Scalable Training of Hierarchical Topic Models

Jianfei Chen[†], Jun Zhu[†][*], Jie Lu[‡], and Shixia Liu[‡]
[†]Dept. of Comp. Sci. & Tech., BNRist Center, State Key Lab for Intell. Tech. & Sys.
[‡]School of Software, BNRist Center, State Key Lab for Intell. Tech. & Sys.
Tsinghua University, Beijing, 100084, China
{chenjian14,luj15}@mails.tsinghua.edu.cn; {dcszj,shixia}@tsinghua.edu.cn

ABSTRACT

Large-scale topic models serve as basic tools for feature extraction and dimensionality reduction in many practical applications. As a natural extension of flat topic models, hierarchical topic models (HTMs) are able to learn topics of different levels of abstraction, which lead to deeper understanding and better generalization than their flat counterparts. However, existing scalable systems for flat topic models cannot handle HTMs, due to their complicated data structures such as trees and concurrent dynamically growing matrices, as well as their susceptibility to local optima.

In this paper, we study the hierarchical latent Dirichlet allocation (hLDA) model which is a powerful nonparametric Bayesian HTM. We propose an efficient partially collapsed Gibbs sampling algorithm for hLDA, as well as an initialization strategy to deal with local optima introduced by tree-structured models. We also identify new system challenges in building scalable systems for HTMs, and propose efficient data layout for vectorizing HTM as well as distributed data structures including dynamic matrices and trees. Empirical studies show that our system is 87 times more efficient than the previous open-source implementation for hLDA, and can scale to thousands of CPU cores. We demonstrate our scalability on a 131-million-document corpus with 28 billion tokens, which is 4-5 orders of magnitude larger than previously used corpus. Our distributed implementation can extract 1,722 topics from the corpus with 50 machines in just 7 hours.

PVLDB Reference Format:
Jianfei Chen, Jun Zhu, Jie Lu and Shixia Liu. Scalable Training of Hierarchical Topic Models. *PVLDB*, 11(7): 826-839, 2018.
DOI: https://doi.org/10.14778/3192965.3192972

1. INTRODUCTION

Topic models are popular machine learning tools that extract a set of latent topics from an input text corpus. Each topic is a unigram distribution over words, and the high-probability words often present strong semantic correlation. As a method for dimensionality reduction and feature extraction, topic models have been widely used in information retrieval [35], text analysis [7, 41], information visualization [33], online advertising [34], recommendation system [31], and many other application areas.

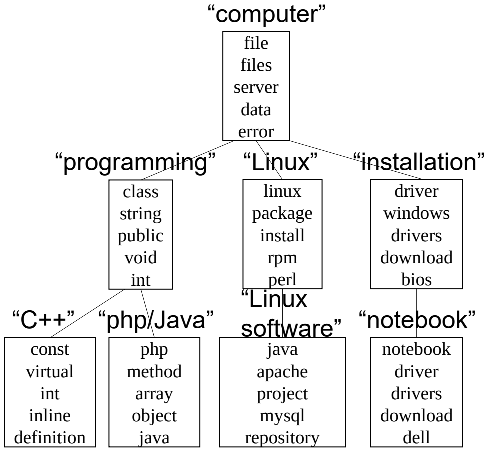


Figure 1: A subtree of the topic hierarchy learned from a corpus with 131-million web pages (See experiments for details). Each node is a topic with its top-5 most frequent words. The full tree has 1,722 topics. The text in the quote is summarized by humans.

Practical applications call for scalable and efficient topic modeling systems to handle big data. For example, online advertisement systems extract topics from billions of search queries [34], and recommendation systems [1] need to handle millions of users and items. Various efforts has been made to develop scalable topic modeling systems, including asynchronous distributed data parallel training [1, 17], hybrid data-and-model-parallel training [37, 36], embarrassingly parallel BSP training [10, 38, 39], and GPU-accelerated training [40, 16]. These topic modeling systems mainly handle the partition of the data and model and the synchronization of the count matrix across machines.

These works are for flat topic models, such as latent Dirichlet allocation (LDA) [6], which assumes that all the topics have the same level of abstraction. In reality, however, topics are naturally organized in a hierarchy [21]. See Fig. 1 as an example (a more complete version is in experiments). When

[*] Corresponding author.

Proceedings of the VLDB Endowment, Vol. 11, No. 7
Copyright 2018 VLDB Endowment 2150-8097/18/03... $ 10.00.
DOI: https://doi.org/10.14778/3192965.3192972

a topic about "programming" is observed in a document, we are also likely to observe the more general topic such as "computer" in the same document. By capturing such relationships, hierarchical topic models can achieve deeper understanding and better generalization [2, 21] of the corpus than the flat models. There are many approaches to learn a topic hierarchy. For example, Google's Rephil [19] puts a hierarchical noisy-or network on the documents; the super-topic approach learns topics of topics [18, 23]; and the nested Chinese Restaurant Process (nCRP) [5, 2, 21] approach utilizes the nCRP as a prior on topic hierarchies, with hierarchical latent Dirichlet allocation (hLDA) [5] as a popular example. These models have been successfully applied to document modeling [21], online advertising [19] and microblog location prediction [2], outperforming flat models.

Despite the success of hierarchical topic models (HTMs) on small scale corpora, they have not been deployed on practical applications due to the lack of scalable systems as well as efficient algorithms. Developing a scalable system for hierarchical topic models is challenging because of the complicated data structures, such as trees and dynamically growing matrices (e.g., in hLDA). We need to develop distributed versions of these data structures while maintaining the efficiency and consistency. Moreover, the access patterns of these data structures are more complicated than the count matrix in flat topic models, making efficient vectorization more difficult. There are also algorithmic challenges, e.g., dealing with the local optima during the learning.

In this paper, we make an initial attempt in scaling up hierarchical topic models. We study the hierarchical latent Dirichlet allocation (hLDA) [5] model, which can automatically learn the hierarchical topical structure with Gibbs sampling [14] by utilizing an nCRP prior. In hLDA, topics form a tree with an nCRP prior, while each document is assigned with a path from the root topic to a leaf topic, and words in the document are modeled with an admixture of topics on the path. We propose a novel partially collapsed Gibbs sampling (PCGS) algorithm that is more efficient than the original collapsed Gibbs sampling (CGS) algorithm [5]. To improve the model quality, we analyze the cause of local optima and propose an effective initialization strategy. On the system side, we propose a data layout for efficient vectorization, and design a concurrent dynamic matrix as well as a distributed tree to support hLDA's complicated data structures for distributed training.

We extensively evaluate the model quality, efficiency and scalability of our algorithm and system. The experimental results show that our PCGS algorithm and the vectorization strategy lead to an 87 times single-thread speedup than the existing open source implementation `hlda-c` [4], while having better model quality due to our initialization strategy. Multi-threading and distributed computing further speed up our system, and we analyze the speedup and impact to model quality as the amount of computational resource grows. Finally, we demonstrate the scalablity of our distributed system by extracting 1,722 topics from a 131-million-document corpus with 50 machines in 7 hours, where the corpus is 4-5 orders of magnitude larger than the largest corpus used for hLDA before. We believe that our algorithm and system can be extended to more hierarchical topic models, such as the nested hierarchial Dirichlet process (nHDP) [21] model.

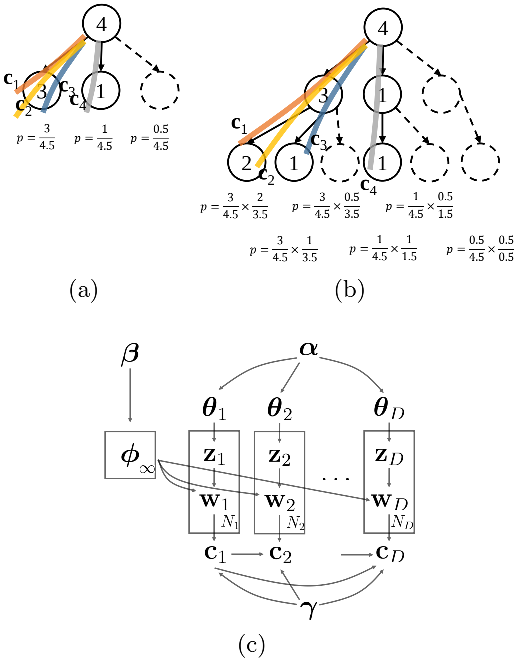


Figure 2: The (a) CRP (b) nCRP distribution for $p(\mathbf{c}_5|\mathbf{c}_1, \dots, \mathbf{c}_4)$ given previous 4 paths, with $\gamma_1 = \gamma_2 = 0.5$. Numbers on the nodes are visited counts m_t, and dashed nodes are unvisited. p is the probability for each path. (c) Graphical model for hLDA.

2. HIERARCHICAL LDA

We first review nCRP and hLDA for learning a topic hierarchy as well as the collapsed Gibbs sampling inference algorithm proposed by Blei et al. [5].

2.1 Nested Chinese Restaurant Process

Nested Chinese Restaurant process (nCRP) [5] represents a powerful nonparametric Bayesian method to learn a tree structure, whose width and depth are unbounded. Following previous work in hierarchical topic modeling [5], we present the nCRP with (finite) L-levels, and the infinite-depth tree can be approximated by setting L to a large number.

Suppose there is a tree with L levels, where each node except the leaves has infinite children. A unique ID is assigned to each node, where the root node has the ID 1. nCRP defines a probability distribution on a series of paths $(\mathbf{c}_1, \mathbf{c}_2, \dots)$ on the tree, where each path $\mathbf{c}_d \in \mathbb{N}_+^L$ consists of L node IDs from the root to a certain leaf. nCRP defines the joint distribution of paths as

$$p(\mathbf{c}_1, \mathbf{c}_2, \dots) = \prod_{d=1}^{\infty} p(\mathbf{c}_d|\mathbf{c}_{<d}),$$

where the subscript $< d$ stands for all the possible indices that are smaller than d, i.e., $\mathbf{c}_{<d} = \{\mathbf{c}_1, \dots, \mathbf{c}_{d-1}\}$. Given previous paths $\mathbf{c}_{<d}$, nCRP defines a probability distribution $p(\mathbf{c}_d|\mathbf{c}_{<d})$ on the next path as

$$p(\mathbf{c}_d|\mathbf{c}_{<d}) = \prod_{l=2}^{L} p(c_{dl}|c_{d,l-1}, \mathbf{c}_{<d}),$$

which can be interpreted as a generative story: a person starts from the root node 1 at level 1. Each time he walks to a child of the current node until he reaches a leaf node, where the probability of going from $c_{d,l-1}$ to c_{dl} is $p(c_{dl}|c_{d,l-1}, \mathbf{c}_{<d})$. The probability of the path he visited is then $p(\mathbf{c}_d|\mathbf{c}_{<d})$.

