makes use of the Stirling number of the first kind to rewrite the factorial (see (Abramowitz and Stegun, 1964)).

Now we derive the sampling equation for α_p. Combining the collapsed distribution (7) with the prior on α_p (6) gives the posterior distribution for α_p[1]:

$$P(\alpha_p|\_) \propto \prod_j \left[\frac{\Gamma(K\alpha_p)}{\Gamma(N_{jp}+K\alpha_p)} \prod_k \frac{\Gamma(N_{kjp}+\alpha_p)}{\Gamma(\alpha_p)} \right] \alpha_p^{c-1} e^{-d\alpha_p} \tag{13}$$

Using the expansions (11,12) we introduce the auxiliary variables $\mathbf{t}$ and $\mathbf{s}$:

$$P(\alpha_p, \mathbf{t}, \mathbf{s}|\_) \propto \left[\prod_j t_j^{K\alpha_p - 1}(1-t_j)^{N_{jp}-1}\, dt_j \right] \left[\prod_j \prod_k S(N_{kjp}, s_{kjp}) \alpha_p^{s_{kjp}} \right] \alpha_p^{c-1} e^{-d\alpha_p} \tag{14}$$

The joint distribution above allows us to create sampling equations for α_p, $\mathbf{t}$, and $\mathbf{s}$:

$$\begin{aligned} P(\alpha_p|\mathbf{t}, \mathbf{s}, \_) &\propto \left[\prod_j t_j^{K\alpha_p} \right] \left[\prod_j \prod_k \alpha_p^{s_{kjp}} \right] \alpha_p^{c-1} e^{-d\alpha_p} \\ &= \text{Gamma} \left[c + \sum_j \sum_k s_{kjp}; d - K \sum_j \log(t_j) \right] \end{aligned} \tag{15}$$

$$\begin{aligned} P(t_j|\alpha_p, \mathbf{s}, \_) &\propto t_j^{K\alpha_p - 1}(1-t_j)^{N_{jp}-1} \\ &= \text{Beta}\left[K\alpha_p, N_{jp}\right] \end{aligned} \tag{16}$$

$$\begin{aligned} P(s_{kjp}|\alpha_p, \mathbf{t}, \_) &\propto S(N_{kjp}, s_{kjp}) \alpha_p^{s_{kjp}} \\ &= \text{Antoniak}\left[N_{kjp}, \alpha_p\right] \end{aligned} \tag{17}$$

[1] To avoid notational clutter, we denote conditioned-upon variables and parameters by a dash. These variables can be inferred from context.

The Antoniak distribution is the distribution of the number of occupied tables if N_{kjp} customers are sent into a restaurant that follows the Chinese restaurant process with strength parameter α_p. Sampling from the Antoniak distribution is done by sampling N_{kjp} Bernoulli variables:

$$s^l_{kjp} \sim \text{Bernoulli}\left[\frac{\alpha_p}{\alpha_p + l - 1}\right] \qquad l = 1 \ldots N_{kjp} \tag{18}$$

$$s_{kjp} = \sum_l s^l_{kjp} \tag{19}$$

Using the same auxiliary variable techniques, we derive sampling equations for β and Φ. These variables are sampled jointly because they are dependent. The posterior distribution for β and Φ and the joint distribution with the auxiliary variables $\mathbf{t}$ and $\mathbf{s}$ are given by:

$$P(\beta_k, \Phi_k|\_) \propto \prod_p \left[\frac{\Gamma(\beta_k)}{\Gamma(N_{kp} + \beta_k)} \prod_w \frac{\Gamma(N_{wkp} + \beta_k \Phi_{w|k})}{\Gamma(\beta_k \Phi_{w|k})}\right] \left[\prod_w \Phi^{\gamma-1}_{w|k}\right] \beta_k^{a-1} e^{-b\beta_k} \tag{20}$$

$$\begin{aligned} P(\beta_k, \Phi_k, \mathbf{t}, \mathbf{s}|\_) \propto & \left[\prod_p t_{kp}^{\beta_k - 1}(1 - t_{kp})^{N_{kp}-1}\right] \left[\prod_p \prod_w S(N_{wkp}, s_{wkp})(\beta_k \Phi_{w|k})^{s_{wkp}}\right] \\ & \left[\prod_k \prod_w \Phi^{\gamma-1}_{w|k}\right] \beta_k^{a-1} e^{-b\beta_k} \end{aligned} \tag{21}$$

