# WarpLDA: a Simple and Efficient O(1) Algorithm for Latent Dirichlet Allocation

Jianfei Chen, Kaiwei Li, Jun Zhu, Wenguang Chen
Dept. of Comp. Sci. & Tech.; State Key Lab of Intell. Tech. & Sys.; Center for Bio-Inspired Research
Parallel Architecture & Compiler technology of Mobile, Accelerated, and Networked systems (PACMAN)
Tsinghua National Lab of Information Science & Technology; Tsinghua University, Beijing, 100084 China
{chenjian14, likw14}@mails.tsinghua.edu.cn; {dcszj, cwg}@tsinghua.edu.cn

## ABSTRACT

Developing efficient and scalable algorithms for Latent Dirichlet Allocation (LDA) is of wide interest for many applications. Previous work has developed an $O(1)$ Metropolis-Hastings sampling method for each token. However, the performance is far from being optimal due to random accesses to the parameter matrices and frequent *cache misses*.

In this paper, we propose WarpLDA, a novel $O(1)$ sampling algorithm for LDA. WarpLDA is a Metropolis-Hastings based algorithm which is designed to optimize the cache hit rate. Advantages of WarpLDA include 1) *Efficiency and scalability*: WarpLDA has good locality and carefully designed partition method, and can be scaled to hundreds of machines; 2) *Simplicity*: WarpLDA does not have any complicated modules such as alias tables, hybrid data structures, or parameter servers, making it easy to understand and implement; 3) *Robustness*: WarpLDA is consistently faster than other algorithms, under various settings from small-scale to massive-scale dataset and model.

WarpLDA is 5-15x faster than state-of-the-art LDA samplers, implying less cost of time and money. With WarpLDA users can learn up to one million topics from hundreds of millions of documents in a few hours, at the speed of 2G tokens per second, or learn topics from small-scale datasets in seconds.

## Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software —*Distributed Systems*; G.3 [**Probability and Statistics**]: Probabilistic algorithms (including Monte Carlo); G.4 [**Mathematical Software**]: Parallel and vector implementations

## General Terms

Design, Algorithms, Experimentation, Performance

## Keywords

Large-scale machine learning; Distributed systems; Parallel computing; Topic model; Metropolis-Hastings; Cache; MPI; NUMA

## 1. INTRODUCTION

Topic modeling provides a suite of statistical tools to discover latent semantic structures from complex corpora. Among various models, latent Dirichlet allocation (LDA) [4] is the most popular one, with many applications in text analysis [5, 31], data visualization [13, 17], recommendation systems [10], information retrieval [27] and network analysis [8, 9]. LDA represents each document as an admixture of topics, each of which is a unigram distribution of words. Since exact inference is intractable, both variational Bayes (VB) and Markov Chain Monte Carlo (MCMC) methods have been developed for approximate inference, including mean-field variational Bayes [4], collapsed variational Bayes [22], expectation propagation [19], and collapsed Gibbs sampling (CGS) [11]. Among these methods, CGS is most popular due to its simplicity and availability for fast sparsity-aware algorithms [28, 15, 29].

Entering the Big Data era, there are two diverse trends, both demanding highly efficient algorithms to learn topic models. On one hand, industrial applications require *large-scale* topic modeling to boost their performance. For example, Wang et al. [26] show that scaling up topic models to 1 million topics leads to significant performance gain on various tasks such as advertisement and recommendation. Other recent endeavours on learning large-scale topic models often contain billions of documents, millions of topics, and millions of unique tokens [26, 29, 16]. On the other hand, applications such as machine learning in the browser [18] may require to do topic modeling on small-scale data in *real time*. There needs an algorithm which performs well on various datasets from small-scale to massive-scale and on different machines from a personal computer to large supercomputers. Such practical requirements have motivated the recent developments of efficient algorithms of LDA, which can be grouped into three categories: stochastic learning, fast sampling algorithm, and system optimization.

Stochastic algorithms explore the statistical redundancy of a given corpus, and estimate the statistics (e.g., gradient) of the whole corpus by a random subset. When the estimate has low variance, faster convergence is expected. Examples include stochastic variational inference (SVI) [12], streaming variational Bayes [6], and stochastic gradient Riemann Langevin dynamics (SGRLD) [21]. However, due to the validity of the data redundancy assumption[1] and the cost of updating models, stochastic algorithms are often limited to small model sizes (e.g., 100 topics and 10,000 unique tokens), which does not satisfy the need for very large models. We therefore do not use stochastic algorithms in this paper, and leave the stochastic case of our algorithm as one of the future works.