The transition probability $p(c_{dl}|c_{d,l-1}, \mathbf{c}_{<d})$ follows the Chinese Restaurant Process (CRP) [27]. We denote a *visit* of node t, if any previous path $\mathbf{c}_{d'}(d' < d)$ passes through it. The number of visits $m_t := \#\{(d', l)|c_{d'} = t, d' < d\}$, where $\#\{\cdot\}$ denotes the cardinality of a set. We assume that the paths $\mathbf{c}_{<d}$ have already visited T nodes in the infinite tree, with the node ID from 1 to T, and the current node $c_{d,l-1}$ has K visited children with the IDs $t_1, \ldots, t_K$. This corresponds to a lazy strategy of assigning IDs to nodes: a node does not have an ID until it is visited. In CRP, the probability of going to each child is proportional to its previous times of visit

$$\begin{cases} p(c_{dl} = t_i|\cdot) = \frac{m_{t_i}}{\gamma_l + m_{c_{d,l-1}}}, & k = 1, \ldots, K_i \\ p(c_{dl} = T+1|\cdot) = \frac{\gamma_l}{\gamma_l + m_{c_{d,l-1}}}, & \end{cases} \tag{1}$$

where γ_l is a hyper-parameter controlling the probability of going to an unvisited child. There are infinite unvisited children and we may choose any of them to visit, but we always assign the ID $T+1$ to the chosen one. This operation of assigning an ID to an unvisited node is referred as *the generation of a new child* in this paper.

Given $\mathbf{c}_{<d}$, the number of different possible paths for $\mathbf{c}_d$ is T, which is finite. There are T_{leaf} paths ending at existing leaves, and $T - T_{leaf}$ paths forking from an internal visited node. Fig. 2(a) and Fig. 2(b) illustrate the possible paths and their probabilities for CRP and nCRP, respectively.

2.2 Hierarchical Latent Dirichlet Allocation

hLDA is an nCRP-based topic model to learn a topic hierarchy [5] from a corpus of D bag-of-words documents $\mathbf{W} = \{\mathbf{w}_d\}_{d=1}^{D}$, where each document $\mathbf{w}_d = \{w_{dn}\}_{n=1}^{N_d}$ has N_d tokens, and each token is represented by its word ID $w_{dn} \in \{1, \ldots, V\}$ in the vocabulary of V unique words. In hLDA, topics form an L-level tree, i.e., each tree node t is a topic, and is associated with a distribution over words $\boldsymbol{\phi}_t \in \Delta^{V-1}$, where Δ^{V-1} is the $(V-1)$-simplex. Since nodes and topics have one-to-one correspondence, we do not distinguish them in the sequel.

The hLDA model assumes that each document is assigned with a path $\mathbf{c}_d$ following the nCRP. The words of each document are modeled by a mixture of the topics in $\mathbf{c}_d$, with the document-specific mixing proportion $\boldsymbol{\theta}_d$. The generative process for the corpus is:

- For each node t, draw $\boldsymbol{\phi}_t \sim \mathrm{Dir}(\beta_{l_t}\mathbf{1})$, where l_t is the level of node t and $\mathbf{1} = (1, \ldots, 1)$ is an all-one vector;
- For each document d:
  - Draw a path of topics $\mathbf{c}_d \sim p(\mathbf{c}_d|\mathbf{c}_{<d})$ according to nCRP (Eq. 1);
  - Draw the topic mixing proportion $\boldsymbol{\theta}_d \sim \mathrm{Dir}(\boldsymbol{\alpha})$;
  - For each position n, choose a node from the path $\mathbf{c}_d$ by drawing the level assignment $z_{dn} \sim \mathrm{Mult}(\boldsymbol{\theta}_d)$. Then, draw a word from the chosen topic $w_{dn} \sim \mathrm{Mult}(\boldsymbol{\phi}_{c_{d,z_{dn}}})$,

where $\mathrm{Dir}(\cdot)$ is the Dirichlet distribution, and $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are Dirichlet hyper-parameters. See Fig. 2(c) for the probabilistic graphical model representation of hLDA.

There are two special cases of hLDA. When the tree degenerates to a chain, hLDA recovers the vanilla latent Dirichlet allocation (LDA) [6] with L topics, and when the tree has two levels and the probability of assigning to the first level θ_{d1} is close to zero, hLDA recovers the Dirichlet Process Mixture Model (DPMM) [20].

2.3 Notation of Counts

Before introducing the inference algorithm for hLDA, we need to define some notations of counts. We will encounter four types of counts. The first is the node visit count m_t defined in Sec. 2.1, which represents how many documents that visit node t. The remaining three counts are token counts. Each token in the corpus has four attributes: its document ID d, word ID w_{dn}, level z_{dn}, and the node ID $c_{d,z_{dn}}$ it is assigned to. The following counts represent the number of tokens that have specific attributes

- The document-level count $a_{dl} = \#\{n|z_{dn} = l\}$ is the number of tokens on level l in document d.
- The topic-word count $b_{tv} = \#\{(d, n)|w_{dn} = v \wedge c_{d,z_{dn}} = t\}$ is the number of tokens of the word v with topic t.
- The topic count $s_t = \sum_v b_{tv}$ is the number of tokens with topic t.

Furthermore, bold uppercase letters denote the matrix formed by the counts, e.g., $\mathbf{A}$ is a $D \times L$ matrix composed by a_{dn}'s. Let $\mathbf{a}_d$ denote the d-th row vector, $\mathbf{a}_{:,l}$ be the l-th column vector. Let $\mathbf{m}$ and $\mathbf{s}$ be T-dimensional vectors composed of m_t and s_t, respectively. Finally, d stands for document d, $\neg d$ represents all documents excluding document d, and $\neg dn$ means all the tokens excluding token (d, n). Using these as the superscript of counts means only counting the objects among the given documents / tokens. For instance, $b_{tv}^{d} = \#\{n|w_{dn} = v \wedge c_{d,z_{dn}} = t\}$ and $b_{tv}^{\neg d} = \#\{(d', n)|w_{d'n} = v \wedge c_{d',z_{d'n}} = t \wedge d' \neq d\}$.

2.4 Collapsed Gibbs Sampling (CGS)

The generative process of hLDA (Sec. 2.2) defines a joint distribution

$$p(\mathbf{W}, \mathbf{z}, \boldsymbol{\theta}, \mathbf{c}, \boldsymbol{\phi}) = \prod_{t=1}^{\infty} p(\boldsymbol{\phi}_t) \prod_{d=1}^{D} p(\mathbf{c}_d|\mathbf{c}_{<d}) p(\boldsymbol{\theta}_d) \prod_{d=1}^{D} \prod_{n=1}^{N_d} p(z_{dn}|\boldsymbol{\theta}_d) p(w_{dn}|\boldsymbol{\phi}_{c_{d,z_{dn}}}),$$

where $\mathbf{z} = \{\mathbf{z}_d\}_{d=1}^{D}$ and $\mathbf{c} = \{\mathbf{c}_d\}_{d=1}^{D}$. Given the training corpus $\mathbf{W}$, we want to *infer* the posterior distribution of the underlying topics $\boldsymbol{\phi}_t$ as well as the per-document latent variables, including the paths $\mathbf{c}_d$, mixing proportions $\boldsymbol{\theta}_d$ and layer assignments $\mathbf{z}_d$. We do not have closed-form solutions for the posterior distribution $p(\mathbf{z}, \boldsymbol{\theta}, \mathbf{c}, \boldsymbol{\phi}|\mathbf{W})$ [5]. Instead we approximate the inference by drawing samples from it, where Collapsed Gibbs sampling (CGS) [5] is one possible sampler.

Based on the conjugacy between Dirichlet and multinomial distributions, CGS integrates out $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to get the collapsed posterior distribution:

$$p(\mathbf{W}, \mathbf{z}, \mathbf{c}) = \prod_{d=1}^{D} \left[p(\mathbf{c}_d|\mathbf{c}_{<d}) \frac{B(\mathbf{a}_d + \boldsymbol{\alpha})}{B(\boldsymbol{\alpha})}\right] \prod_{t=1}^{\infty} \frac{B(\mathbf{b}_t + \beta_{l_t})}{B(\beta_{l_t}\mathbf{1})}, \tag{2}$$

where $\Gamma(\cdot)$ is the gamma function, and $B(\cdot)$ is the multivariate beta function, $B(\boldsymbol{\alpha}) = \prod_k \Gamma(\alpha_k)/\Gamma(\sum_k \alpha_k)$.

CGS samples from the collapsed posterior distribution in Eq. (2) by Gibbs sampling [14], i.e., alternatively sampling $\mathbf{z}$ and $\mathbf{c}$ from their conditional distributions:

$$p(z_{dn} = l|w_{dn} = v, \mathbf{W}, \mathbf{z}_{\neg dn}, \mathbf{c}) \propto (a_{dl}^{\neg dn} + \alpha_l) \frac{b_{c_{dl},v}^{\neg dn} + \beta_l}{s_{c_{dl}}^{\neg dn} + V\beta_l},$$

$$p(\mathbf{c}_d = \mathbf{c}|\mathbf{W}, \mathbf{z}, \mathbf{c}_{\neg d}) \propto p(\mathbf{c}|\mathbf{c}_{\neg d}) \prod_{l=1}^{L} f_C(d, c_l),$$

where

$$f_C(d, t) = \frac{B(\mathbf{b}_t^{\neg d} + \mathbf{b}_t^d + \beta_{l_t})}{B(\mathbf{b}_t^{\neg d} + \beta_{l_t})}. \tag{3}$$

3. EFFICIENT TRAINING FOR HLDA

The existing algorithms for hLDA (e.g., CGS) are not efficient, which have limited the evaluation at a small-scale and made hLDA not applicable in practice. To scale up hLDA, there are two main algorithmic challenges:

- Efficiency. The main computational burden for CGS, Eq. (3), needs lots of evaluation of the logarithms (as we will show in Sec. 3.2), which is expensive.
- Local optimum. The posterior distribution $p(\mathbf{z}, \mathbf{c}|\mathbf{W})$ is multi-modal, so the sampler can be trapped around a local optimum, resulting in bad model quality.

To address these challenges, we propose a partially collapsed Gibbs sampler (PCGS) with an effective initialization strategy.

3.1 Partially Collapsed Gibbs Sampling

We propose a partially collapsed Gibbs sampling (PCGS) algorithm to avoid computing most of the logarithms, while preserving the quality of inference.