Note that the set of variables ($\mathbf{t}$ and $\mathbf{s}$) is unrelated to the set of auxiliary variables introduced for α_p. The sampling equations for β, Φ, $\mathbf{t}$, and $\mathbf{s}$ are:

$$\begin{aligned} P(\beta_k|\Phi, \mathbf{t}, \mathbf{s}, \_) & \propto \left[\prod_p t_{kp}^{\beta_k}\right] \left[\prod_p \prod_w (\beta_k)^{s_{wkp}}\right] \beta_k^{a-1} e^{-b\beta_k} \\ & = \text{Gamma}\left[a + \sum_p \sum_w s_{wkp}; b - \sum_p log(t_{kp})\right] \end{aligned} \tag{22}$$

$$\begin{aligned} P(\Phi_k|\beta_k, \mathbf{t}, \mathbf{s}, \_) & \propto \left[\prod_p \prod_w \Phi^{s_{wkp}}_{w|k}\right] \left[\prod_w \Phi^{\gamma-1}_{w|k}\right] \\ & = \text{Dirichlet}\left[\gamma + \sum_p s_{wkp}\right] \end{aligned} \tag{23}$$

$$\begin{aligned} P(t_{kp}|\beta_k, \Phi_k, \mathbf{s}, \_) & \propto t_{kp}^{\beta_k - 1}(1 - t_{kp})^{N_{kp}-1} \\ & = \text{Beta}\left[\beta_k, N_{kp}\right] \end{aligned} \tag{24}$$

$$\begin{aligned} P(s_{wkp}|\beta_k, \Phi_k, \mathbf{t}, \_) & \propto S(N_{wkp}, s_{wkp})(\beta_k \Phi_{w|k})^{s_{wkp}} \\ & = \text{Antoniak}\left[N_{wkp}, \beta_k \Phi_{w|k}\right] \end{aligned} \tag{25}$$

References

M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, 1964.

A. Asuncion and D. Newman. UCI machine learning repository, 2007. URL `http://www.ics.uci.edu/∼mlearn/MLRepository.html`.

D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. In *Advances in Neural Information Processing Systems*, volume 14, pages 601–608, Cambridge, MA, 2002. MIT Press.

D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

A. Brockwell. Parallel Markov chain Monte Carlo simulation by pre-fetching. *Journal of Computational & Graphical Statistics*, 15, No. 1:246–261, 2006.

R. Burkard and E. Çela. Linear assignment problems and extensions. In P. Pardalos and D. Du, editors, *Handbook of Combinatorial Optimization, Supplement Volume A*. Kluwer Academic Publishers, 1999.

G. Casella and C. Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 1996.

C. Chemudugunta, P. Smyth, and M. Steyvers. Modeling general and specific aspects of documents with a probabilistic topic model. In *Advances in Neural Information Processing Systems 19*, pages 241–248. MIT Press, Cambridge, MA, 2007.

C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun. Map-Reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 19*, pages 281–288. MIT Press, Cambridge, MA, 2007.

A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 271–280, New York, NY, 2007. ACM.

M. Escobar and M. West. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90(430):577–588, 1995.

G. Forman and B. Zhang. Distributed data clustering can be efficient and exact. In *ACM KDD Explorations*, volume 2, pages 34–38, New York, NY, 2000. ACM.

T. Griffiths and M. Steyvers. Finding scientific topics. In *Proceedings of the National Academy of Sciences*, volume 101, pages 5228–5235, 2004.

E. Kontoghiorghes. *Handbook of Parallel Computing and Statistics (Statistics, Textbooks and Monographs)*. Chapman & Hall / CRC, 2005.

W. Li and A. McCallum. Pachinko allocation: DAG-structured mixture models of topic correlations. In *Proceedings of the International Conference on Machine Learning*, volume 23, pages 577–584, New York, NY, 2006. ACM.

J. Liu, W. Wong, and A. Kong. Covariance structure of the Gibbs sampler with applications to the comparisons of estimators and augmentation schemes. *Biometrika*, 81(1):27–40, 1994.

D. Mimno and A. McCallum. Organizing the OCA: Learning faceted subjects from a library of digital books. In *JCDL '07: Proceedings of the 2007 conference on digital libraries*, pages 376–385, New York, NY, 2007. ACM.

R. Nallapati, W. Cohen, and J. Lafferty. Parallelized variational EM for latent Dirichlet allocation: An experimental evaluation of speed and scalability. In *ICDMW '07: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, pages 349–354, Washington, DC, 2007. IEEE Computer Society.

P. Ferrari, A. Frigessi, and R. Schonmann. Convergence of some partially parallel Gibbs samplers with annealing. In *Annals of Applied Probability*, volume 3, pages 137–153. 1993.

M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth. The author-topic model for authors and documents. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, volume 20, pages 487–494, Arlington, VA, 2004. AUAI Press.

A. Rossini, L. Tierney, and N. Li. Simple parallel statistical computing in R. *Journal of Computational & Graphical Statistics*, 16(2):399, 2007.

Y. W. Teh, M. Jordan, M. Beal, and D. Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

W. Kowalczyk and N. Vlassis. Newscast EM. In *Advances in Neural Information Processing Systems 17*, pages 713–720. MIT Press, Cambridge, MA, 2005.

J. Wolfe, A. Haghighi, and D. Klein. Fully distributed EM for very large datasets. In *Proceedings of the International Conference on Machine Learning*, pages 1184–1191. ACM, New York, NY, 2008.

L. Younes. Synchronous random fields and image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4):380–390, 1998.