Fast sampling algorithms optimize the time complexity of sam-

[1] Redundancy is relative to model size. For a given corpus, redundancy decreases as the model gets larger; thereby a stochastic algorithm becomes less effective.

Table 1: Summary of existing algorithms of LDA, where $K$: number of topics, $K_d$: average number of topics per document, $K_w$: average number of topics per word, $D$: number of documents, $V$: size of vocabulary. $^*$: requires large mini batch size for good speedup.

| Type | Machines | Algorithm | Time | Time | Locality | Bottle- | Parallel | $K$ | $V$ | $D$ |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $O$(per token) | $O$(per update) | | neck | | $(\times 10^3)$ | $(\times 10^3)$ | $(\times 10^3)$ |
| Batch | 1 | VB [4] | $K$ | $(D+V)K$ | good | compute | - | 0.2 | 23 | 16 |
| | | CGS [11] | $K$ | - | good | compute | - | 1 | 20 | 28 |
| | | SparseLDA [28] | $K_d + K_w$ | - | good | compute | - | 0.1 | unknown | 39 |
| | | AliasLDA [15] | $K_d$ | $VK_w$ | medium | compute | - | 4 | 101 | 300 |
| | $2 \times 10^3$ | Yahoo!LDA [1] | $K_d + K_w$ | - | good | compute | lock | 1 | unknown | 20 |
| | $6 \times 10^3$ | PS [16] | $K_d + K_w$ | - | good | compute | unknown | 2 | $5 \times 10^3$ | $5 \times 10^6$ |
| | 24 | LightLDA [29] | 1 | $VK_w$ | poor | memory | delta thread | $10^3$ | $10^3$ | $10^6$ |
| | $1 \sim 128$ | **WarpLDA** | 1 | - | medium | memory | embarrassingly | $10^3$ | $10^3$ | $1.5 \times 10^5$ |
| Stochastic | 1 | SVI [12] | $K$ | $KV$ | good | compute | embarrassingly$^*$ | 0.1 | 8 | $4 \times 10^3$ |
| | | SGRLD [12] | $K$ | $KV$ | good | compute | embarrassingly$^*$ | 0.05 | 8 | 150 |
| | 4 GPUs | BIDMach [7] | $K$ | $KV$ | good | compute | embarrassingly$^*$ | 0.3 | 50 | $3 \times 10^4$ |

pling a single token, by utilizing sparse structures of the model or Metropolis-Hastings (MH) approaches. Popular examples include SparseLDA [28], AliasLDA [15], and LightLDA [29]. Remarkably, LightLDA uses Walker's alias method [23] and MH techniques to achieve an amortized $O(1)$ time complexity,[2] which is the theoretical lower bound. These fast per-token sampling methods have been implemented in distributed systems to deal with large data and model sizes. Examples include parameter servers (PS) [1, 16] and model parallel systems [26, 29].

Though previous algorithms such as LightLDA have a time complexity at the theoretical lower bound, it is still far from being optimal. According to the throughput (∼4M token/s per machine) reported in its paper [29], each token takes about 10,000 CPU cycles per iteration to sample, making it not fundamentally faster than other non-$O(1)$ algorithms when the number of topics $K$ is moderate (e.g., from $1,000$ to $10,000$). The large constant attributes to *cache misses*, which are orders of magnitude slower than cache hits (See Table 2). These cache misses arise from random memory accesses to the large topic count matrix, which can be in the order of tens of Gigabytes (GB) and too large to fit in the cache. Generally, to ensure a good cache hit rate, the memory access of a program must fall into one of the two patterns: 1) *continuous memory access*: a single memory access will load a 64-byte *cache line* to the cache, thus if the program is accessing continuous 4-byte integers, after the initial cache misses, there will be 15 cache hits, meanwhile CPU will prefetch the next cache line to avoid cache miss; 2) *random access within small scope*: if the scope of random access is limited to the Megabyte (MB) scale, which fits in the L3 cache, the speed will be 5x+ faster than the main memory. The speedup can be even larger if the scope of random access is smaller (e.g., fitting in L1 or L2 cache).