We pick a subset $\mathcal{I}$ of the visited topic IDs $\{1, \ldots, T\}$. Unlike CGS which collapses out all the topic distributions $\boldsymbol{\phi}_t$, PCGS only collapses out the topics that do not belong to $\mathcal{I}$, while leaving the topics in $\mathcal{I}$ instantiated. Let $\bar{\mathcal{I}} = \mathbb{N}_+ \backslash \mathcal{I}$ be the complement of $\mathcal{I}$. We allow matrices to use sets as subscripts, e.g., $\boldsymbol{\phi}_{\mathcal{I}}$. In this case, $\boldsymbol{\phi}_{\mathcal{I}}$ is a matrix composed of all the row vectors indexed by $t \in \mathcal{I}$. PCGS samples from the marginal distribution

$$\begin{aligned} p(\mathbf{W}, \mathbf{z}, \mathbf{c}, \boldsymbol{\phi}_{\mathcal{I}}) &= \int_{\boldsymbol{\theta}, \boldsymbol{\phi}_{\bar{\mathcal{I}}}} p(\mathbf{W}, \mathbf{z}, \boldsymbol{\theta}, \mathbf{c}, \boldsymbol{\phi}) d\boldsymbol{\theta} d\boldsymbol{\phi}_{\bar{\mathcal{I}}} \\ &= \prod_{d=1}^{D} \left[p(\mathbf{c}_d|\mathbf{c}_{<d}) \frac{B(\mathbf{a}_d + \boldsymbol{\alpha})}{B(\boldsymbol{\alpha})}\right] \prod_{t \in \bar{\mathcal{I}}} \frac{B(\mathbf{b}_t + \beta_{l_t})}{B(\beta_{l_t})} \\ &\prod_{t \in \mathcal{I}} \left[\mathrm{Dir}(\boldsymbol{\phi}_t; \beta_{l_t}) \prod_{v=1}^{V} (\phi_{tv})^{b_{tv}}\right]. \end{aligned}$$

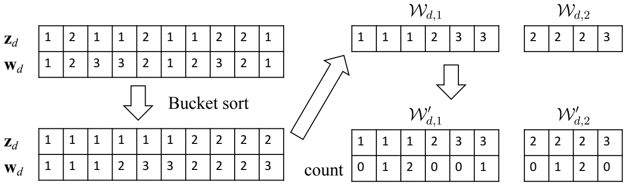


Figure 3: Illustration of $\mathcal{W}_d$ and $\mathcal{W}'_d$

Utilizing the identity $\frac{B(\boldsymbol{\alpha}+\mathbf{e}_k)}{B(\boldsymbol{\alpha})} = \frac{\alpha_k}{\sum_k \alpha_k}$, where $\mathbf{e}_k$ is a coordinate vector, we can derive the Gibbs sampling updates (see the appendix for derivation details):

Sample z: draw the level assignment for each token from

$$p(z_{dn} = l|w_{dn} = v, \mathbf{W}, \mathbf{z}_{\neg dn}, \mathbf{c}, \boldsymbol{\phi}_{\mathcal{I}}) \propto (a_{dl}^{\neg dn} + \alpha_l) \begin{cases} \phi_{c_{dl},v} & c_{dl} \in \mathcal{I}, \\ \frac{b_{c_{dl},v}^{\neg dn} + \beta_{l_t}}{s_{c_{dl}}^{\neg dn} + V\beta_{l_t}} & c_{dl} \in \bar{\mathcal{I}}. \end{cases} \tag{4}$$

Sample c: sample the path from

$$p(\mathbf{c}_d = \mathbf{c}|\mathbf{W}, \mathbf{z}, \mathbf{c}_{\neg d}, \boldsymbol{\phi}_{\mathcal{I}}) \propto p(\mathbf{c}|\mathbf{c}_{\neg d}) \prod_{l=1}^{L} \begin{cases} f_I(d, c_l) & c_l \in \mathcal{I}, \\ f_C(d, c_l) & c_l \in \bar{\mathcal{I}}, \end{cases} \tag{5}$$

where f_C is defined as Eq. (3), and

$$f_I(d, t) = \prod_{v=1}^{V} (\phi_{tv})^{b_{tv}^d}. \tag{6}$$

Sample $\boldsymbol{\phi}_{\mathcal{I}}$: For $t \in \mathcal{I}$, draw $\boldsymbol{\phi}_t \sim \mathrm{Dir}(\beta_{l_t} + \mathbf{b}_t)$. To speed up the mixing, we replace the sampling with expectation in our implementation, i.e.,

$$\boldsymbol{\phi}_t \leftarrow \frac{\beta_{l_t} + \mathbf{b}_t}{V\beta_{l_t} + s_t}. \tag{7}$$

3.2 Time complexity analysis

We now discuss some implementation details of sampling $\mathbf{c}$, and then we compare the time complexity of CGS and PCGS. To sample $\mathbf{c}$, we need to enumerate all the possible T paths and compute their nCRP probability $p(\mathbf{c}|\mathbf{c}_{\neg d})$ as described in Sec. 2.1. Then we compute $f_I(d, t)$ for all $t \in \mathcal{I}$ and $f_C(d, t)$ for all $t \in \bar{\mathcal{I}}$. Finally, we compute the posterior probability $p(\mathbf{c}_d = \mathbf{c}|\mathbf{W}, \mathbf{z}, \mathbf{c}_{\neg d}, \boldsymbol{\phi}_{\mathcal{I}})$ for all the possible T paths, and sample a path.

Because both $f_I(d, t)$ and $f_C(d, t)$ are very close to zero, we compute their logarithms. Rewrite Eq. (6):

$$\log f_I(d, t) = \log \prod_{v=1}^{V} (\phi_{tv})^{b_{tv}^d} = \sum_{v \in \mathcal{W}_{d,l_t}} \log \phi_{tv}, \tag{8}$$

where l_t is the level of node t, and $\mathcal{W}_{dl} = \{w_{dn}|z_{dn} = l\}$ is the multiset of all tokens in document d that are assigned to level l, which can be computed by bucket sorting the (z_{dn}, w_{dn}) pairs.

Similarly, Eq. (3) can be rewritten as:

$$\log f_C(d, t) = \sum_{(v,o) \in \mathcal{W}'_{d,l_t}} \log(b_{tv}^{\neg d} + o + \beta_{l_t}) + h_t, \tag{9}$$

where we convert the logarithm of multivariate beta function as the sum of logarithms (the derivation details can be

Alg.	Step	Time Complexity	Operation
Both	Sample $\mathbf{z}$	$O(NL)$	Division
CGS	Sample $\mathbf{c}$	$O(NT)$	Logarithm
PCGS	Sample $\mathbf{c}$	$\epsilon \times O(NT)$	Logarithm
	Sample $\mathbf{c}$	$(1-\epsilon) \times O(NT)$	Addition
	Sample $\phi_{\mathcal{I}}$	$O(TV)$	Division

Table 1: Time complexity analysis of CGS and PCGS. N: corpus size, typically $10^6 - 10^{10}$; T: number of topics, typically $10^2 - 10^3$; L: number of layers, typically 10^1; V: vocabulary size, typically $10^4 - 10^6$; $\epsilon = 0.05$: proportion of collapsed topics.

found in the appendix). The term $h_t = \log\Gamma(s_t^{\neg d} + V\beta_{l_t}) - \log\Gamma(s_t + V\beta_{l_t})$. The set $\mathcal{W}'_{dl}$ is similar as $\mathcal{W}_{dl}$, but we assign each token with a count indicating which time does this word appear, e.g., if a word v is in $\mathcal{W}_{dl}$ for three times, we put $(v, 0)$, $(v, 1)$ and $(v, 2)$ into $\mathcal{W}'_{dl}$. See Fig. 3 as an illustration.

Theoretically, both CGS and PCGS can perform correct inference since they both converge to the true posterior distribution. We also present some intuition on why PCGS works: $b_{tv}^{\neg d} + o + \beta_{l_t}$ in f_C is a more fresh version of ϕ_{tv} in f_I, in the sense that $b_{tv}^{\neg d}$ is updated instantly after each sampling operation, but ϕ_{tv} is only updated once per iteration. However, if $b_{tv}^{\neg d}$ is large, it will not change much during the iteration, so PCGS should perform similarly with CGS. Based on this intuition, we choose $\mathcal{I}$ to be the nodes t with top 95% s_t, whose corresponding $b_{tv}^{\neg d}$ should be large.

We now analyze the time complexity. By analyzing Eq. (8) and Eq. (9) we know that the time complexity of computing both $f_I(d, t)$ and $f_C(d, t)$ for every document d and topic t is at most $O(NT)$, where $N = \sum_{d=1}^{D} N_d$ is the size of the corpus. Despite having the same time complexity, computing f_I is much more efficient than computing f_C. Because $\log\phi_{tv}$ only changes once per iteration according to Eq. (7), it can be pre-processed. Hence, computing f_I only involves NT floating-point additions. In contrast, the logarithm cannot be avoided during computing f_C because the count $b_{tv}^{\neg d}$ is changing all the time. Since PCGS computes f_C for only 5% of the topics, it is much more cheaper than CGS which computes f_I for 100% of the topics.

Sampling $\mathbf{z}$ requires enumerating the level for every token, so its time complexity is $O(NL)$, which is much cheaper than $O(NT)$ of sampling $\mathbf{c}$ because the number of levels L is much smaller than the number of topics T. Sampling $\boldsymbol{\phi}_{\mathcal{I}}$ is $O(TV)$, which is again much smaller than $O(NT)$ because the vocabulary size V, i.e., the number of unique tokens, is much smaller than the total number of tokens N. Therefore, we show that computing f_C and f_I is indeed the main computational burden, and PCGS is much cheaper than CGS because floating-point additions are much cheaper than logarithms. Table 1 summarizes the time complexity of CGS and PCGS.

3.3 Initialization Strategy

The tree structure of hLDA introduces more local optima than flat topic models. As in Sec. 2.2, the topic assignment for a particular token w_{dn} is defined by a two-step allocation: one first chooses a path $\mathbf{c}_d$ from the infinite tree and then chooses one topic $c_{d,z_{dn}}$ from the path. If we want to change the topic assignment, we need to change $\mathbf{c}_d$ and z_{dn} simultaneously. More concretely, after a document d is assigned to a certain path $\mathbf{c}_d$, its words are assigned to the levels $\mathbf{z}_d$ *of the current path.* In the next iteration, even if there is another path $\mathbf{c}'_d$ such that $p(\mathbf{w}_d|\mathbf{c}'_d)$ is larger than $p(\mathbf{w}_d|\mathbf{c}_d)$, $p(\mathbf{w}_d|\mathbf{c}'_d, \mathbf{z}_d)$ is not likely to be larger than $p(\mathbf{w}_d|\mathbf{c}_d, \mathbf{z}_d)$ because $\mathbf{z}_d$ is already optimized for $\mathbf{c}_d$. In this case, the path assignments $\mathbf{c}_d$ will be quickly trapped in a local optimum even if there are better path. We also noticed that similar as multinomial mixture models [25], the sampling of $\mathbf{c}_d$ is almost deterministic, because $p(\mathbf{w}_d|\mathbf{c}_d, \mathbf{z}_d)$ can differ by hundreds orders of magnitudes for different $\mathbf{c}_d$'s. Therefore, it is difficult for a sampler to jump out of the local trap simply by its randomness. In contrast, flat models such as LDA do not have this type of local optima because their topic assignments are directly chosen from all the T topics.

Since hLDA is sensitive to local optima, a proper initialization is crucial to get good results. We obtain a good initialization by sampling $\mathbf{c}$ from $p(\mathbf{c}|\mathbf{W})$ directly instead of from $p(\mathbf{c}|\mathbf{z}, \mathbf{W})$ (Eq. 5) for the first I iterations. In other words, we integrate $\mathbf{z}$ out. In the first I iterations, the sampler focuses on finding the optimal assignment $\mathbf{c}$ for each document. Afterwards, the algorithm samples $p(\mathbf{c}|\mathbf{z}, \mathbf{W})$ to refine the model.

The marginal distribution $p(\mathbf{c}|\mathbf{W}) = \sum_{\mathbf{z}} p(\mathbf{c}|\mathbf{z}, \mathbf{W})p(\mathbf{z})$ has no closed-form representation. We approximate it with Monte-Carlo integration $p(\mathbf{c}|\mathbf{W}) \approx \frac{1}{S}\sum_{\mathbf{z}_s \sim p(\mathbf{z})} p(\mathbf{c}|\mathbf{z}_s, \mathbf{W})$, where S is the number of samples, and $p(\mathbf{z}) = \int p(\mathbf{z}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$ is a Polya distribution which is approximated with a uniform discrete distribution over levels.

We also adopt the progressive online initialization strategy [5, 30], which starts with an empty collection of documents, and gradually adds documents by inferring the posterior of document-specific variables $(\mathbf{c}_d, \mathbf{z}_d)$ given all the previously observed documents. The documents are organized into mini-batches, and $\boldsymbol{\phi}_{\mathcal{I}}$ is sampled per mini-batch.

4. SYSTEM IMPLEMENTATION

The tree structured model poses challenges on developing the distributed system for training hLDA:

1. Efficient vectorization is difficult due to the complicated tree-related data structures.

2. The distributed count matrix has a dynamic number of columns (topics).

3. The model needs a distributed tree which supports adding and removing nodes as well as editing the counts.

4. Complicated consistency requirements for the tree and the matrix. For example, we need to avoid adding children to a non-existent node of the tree.

Previous flat topic modeling systems [10, 38, 17] that focus on the data / model partition and count matrix synchronization do not solve the aforementioned challenges of hierarchical topic models. In this section, we propose a distributed system for training hLDA. Our system follows the data-parallel paradigm, where the documents are distributed across machines and the model is shared. Our system consists of machine-level and thread-level parallelism, as shown in Fig. 4. On the machine level, we update the topic distribution $\boldsymbol{\phi}$ in a bulk-synchronous-parallel (BSP) fashion and update the tree structure and the counts $(\mathbf{B}, \mathbf{m})$

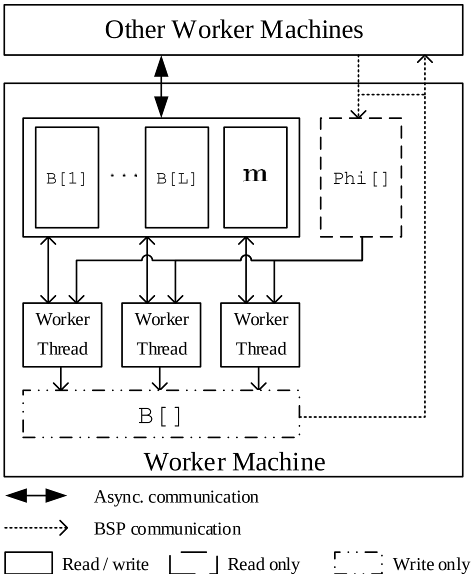


Figure 4: An overview of the distributed inference system.

asynchronously; On the thread-level, a number of threads concurrently read and update the local counts.

4.1 Vectorization and Data Layout

We first introduce the design of our data layout for efficient vectorization. As we already analyzed in Sec. 3.2, the most time-consuming part is the computation of f_I and f_C for every t according to Eq. (8) and Eq. (9). We can see that both f_I and f_C have similar access patterns. For each document d and topic t, they read the tokens v in the multiset $\mathcal{W}_{d,l_t}$, and access the corresponding topic-word entry (t, v) of the topic distribution matrix $\boldsymbol{\phi}$ or the count matrix $\mathbf{B}$. Hence, we only discuss the vectorization of f_I. The computation of $f_I(d,t)$ for a single topic t cannot be vectorized because the word IDs in $\mathcal{W}_{d,l_t}$ is discontinuous and varies by document. However, we can compute $\log f_I(d,t)$ for all the t's in one level altogether. For vectorization, we store a matrix `Phi[l]` for each level l, whose v-th row `Phi[l][v]` is the $\log \phi_{tv}$ for all the topics t of level l. In this way, we can compute all the $\log f_I(d,t)$ for level l by vector summation: summing up all the rows `Phi[l][v]` for $v \in \mathcal{W}_{d,l}$.

Fig. 5 illustrates the data layout. For each node t, we assign a *rank* r_t to it. On level l, the I_l nodes in $\mathcal{I}$ are assigned with the rank from 1 to I_l, and the other nodes have ranks from $I_l + 1$. We store $\log \phi_{tv}$ at Phi[$\mathtt{l_t}$][$\mathtt{v}, \mathtt{r_t}$] and $b_{tv}^{\neg dn}$ at B[$\mathtt{l_t}$][$\mathtt{v}, \mathtt{r_t}$], where l_t is the level of topic t, and `B[l]` has the same shape with `Phi[l]`, but storing the counts.

We maintain a counter on each level to store the largest rank on that level. If the sampler decides to generate a new node on level l, the rank counter is increased by one and the rank of the new node is set to be the rank counter. Then, we grow `B[l]` to have one more column. After each iteration, we reassign the rank by sorting the nodes on each level by their topic count s_t's. We put t into $\mathcal{I}$ if s_t is among the top 95%, so that the rank of the nodes in $\mathcal{I}$ is still continuous. We also need to permute the columns of `Phi[l]` and `B[l]` according to the new rank. The reassignment of the rank is performed on a single worker machine and then broadcasted to all the machines.

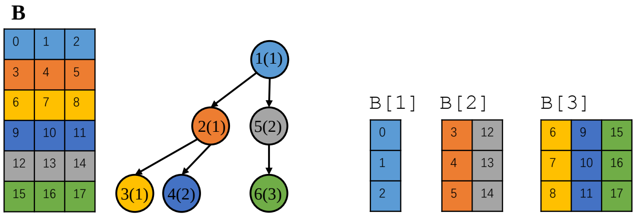


Figure 5: Data layout of our system. The number on the node is "node ID (node rank)". **B** is the counting matrix described in our algorithm and `B[]` is the actual stored format.

To summarize, the main data structures for hLDA are a tree, and L matrices `Phi[]` and `B[]`. `Phi[l]` is a read-only and static matrix during the sampling of $\mathbf{c}$ since both $\log \phi_{tv}$ and $\mathcal{I}$ are static. However, `B[l]` is a read+write matrix with dynamically growing number of columns, which is challenging to implement. We will present the implementation of `B[l]` in the next section.

4.2 Concurrent Dynamic Matrices

The count matrix `B[l]` is concurrently read and updated by the worker threads, and the number of columns (topics) can grow over time. There are $O(NT)$ reads and $O(NL)$ updates per iteration, so the read operation must be efficient, i.e., lock free. On the other hand, the consistency can be relaxed since a small deviation of the counts will not affect the result much. Therefore, we only ask the matrices to have *eventual consistency*, i.e., the values of the matrices should be eventually correct if no new updates are given. We adopt atomic updates to preserve eventual consistency, while the reads are lock-free operations, to maximize the reading performance.

The dynamic number of columns makes the implementation challenging. The straightforward implementation for growing the matrix involves allocating a new memory space, copying the original content to the new space, and deallocating the original space. However, this implementation cannot achieve eventual consistency because the updates during copying will not be incorporated.

Inspired by the lock-free design of a concurrent vector [11], we provide an efficient implementation of the concurrent matrix. Internally, it holds a list of *matrix blocks*, where the i-th matrix block has the size $R \times 2^{c+i-1}$, while c is a constant. The first matrix block represents the $[0, 2^c)$-th columns of the original matrix, the second matrix block represents the $[2^c, 3 \times 2^c)$-th columns of the original matrix, and so on. If there is a growing request that exceeds the current capacity, we allocates the next matrix block on the list. For every reading and updating request, the requested (r, c) coordinate is converted to the (r, b, c') coordinate, where b is the index of the matrix block on the list and c' is the column index within the matrix block. According to Sec. 3.2, each matrix read is always accompanied with an evaluation of the logarithm, so the overhead of the coordinate conversion is acceptable. Finally, to improve the locality, we defragment after each PCGS iteration, i.e., deallocating all the matrix blocks and concatenating their content to form a single larger matrix block. We implement a parameter server to synchronize the updates of the matrices across machines.

4.3 Distributed Tree

To support the tree-structured model, the system needs a distributed tree implementation, which supports adding node, removing node, updating the counts $\mathbf{m}$ and querying the current tree structure. The time complexity of the tree operations is only $O(DL)$ in total, which is much less than the $O(NT)$ complexity of sampling $\mathbf{c}$. Therefore, we simply use mutexes for a thread-safe version of the tree.

To share the tree across machines, one possible solution is a master-client architecture, where the tree is only maintained on the master node, who answers all the queries via RPC calls. This approach has $O(DT)$ communication complexity for the master. Moreover, the master cannot handle requests simultaneously from multiple clients because it uses mutexes. Therefore, this approach is not scalable.

We optimize this solution by maintaining the counts $\mathbf{m}$ in a parameter server. Each machine maintains its own copy of the tree, and the master is only responsible for adding and removing nodes. The count updates are handled by the parameter server. There are $O(L)$ count updates per document, so the cost of updating counts is $O(DL)$. The number of operations for adding or removing nodes is roughly $O(T)$, so the total amount of communication is $O(T^2+DL)$. Since $T \ll D$ and $L \ll T$, this complexity is much lower than the original $O(DT)$. The consistency of the tree structure is guaranteed because the master handles all the node adding and removing operations.

5. RELATED WORK

We discuss on the related work from both parallel inference for nonparametric Bayesian models and scalable systems for flat topic models.

5.1 Parallel Inference for Nonparametric Bayesian Models

Nonparametric Bayesian models [42] are flexible and they can grow their structures as more data are observed. hLDA is a nonparametric Bayesian model that is based on the nCRP prior. Dirichlet process mixture models (DPMMs) [20] and hierarchical Dirichlet processes (HDPs) [28] are the flat nonparametric Bayesian counterparts of hLDA which are also based on CRP. There are many parallel algorithms for DPMMs and HDPs, but none of them can be readily applied to hLDAs. We now briefly review them.

Instantiated weight algorithms convert DPMMs and HDPs to the equivalent stick-breaking process [26] formulation, where an infinite-dimensional mixing proportion is explicitly instantiated. Variational inference [5, 32] or slice sampling [13] algorithms are then developed for the stick-breaking process model variant. Instantiated weight algorithms are efficient and embarrassingly parallel, but we find that they cannot obtain good results for hLDAs due to their susceptibility to local optima.

Split-merge algorithms rely on a split-merge Markov chain Monte-Carlo transition kernel [15] that is able to jump out of local optima via non-local moves, such as splitting a topic or merging two topics. Split-merge algorithms are developed for parallel sampling for DPMMs [9, 8]. However, it is not clear how to split and merge topics on trees.

For hLDA, the only parallel algorithm we are aware of is Wang and Blei's instantiated weight variational inference algorithm with split-merge moves [30]. However, its time complexity is quadratic with respect to the number of topics, which is not scalable to practical big data applications where we often deal with thousands of topics.

5.2 Scalable Systems for Flat Models

There are various distributed systems developed for flat topic models, particularly latent Dirichlet allocation (LDA). These systems mainly focus on partitioning of the data and the model, and the synchronization of the topic-word count matrix $\mathbf{B}$. We categorize existing topic modeling systems in two dimensions

- Style of parallelism: there is data parallel (DP) and hybrid data-and-model parallel (HDMP) approaches;
- Style of synchronization: there are asynchronous updates by assuming that $\mathbf{B}$ changes slowly; and bulk-synchronous-parallel (BSP) updates by formulating the algorithm to be an embarrassingly parallel expectation-maximization (EM) or variational inference (VI) algorithm.

DP+async. Examples include Yahoo! LDA [1] and ParameterServer [17]. They use a parameter server, which is a distributed key-value store, to synchronize the counts. These systems directly approximate the collapsed Gibbs sampling (CGS) algorithm for LDA. However, they assume that the count matrix $\mathbf{B}$ fits in the main memory of a single machine, thus cannot handle a large number of topics or a large vocabulary.
HDMP+sync. Examples include LightLDA, F+LDA and Peacock [37, 36, 34]. Based on a careful hybrid partitioning strategy, they implement the exact collapsed Gibbs sampling algorithm for LDA and thus have better model quality than the DP+async. systems. They can also handle large $\mathbf{B}$ by splitting it across machines. However, the throughput is typically lower than pure DP systems due to the complicated scheduling and the large number of partitions, where WarpLDA [10] is an exception who carefully organizes the scheduling.
DP+sync. Examples include ESCA [38] and Mr.LDA [39]. They use embarrassingly parallel EM or VI algorithms, where the EM algorithm is shown to have similar model quality with CGS [38]. Due to the low system overhead, the throughput can be very high [38], but again they cannot handle large $\mathbf{B}$. This category also includes some recent GPU-based systems such as SaberLDA [16] and BIDMach [40].

Our system for hierarchical topic models is a data parallel one. We borrow the synchronization strategy from both asynchronous and synchronous systems, where we asynchronously update `B[]` via a parameter server and synchronously update `Phi[]` via an `MPI_Allreduce` operation. Different from existing flat topic modeling systems, we propose efficient data layouts and distributed data structures for hierarchical topic models. Exploiting better partition strategies to support large $\mathbf{B}$ is one for our future work.

6. EXPERIMENTS

We now present extensive results to evaluate the quality, efficiency and scalability of our algorithm and system on several datasets, including NIPS, 20NewsGroups, NYTimes and PubMed from the UCI machine learning repository [3], a subset of the NYTimes dataset, and two subsets of the ClueWeb12 dataset [22] (Table 2). Our quantitative and

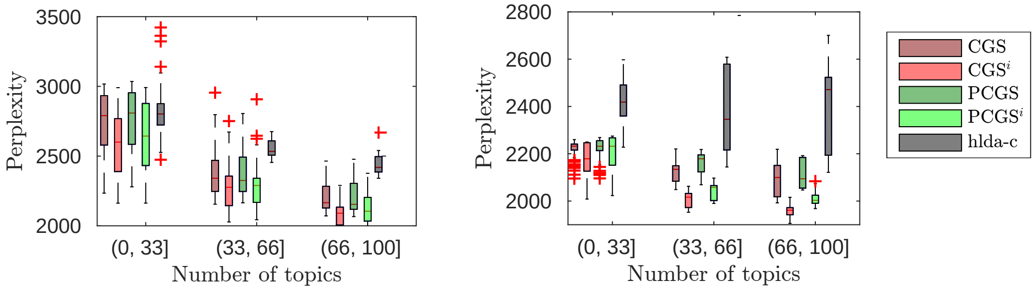


Figure 6: Comparison of inference quality on sNYTimes (left) and NIPS (right). ^{i} denotes for our initialization strategy.

Table 2: Statistics of the datasets.

Dataset	D	# tokens	V
sNYTimes	3×10^3	7.23×10^5	101635
NIPS	1.5×10^3	1.93×10^6	12375
20NewsGroups	1.88×10^4	2.22×10^6	60698
NYTimes	2.93×10^5	9.7×10^7	101635
PubMed	8.2×10^6	7.38×10^8	141043
sClueWeb12	1.5×10^7	5.6×10^9	100000
lClueWeb12	1.31×10^8	2.8×10^{10}	100000

Table 3: Text categorization results on 20NewsGroups.

hLDA algorithm	Perplexity	Testing accuracy	
		+tfidf	+tfidf+lda
None	-	.818	.827
CGS	8168	.828	.830
CGS^i	6500	**.834**	**.836**
PCGS	8228	.812	.825
PCGS^i	7382	.828	.835

qualitative results demonstrate the promise of our system. The experiments are conducted on the Tianhe-2 supercomputer, which has two 12-core Xeon E5-2692v2 CPUs per node and an InfiniBand network.

We quantitatively compare the quality of the inferred models by predictive log-likelihood, which indicates how well we model the testing corpus, using the document completion approach [29]. To compute the predictive log-likelihood, the corpus is divided as a *training corpus* $\mathbf{W}^{\mathrm{t}}$ and a *testing corpus*, and the testing corpus is further divided as an *observed* corpus $\mathbf{W}^{\mathrm{o}}$, which contains a random half of the tokens for each document in the testing corpus; and a *heldout* corpus $\mathbf{W}^{\mathrm{h}}$ of the other half of the tokens. The predictive log-likelihood is defined as $p(\mathbf{W}^{\mathrm{h}}|\mathbf{W}^{\mathrm{o}}, \boldsymbol{\phi})$, where the model $\boldsymbol{\phi}$ is inferred from the training corpus $\mathbf{W}^{\mathrm{t}}$, and is approximated with a Monte-Carlo integration:

$$p(\mathbf{W}^{\mathrm{h}}|\mathbf{w}^{\mathrm{o}}, \boldsymbol{\phi}) \approx \prod_{d=1}^{D} \frac{1}{S} \sum_{s=1}^{S} \prod_{n=1}^{L_d} \sum_{l=1}^{L} p(\mathbf{w}_{dn}^{\mathrm{h}}, \mathbf{z}_{dn}^{\mathrm{h}} = l | \mathbf{c}_d^{(s)}, \boldsymbol{\theta}_d^{(s)}, \boldsymbol{\phi}),$$

where $\mathbf{c}_d^{(s)}$ and $\boldsymbol{\theta}_d^{(s)}$ are the samples from the posterior distribution $p(\mathbf{c}_d^{(s)}, \boldsymbol{\theta}_d^{(s)}|\mathbf{w}^{\mathrm{o}}, \boldsymbol{\phi})$, which can be obtained with Gibbs sampling, and S is the number of samples. Finally, we convert predictive log-likelihood to predictive perplexity

$$\text{perplexity} = \exp(-\text{log likelihood}/\text{number of tokens}).$$

Intuitively, a perplexity of 1,000 means that the model is as confused as the test data as if it chooses uniformly from 1,000 possibilities per word. The perplexity is between 1 and the vocabulary size V, and a lower perplexity score indicates a better model.

6.1 Quality of Inference

We first compare the model inferred by our implementation of CGS and PCGS and examine the effect of the initialization strategy (Sec. 3.3), where CGS is a special case of PCGS with an empty set as $\mathcal{I}$. We also include a comparison with the open source implementation for hLDA, `hlda-c`, which is a CGS algorithm but has a stick-breaking prior on $\boldsymbol{\theta}$ instead of a Dirichlet prior [5].

Unlike parametric models, whose numbers of topics are fixed, nonparametric models such as hLDA produce different numbers of topics for different runs and various inference algorithms, even with the same hyper-parameter setting. It is not fair to directly compare the perplexity of two models with different numbers of topics. For a fair comparison, we choose a rich set of hyper-parameter configurations, run the algorithms for all these configurations, and plot the perplexity against the number of topics as in Fig. 6. In this experiment, we train a 4-layer model (i.e., $L = 4$) on the sNYTimes dataset, which is a subset of the NYTimes dataset, and the NIPS dataset, and $\boldsymbol{\beta} = (\beta_0, 0.5\beta_0, 0.25\beta_0, 0.25\beta_0)$, where β_0 is chosen from $\{e^{-4.0}, e^{-3.5}, \dots, e^{2.0}\}$, γ is chosen from $\{e^{-6.0}, e^{-5.5}, \dots, e^{0.0}\}$, and $\boldsymbol{\alpha} = 0.2 \times \mathbf{1}$.

By comparing the perplexity produced by different algorithms in Fig. 6, we have a number of observations:

- CGS and PCGS have similar quality.
- Our initialization strategy helps obtain better results for both CGS and PCGS.
- Our result is not worse (actually better) than `hlda-c`. The discrepancy attributes to the different choice of prior on $\boldsymbol{\theta}$.

Besides perplexity, we also present accuracy results of the supervised text categorization task on the 20NewsGroups dataset, whose each document is labeled as one of the 20 categories. We utilize topic models for feature extraction, where the feature for document d is the count vector $(f_{dt})_{t=1}^{T}$, where $f_{dt} = \sum_v b_{tv}^d$ indicates the numbers of occurrences of topic t in document d. We also incorporate a popular flat topic model, latent Dirichlet allocation (LDA) for comparison. The result is in Table 3. For each algorithm, we report the perplexity, and the testing accuracy of a linear support

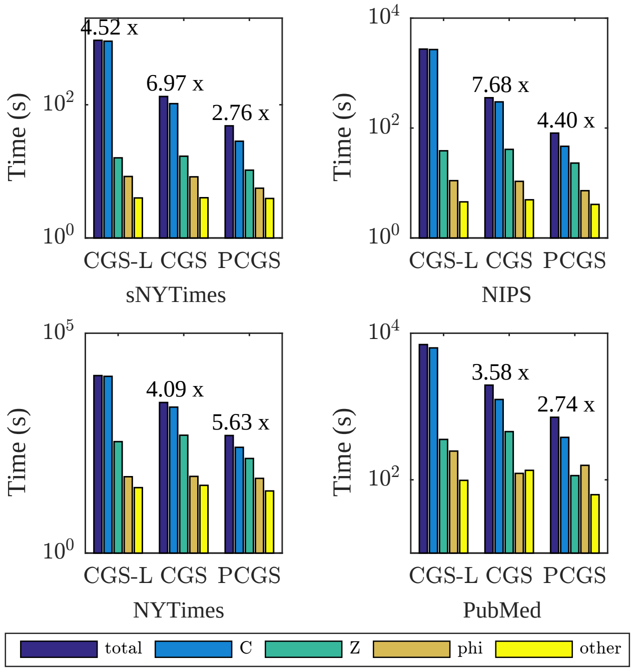


Figure 7: Running time comparison. The numbers on the plot are the speedups over the previous implementation, e.g., the speedup of CGS over CGS-L. CGS-L is 4.52 times faster than `hlda-c` on sNYTimes.

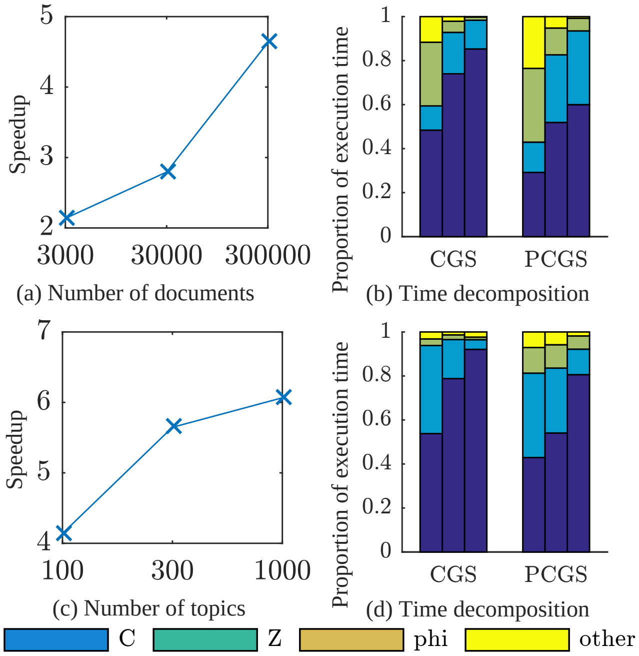


Figure 8: Speedup of PCGS over CGS w.r.t. the number of documents. The three bars in are 3000, 30,000, and 300,000 documents in (b) ; and 100, 300, and 1,000 topics in (d).

vector machine (liblinear [12]) with two different set of features. The first feature set "+tfidf" contains standard tf-idf word features [24] and hLDA topic features; and the second feature set "+tfidf+lda" further has LDA features. We report the accuracy for different hLDA training algorithms, and the baseline "None" does not utilize any hLDA features.

CGS^i+tfidf features give 0.834 accuracy, outperforming the 0.827 accuracy by the None+tfidf+LDA baseline. Utilizing the features produced by both hLDA and LDA, the CGS^i+tfidf+LDA approach further improves the accuracy to 0.836. The comparison between different training algorithms for hLDA is similar with our previous perplexity study of Fig. 6. PCGS^i+tfidf+LDA gives similar 0.835 accuracy with CGS^i+tfidf+LDA, and the algorithms without our initialization strategies, CGS and PCGS, are considerably worse in terms of both perplexity and testing accuracy.

6.2 Efficiency

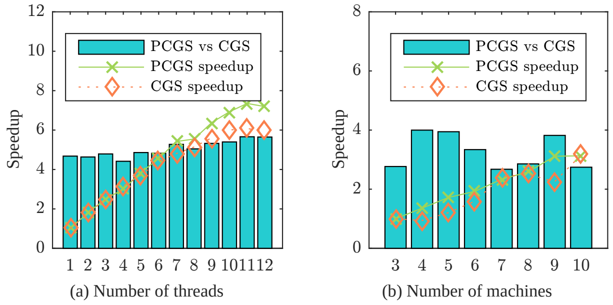
We now study the impact of our PCGS algorithm and the data layout to the efficiency of system. We compare the running time of three implementations, PCGS, CGS, and CGS without the proposed data layout and vectorization ("CGS-L") on sNYTimes, NIPS, NYTimes and PubMed datasets. The number of topics is kept around 300 by tuning the hyper-parameters. To keep the running time reasonable, we utilize a single thread on sNYTimes and NIPS, 12 threads on NYTimes, and ten 12-core CPUs on PubMed. On sNYTimes, we also report the running time of `hlda-c`, whose running time is longer than 2 hours on all the other datasets. The running time is shown in Fig. 7. We report the total running time as well as the running time for each sampling step, i.e., sampling C, Z and ϕ as shown in Table 1. Non-sampling operations, such as maintaining the tree and the system overhead, are denoted as "other". PCGS gives 2.74-5.63 times speedup over CGS, by avoiding the computing of logarithms and cheaper read-only data structures. The data layout and vectorization proposed in Sec. 4.1 give 3.58-7.68 times speedup. Furthermore, CGS-L is 4.52 times faster than `hlda-c` due to the implementation, mostly because we cast difference of logarithm of gamma function as sums of logarithms as Eq. 3. Overall, our PCGS implementation is 87 times faster than `hlda-c` on the sNYTimes dataset.

6.3 Impact of the Problem Size

We further examine the impact factors to the speedup of PCGS over CGS. PCGS mainly reduces the cost of sampling **c**. By Table 1, the time complexity of sampling **c** is proportional to both the corpus size N and the number of topics T. Therefore, we expect the speedup of PCGS over CGS is large when the cost of sampling **c** dominates, i.e., when N and T are large. We report the speedup results on the NYTimes dataset, varying N and T, while keeping all the other factors fixed. Fig. 8(a) shows the speedup of PCGS over CGS increased from 2.1 to 4.6 times when the number of documents increases from 3,000 to 300,000. The increase of speedup is due to the increased proportion of time consumption for sampling **c**, as shown in Fig. 8(b). Similarly, Fig. 8(c) shows that the speedup of PCGS over CGS increased from 4.1 to 6.1 times when the number of topics increases from 100 to 1,000; and the time proportion of each step is shown in Fig. 8(d).

6.4 Impact of the Machine Size

We also study whether of the number of threads and CPUs affects the speedup of PCGS over CGS. We run PCGS and CGS from 1 to 12 threads on the NYTimes corpus, and



(a) Number of threads (b) Number of machines

Figure 9: Speedup of PCGS over CGS w.r.t. the number of threads and machines. The two lines are the speedup of PCGS / CGS as the number of threads / machines grows, and the bar is the speedup of PCGS over CGS.

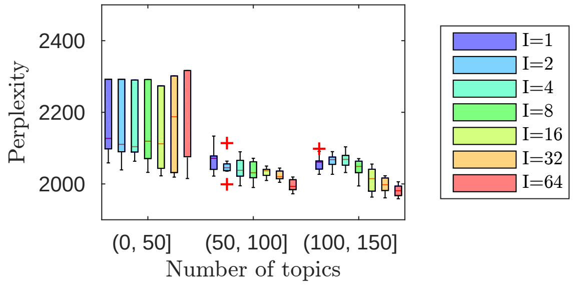


Figure 10: Impact of I to perplexity.

from 3 to 10 CPUs on the PubMed corpus, because CGS exceeds the main memory capacity on 1-2 machines. CGS needs to access the concurrent dynamic matrix more often than PCGS. The scalability of the concurrent dynamic matrix is worse than a read-only matrix which PCGS mostly accesses, due to the contention of the atomic update operations. Therefore, the speedup of PCGS over CGS increases as the number of threads grows, as shown in Fig. 9(a). Meanwhile, the updates of the distributed matrices and trees are done asynchronously, so the speedup of PCGS over CGS remains relatively stable with respect to the number of machines, as shown in Fig. 9(b).

6.5 Sensitivity of Parameters

We now examine the impact of the hyper-parameters I and S of our initialization strategy.

Impact of I: I is the number of initializing iterations to sample from $p(\mathbf{c}|\mathbf{w})$. We run PCGS on the sNYTimes dataset, setting $\beta_0 = 1$, and varying I and γ. It can be seen from Fig. 10 that the perplexity steadily decreases for large I, which again shows that our initialization strategy is helpful. We select a moderate $I = 32$ for all the experiments.

Impact of S: S is the number of Monte-Carlo samples to approximate $p(\mathbf{c}|\mathbf{w})$. When $S \to \infty$, we directly sample from $p(\mathbf{c}|\mathbf{w})$ in the first I iterations. We run PCGS on sNYTimes, with $\boldsymbol{\beta} = (1.0, 0.5, 0.25, 0.25)$ and $\gamma = 10^{-40}$, and vary S from 1 to 128. As shown in Fig. 11, S has little impact on both the number of topics and the perplexity, implying that a small S, e.g., $S = 5$, is adequate.

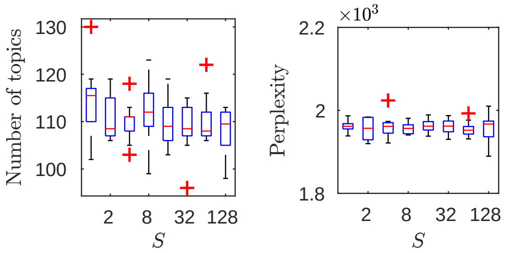


Figure 11: Impact of the number of Monte-Carlo samples S.

6.6 Scalability

Our experiments on scalability are in two folds: whether the quality of inference is affected by parallelization; and how good is the speedup. We repeat each experiment for 10 times. For each experiment, we choose the largest dataset such that the data fit in the main memory and the running time is less than two hours. Based on this criteria, we choose NYTimes for the multi-thread setting, PubMed for 1-10 machines, and sClueWeb12 for 10-100 machines, which is a subset of the ClueWeb12 corpus.

For the multi-thread setting, where the number of threads varies from 1 to 12 on the NYTimes corpus. The result is shown in Fig. 12(a), where we observe that there is no apparent increase of the perplexity as the number of threads grows. The speedup with 12 threads is 8.56. The probable reasons of imperfect speedup include serial region, contention for atomic variables, and limited memory bandwidth.

For the multi-machine setting, there are two CPUs per machine. We run our implementation on the PubMed corpus on 1 to 10 CPUs as shown in Fig. 12(b), and on the larger sClueWeb12 corpus for 10 to 100 CPUs as shown in Fig. 12(c). The speedup is 8.5 from 1 to 10 CPUs, and 7.15 from 10 to 100 CPUs. The perplexity is slightly affected by parallelization when the number of CPUs exceeds 7 and 80 on the two datasets, respectively, indicating that the dataset is not large enough to utilize that many CPUs.

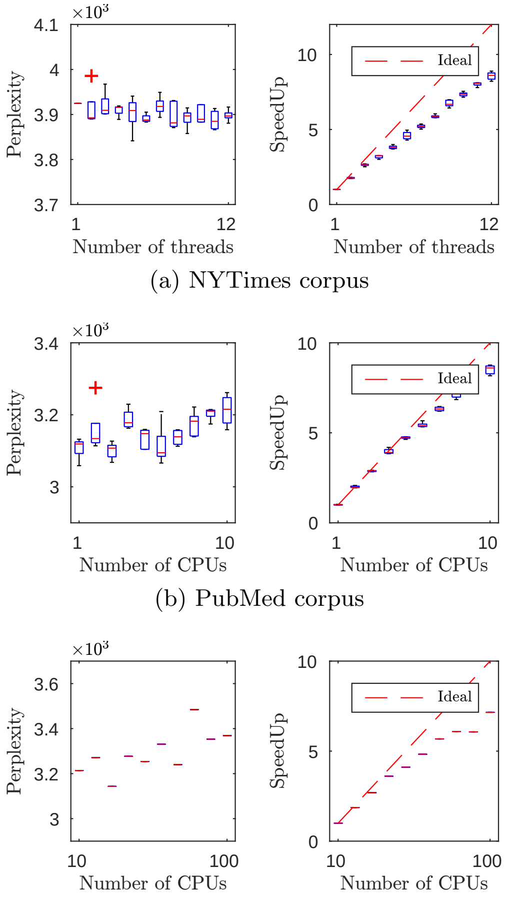
Finally, to demonstrate the scalability, we learn a model with 1,722 topics of the 131-million-document lClueWeb12 corpus with 50 machines, and the inference finishes in 7 hours. The results will be presented for qualitative evaluation in the next section.

6.7 Qualitative Analysis

We now demonstrate the topic hierarchy obtained from the lClueWeb12 corpus, which is a crawl of web pages. The corpus is obtained by tokenizing the original ClueWeb12 dataset, randomly selecting about 30% documents, truncating the vocabulary size to 100,000 and keeping only the documents whose length are between $[50, 500]$. We show the selected parts of the obtained tree in Fig. 13, where some topics whose number of occurrences does not pass a particular threshold are filtered out, and the font size of words is proportional to the 4th root of their frequency in the topic. The tree has 5 levels in total.[1]

Fig. 13(a) shows some selected topics on the first 3 levels. The root node contains the most commonly used words shared by all the documents. The second level contains a

[1] The visualization demo is available online at `http://ml.cs.tsinghua.edu.cn/~jianfei/scalable-hlda.html`.



(a) NYTimes corpus

(b) PubMed corpus

(c) sClueWeb12 corpus

Figure 12: Perplexity and speedup as the amount of computational resource increases.

variety of general topics, such as “software”, “travel” and “city”, and the third level has more detailed concepts, e.g., the “city” topic on the second level splits as “shopping”, “city names”, and “locations”. We further show the topic subtrees of all the layers rooted at the highlighted nodes to examine the fine-grained concepts. For example, in Fig. 13(b) the “travel” topic is divided as “islands”, “India” and “vacations”, and the leaf level contains specific concepts, such as “ferry” and “diving”, which are correctly placed under the “islands” topic. In Fig. 13(c), the “computer” topic is divided as “website”, “windows”, “vps”, “programming”, “linux” and “forum”. To our knowledge, this is the first time that hLDA is applied to large-scale web data, and the results demonstrate our ability on automatically learning topic hierarchy from web data.

7. CONCLUSIONS AND DISCUSSIONS

We address the problem of scalable training of hierarchical topic models, taking hierarchical latent Dirichlet allocation (hLDA) as a concrete example. We present a partially collapsed Gibbs sampling algorithm for the hierarchical latent Dirichlet allocation model. PCGS is efficient because it avoids most computation of logarithms. To deal with the local minima introduced by the tree structure, we present an initialization strategy that improves the model quality. The complicated data structures of hierarchical topic models pose challenges for building scalable training systems. To solve these challenges, we design a distributed training system with an efficient data layout for vectorization, concurrent dynamic matrices and distributed trees. The proposed system can handle hundreds of millions of documents, thousands of topics, with thousands of CPU cores.

In the future, we plan to extend our method to the more sophisticated nested hierarchical Dirichlet process model [21, 2]. Developing sampling algorithms with a sub-linear time complexity with respect to the number of topics is also a promising direction.

8. ACKNOWLEDGMENTS

We thank Arnab Bhadury for his help in proofreading the paper. The work was supported by the National NSF of China (Nos. 61620106010, 61621136008, 61332007, 61761136020), Beijing Natural Science Foundation (No. L172037), and a Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund (the 2nd phase).

Appendix

Derivation of PCGS updates: Rewrite the joint distribution in Sec. 2.4 as:

$$p(\mathbf{W},\mathbf{z},\boldsymbol{\theta},\mathbf{c},\boldsymbol{\phi}_{\bar{\mathcal{I}}},\boldsymbol{\phi}_{\mathcal{I}})=\prod_{t\in\bar{\mathcal{I}}}p(\boldsymbol{\phi}_t)\prod_{t\in\mathcal{I}}p(\boldsymbol{\phi}_t)$$

$$\prod_{d=1}^{D}p(\mathbf{c}_d|\mathbf{c}_{<d})p(\boldsymbol{\theta}_d)\prod_{d=1}^{D}\prod_{n=1}^{N_d}p(z_{dn}|\boldsymbol{\theta})p(w_{dn}|z_{dn},\boldsymbol{\phi}_{\bar{\mathcal{I}}},\boldsymbol{\phi}_{\mathcal{I}}).$$

Integrating out $\boldsymbol{\phi}_{\mathcal{C}}$ and $\boldsymbol{\theta}$, we have the marginal distribution

$$p(\mathbf{W},\mathbf{z},\mathbf{c},\boldsymbol{\phi}_{\mathcal{I}})=\prod_{d=1}^{D}\left[p(\mathbf{c}_d|\mathbf{c}_{<d})\frac{B(\mathbf{a}_d+\boldsymbol{\alpha})}{B(\boldsymbol{\alpha})}\right]\prod_{t\in\bar{\mathcal{I}}}\frac{B(\mathbf{b}_t+\beta_{l_t})}{B(\beta_{l_t})}$$

$$\prod_{t\in\mathcal{I}}\left[\mathrm{Dir}(\boldsymbol{\phi}_t;\beta_{l_t})\prod_{v=1}^{V}(\phi_{tv})^{b_{tv}}\right].$$

Utilizing the identity $\frac{B(\boldsymbol{\alpha}+\mathbf{e}_k)}{B(\boldsymbol{\alpha})}=\frac{\alpha_k}{\sum_k\alpha_k}$, where $\mathbf{e}_k$ is a coordinate vector, we can derive the Gibbs sampling updates:

Sample z: Keeping only the terms relevant with z_{dn}, we have $p(z_{dn}=l|w_{dn}=v,\mathbf{w},\mathbf{z}_{\neg dn},\mathbf{c},\boldsymbol{\phi}^{\mathcal{I}})\propto\frac{B(\mathbf{C}_d+\boldsymbol{\alpha})}{B(\boldsymbol{\alpha})}\prod_{t\in\mathcal{C}}\frac{B(\mathbf{C}_t+\beta_{l_t})}{B(\beta_{l_t})}\prod_{t\in\mathcal{I}}\prod_{v=1}^{V}\phi_{tv}^{C_{tv}}\propto\frac{B(\mathbf{C}_d^{\neg dn}+\mathbf{C}_d^{dn}+\boldsymbol{\alpha})}{B(\mathbf{C}_d^{\neg dn}+\boldsymbol{\alpha})}$ $\prod_{t\in\mathcal{C}}\frac{B(\mathbf{C}_t^{\neg dn}+\mathbf{C}_t^{dn}+\beta_{l_t})}{B(\mathbf{C}_t^{\neg dn}+\beta_{l_t})}\prod_{t\in\mathcal{I}}\prod_{v=1}^{V}\phi_{tv}^{C_{tv}^{\neg dn}+C_{tv}^{dn}}$.

Sample c: $p(\mathbf{c}_d=\mathbf{c}|\mathbf{w},\mathbf{z},\mathbf{c}_{\neg d},\boldsymbol{\phi}^{\mathcal{I}})\propto p(\mathbf{c}_d|\mathbf{c}_{\neg d})\prod_{t\in\mathcal{C}}\frac{B(\mathbf{C}_t+\beta_{l_t})}{B(\beta_{l_t})}\prod_{t\in\mathcal{I}}\prod_{v=1}^{V}\phi_{tv}^{C_{tv}}\propto p(\mathbf{c}_d|\mathbf{c}_{\neg d})\prod_{t\in\mathcal{C}}\frac{B(\mathbf{C}_t^{\neg d}+\mathbf{C}_t^{d}+\beta_{l_t})}{B(\mathbf{C}_t^{\neg d}+\beta_{l_t})}$ $\prod_{t\in\mathcal{I}}\prod_{v=1}^{V}\phi_{tv}^{C_{tv}^{d}}\propto\mathrm{nCRP}(\mathbf{c};\boldsymbol{\gamma},\mathbf{c}_{\neg d})\prod_{l=1}^{L}\begin{cases}f_I(d,c_l) & c_l\in\mathcal{I},\\ f_C(d,c_l) & c_l\in\mathcal{C}.\end{cases}$

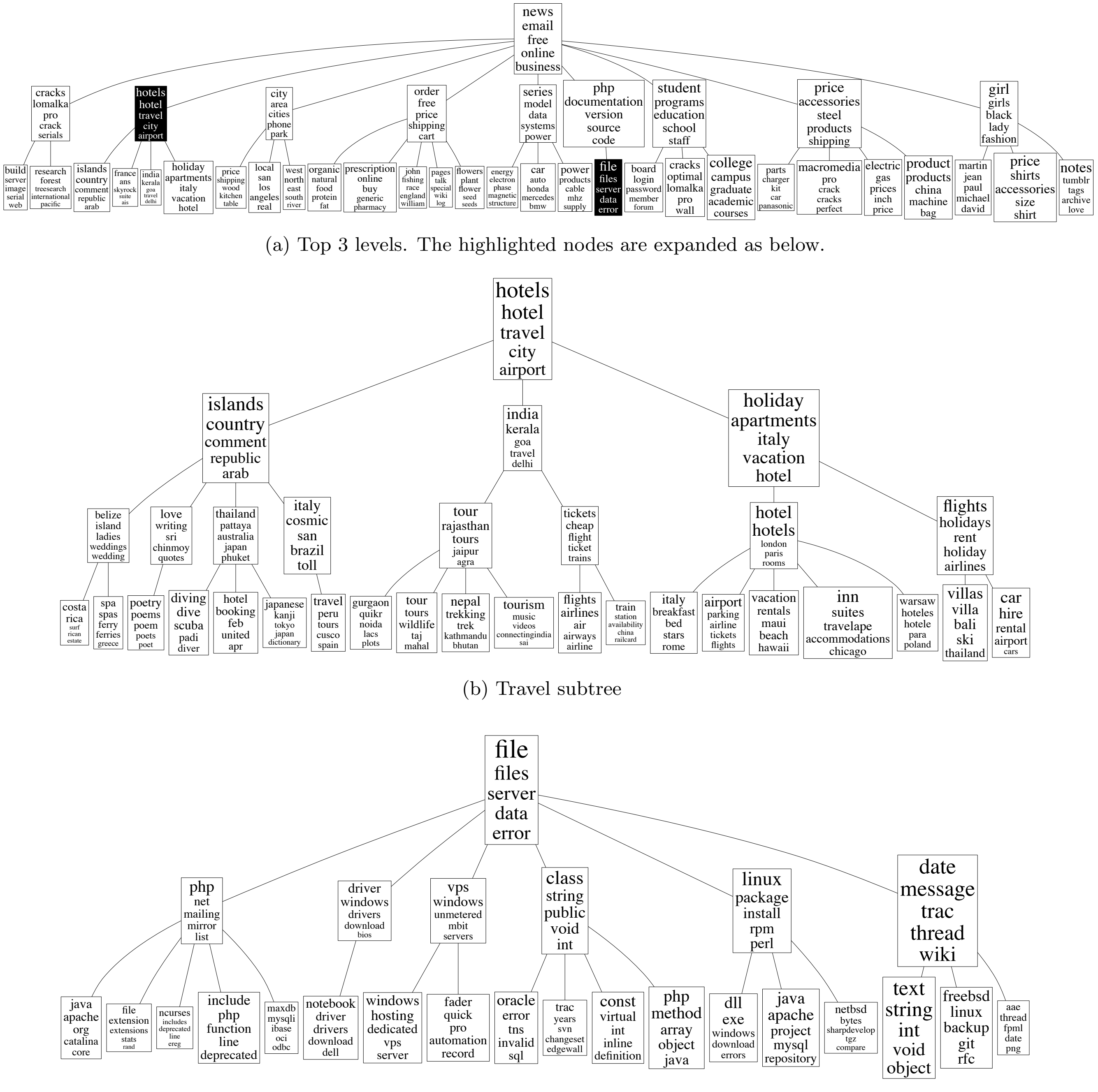
Sample $\boldsymbol{\phi}^{\mathcal{I}}$: For $t\in\mathcal{I}$, draw $\boldsymbol{\phi}_t\sim\mathrm{Dir}(\beta_{l_t}+\mathbf{b}_t)$.

Derivation of $\log f_C(d,t)$:

We have $\log f_C(d,t)=\log\frac{B(\mathbf{b}_t^{\neg d}+\mathbf{b}_t^{d}+\beta_{l_t})}{B(\mathbf{b}_t^{\neg d}+\beta_{l_t})}$

$=\left[\sum_{v=1}^{V}\sum_{i=0}^{b_{tv}^{d}-1}\log(b_{tv}^{\neg d}+i+\beta_{l_t})\right]+h_t$

$=\sum_{(v,o)\in\mathcal{W}'_{dl}}\log(b_{tv}^{\neg d}+o+\beta_{l_t})+h_t.$


news email free online business
cracks lomalka pro crack serials
hotels hotel travel city airport
city area cities phone park
order free price shipping cart
series model data systems power
php documentation version source code
student programs education school staff
price accessories steel products shipping
girl girls black lady fashion
build server image serial web
research forest treesearch international pacific
islands country comment republic arab
france ans skyrock suite ais
india kerala goa travel delhi
holiday apartments italy vacation hotel
price shipping wood kitchen table
local san los angeles real
west north east south river
organic natural food protein fat
prescription online buy generic pharmacy
john fishing race england william
pages talk special wiki log
flowers plant flower seed seeds
energy electron phase magnetic structure
car auto honda mercedes bmw
power products cable mhz supply
file files server data error
board login password member forum
cracks optimal lomalka pro wall
college campus graduate academic courses
parts charger kit car panasonic
macromedia pro crack cracks perfect
electric gas prices inch price
product products china machine bag
martin jean paul michael david
price shirts accessories size shirt
notes tumblr tags archive love

(a) Top 3 levels. The highlighted nodes are expanded as below.

hotels hotel travel city airport
islands country comment republic arab
india kerala goa travel delhi
holiday apartments italy vacation hotel
belize island ladies weddings wedding
love writing sri chinmoy quotes
thailand pattaya australia japan phuket
italy cosmic san brazil toll
tour rajasthan tours jaipur agra
tickets cheap flight ticket trains
hotel hotels london paris rooms
flights holidays rent holiday airlines
costa rica surf rican estate
spa spas ferry ferries greece
poetry poems poem poets poet
diving dive scuba padi diver
hotel booking feb united apr
japanese kanji tokyo japan dictionary
travel peru tours cusco spain
gurgaon quikr noida lacs plots
tour tours wildlife taj mahal
nepal trekking trek kathmandu bhutan
tourism music videos connectingindia sai
flights airlines air airways airline
train station availability china railcard
italy breakfast bed stars rome
airport parking airline tickets flights
vacation rentals maui beach hawaii
inn suites travelape accommodations chicago
warsaw hoteles hotele para poland
villas villa bali ski thailand
car hire rental airport cars

(b) Travel subtree

file files server data error
php net mailing mirror list
driver windows drivers download bios
vps windows unmetered mbit servers
class string public void int
linux package install rpm perl
date message trac thread wiki
java apache org catalina core
file extension extensions stats rand
ncurses includes deprecated line ereg
include php function line deprecated
maxdb mysqli ibase oci odbc
notebook driver drivers download dell
windows hosting dedicated vps server
fader quick pro automation record
oracle error tns invalid sql
trac years svn changeset edgewall
const virtual int inline definition
php method array object java
dll exe windows download errors
java apache project mysql repository
netbsd bytes sharpdevelop tgz compare
text string int void object
freebsd linux backup git rfc
aae thread fpml date png

(c) Computer subtree

Figure 13: Selected subtrees on the topic hierarchy extracted from ClueWeb12 (large).

9. REFERENCES

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

[2] A. Ahmed, L. Hong, and A. J. Smola. Nested chinese restaurant franchise process: Applications to user tracking and document modeling. In *ICML*, 2013.

[3] A. Asuncion and D. Newman. Uci machine learning repository, 2007.

[4] D. Blei. hlda-c. `http://www.cs.columbia.edu/~blei/downloads/hlda-c.tgz`, 2009.

[5] D. M. Blei, T. L. Griffiths, and M. I. Jordan. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)*, 57(2):7, 2010.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[7] J. L. Boyd-Graber, D. M. Blei, and X. Zhu. A topic model for word sense disambiguation. In *EMNLP-CoNLL*, 2007.

[8] T. Campbell, J. Straub, J. W. Fisher III, and J. P. How. Streaming, distributed variational inference for bayesian nonparametrics. In *NIPS*, 2015.

[9] J. Chang and J. W. Fisher III. Parallel sampling of dp mixture models using sub-cluster splits. In *Advances in Neural Information Processing Systems*, pages 620–628, 2013.

[10] J. Chen, K. Li, J. Zhu, and W. Chen. Warplda: a cache efficient o(1) algorithm for latent dirichlet allocation. *PVLDB*, 9(10):744–755, 2016.

[11] D. Dechev, P. Pirkelbauer, and B. Stroustrup. Lock-free dynamically resizable arrays. In *International Conference On Principles Of Distributed Systems*, 2006.

[12] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[13] H. Ge, Y. Chen, E. CAM, M. Wan, and Z. Ghahramani. Distributed inference for dirichlet process mixture models. In *ICML*, 2015.

[14] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[15] S. Jain and R. M. Neal. A split-merge markov chain monte carlo procedure for the dirichlet process mixture model. *Journal of Computational and Graphical Statistics*, 13(1):158–182, 2004.

[16] K. Li, J. Chen, W. Chen, and J. Zhu. Saberlda: Sparsity-aware learning of topic models on gpus. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 497–509. ACM, 2017.

[17] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 1, page 3, 2014.

[18] W. Li and A. McCallum. Pachinko allocation: Dag-structured mixture models of topic correlations. In *ICML*, 2006.

[19] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[20] R. M. Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

[21] J. Paisley, C. Wang, D. M. Blei, and M. I. Jordan. Nested hierarchical dirichlet processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):256–270, 2015.

[22] T. L. Project. The clueweb12 dataset. `http://lemurproject.org/clueweb12/`, 2013.

[23] J. Pujara and P. Skomoroch. Large-scale hierarchical topic models. In *NIPS Workshop on Big Learning*, 2012.

[24] J. Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142, 2003.

[25] L. Rigouste, O. Cappé, and F. Yvon. Inference and evaluation of the multinomial mixture model for text clustering. *Information processing & management*, 43(5):1260–1280, 2007.

[26] J. Sethuraman. A constructive definition of dirichlet priors. *Statistica sinica*, pages 639–650, 1994.

[27] Y. W. Teh. Dirichlet process. In *Encyclopedia of machine learning*, pages 280–287. Springer, 2011.

[28] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. Sharing clusters among related groups: Hierarchical dirichlet processes. In *NIPS*, 2004.

[29] H. M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno. Evaluation methods for topic models. In *ICML*, 2009.

[30] C. Wang and D. M. Blei. Variational inference for the nested chinese restaurant process. In *NIPS*, 2009.

[31] C. Wang and D. M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 448–456. ACM, 2011.

[32] C. Wang and D. M. Blei. Truncation-free online variational inference for bayesian nonparametric models. In *Advances in neural information processing systems*, pages 413–421, 2012.

[33] X. Wang, S. Liu, J. Liu, J. Chen, J. J. H. Zhu, and B. Guo. Topicpanorama: A full picture of relevant topics. *TVCG*, 22(12):2508–2521, 2016.

[34] Y. Wang, X. Zhao, Z. Sun, H. Yan, L. Wang, Z. Jin, L. Wang, Y. Gao, J. Zeng, Q. Yang, et al. Towards topic modeling for big data. *ACM Transactions on Intelligent Systems and Technology*, 9(4), 2014.

[35] X. Wei and W. B. Croft. Lda-based document models for ad-hoc retrieval. In *SIGIR*, 2006.

[36] H.-F. Yu, C.-J. Hsieh, H. Yun, S. Vishwanathan, and I. S. Dhillon. A scalable asynchronous distributed algorithm for topic modeling. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1340–1350. International World Wide Web Conferences Steering Committee, 2015.

[37] J. Yuan, F. Gao, Q. Ho, W. Dai, J. Wei, X. Zheng,

E. P. Xing, T.-Y. Liu, and W.-Y. Ma. Lightlda: Big topic models on modest computer clusters. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1351–1361. International World Wide Web Conferences Steering Committee, 2015.

[38] M. Zaheer, M. Wick, J.-B. Tristan, A. Smola, and G. L. Steele Jr. Exponential stochastic cellular automata for massively parallel inference. In *AISTATS*, 2016.

[39] K. Zhai, J. Boyd-Graber, N. Asadi, and M. L. Alkhouja. Mr. lda: A flexible large scale topic modeling package using variational inference in mapreduce. In *WWW*, 2012.

[40] H. Zhao, B. Jiang, J. F. Canny, and B. Jaros. Same but different: Fast and high quality gibbs parameter estimation. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1495–1502. ACM, 2015.

[41] J. Zhu, A. Ahmed, and E. P. Xing. MedLDA: Maximum margin supervised topic models. *JMLR*, 13:2237–2278, 2012.

[42] J. Zhu, J. Chen, W. Hu, and B. Zhang. Big learning with bayesian methods. *National Science Review*, page nwx044, 2017.