Table 1 summarizes some typical algorithms for LDA. VB and CGS are vanilla $O(K)$-per-token implementations, which have very good locality for the continuous memory access, but the speed degrades very quickly for even a thousand of topics. SparseLDA and AliasLDA are sparsity-aware fast-sampling algorithms. The locality of SparseLDA is good because it is still continuous, while the locality of AliasLDA is worse for the need to randomly access the topic count matrix. For SparseLDA and its implementations Yahoo!LDA and PS, the time complexity approaches $O(K)$ at the beginning, making them slow to even finish the first iteration for large models (e.g., $K = 10^6$). LightLDA can handle a very large number of topics due to its $O(1)$ time complexity, but the locality is poor because the access is random. Furthermore, it requires an $O(VK_w)$ per iteration time to generate alias table, where $V$ is the vocabulary size and $K_w$ is the average number of topics per word, which might dominate the cost for large models and small data. Previous parallel implementations also have data races for accessing the topic count matrix, resulting in additional complication and overhead for multi-threaded implementations. SVI, SGRLD and BIDMach [7] are stochastic algorithms; however they are only suitable for very small $K$ and $V$, due to the validity of their data redundancy assumption as mentioned before and the $O(KV)$ cost of updating the model. Stochastic algorithms are embarrassingly parallel, but the batch size must be large for good speedup. There are some further works on GPU acceleration for LDA [7, 30]. However, the algorithm they accelerate is $O(K)$. GPU acceleration for large models with up to millions of topics is still very challenging.

Table 2: Configuration of the memory hierarchy in a Intel Ivy Bridge CPU. L1D = L1 data cache.

| | L1D | L2 | L3 | Main memory |
|---|---|---|---|---|
| Latency (cycles) | 5 | 12 | 30 | 180+ |
| Size (per core) | 32K | 256K | 2.5M | 10-100G+ |

In this paper, we propose WarpLDA[3] – a novel $O(1)$ sampling algorithm for LDA, with specially designed proposal distributions and a delayed update strategy. Compared to existing methods, WarpLDA has the following advantages: 1) *Efficiency and scalability*: WarpLDA access memory either continuously or within a small scope, so the cache hit rate is high. We also develop multi-threading and MPI-based distributed implementations for large-scale applications, which supports up to 128 machines. 2) *Simplicity*: WarpLDA does not have any complicated modules such as alias tables, hybrid data structures, locks, or parameter servers, making it easy to understand and implement; 3) *Robustness*: With little tuning, WarpLDA is consistently several times faster than other algorithms, under settings varying from thousands of documents to hundreds of millions of documents, from hundreds of topics to millions of topics, from personal computer to supercomputer.

**Outline:** Section 2 introduces some basics for LDA and LightLDA. Section 3 introduces the WarpLDA algorithm. Section 4 provides some implementation details. Experiments are presented in Section 5. Section 6 concludes.

## 2. BASICS OF LDA

[2] $O(1)$ time complexity means the time sampling the latent variable related to a token is irrespective with a model size.

[3] The name comes after Warp Drive, the hypothetical faster-than-light propulsion system in Star Trek.

Latent Dirichlet Allocation [4] is a hierarchical Bayesian model, which models the distribution of a word as a mixture of topic distributions, with the shared mixing proportion for words within the same document. Let $\mathbf{W} = \{\mathbf{w}_d\}_{d=1}^{D}$ be a set of $D$ documents, where $\mathbf{w}_d = \{w_{dn}\}_{n=1}^{L_d}$ denotes the set of $L_d$ words in document $d$. Let $V$ denote the number of unique tokens, and $K$ denote the number of topics. The generative process of LDA is

For each topic $k \in \{1, \ldots, K\}$: $\boldsymbol{\phi}_k \sim \mathrm{Dir}(\boldsymbol{\alpha})$

For each document $d \in \{1, \ldots, D\}$: $\boldsymbol{\theta}_d \sim \mathrm{Dir}(\boldsymbol{\beta})$

For each position $n \in \{1, \ldots, L_d\}$

$z_{dn} \sim \mathrm{Mult}(\boldsymbol{\theta}_d)$
$w_{dn} \sim \mathrm{Mult}(\boldsymbol{\phi}_{z_{dn}})$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the parameters of Dirichlet priors, $\boldsymbol{\phi}_k$ is the $V$-dim word distribution of topic $k$, $\boldsymbol{\theta}_d$ is the $K$-dim topic mixing proportion of document $d$, and $z_{dn}$ is the topic assignment of the word $w_{dn}$. We let $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1 \cdots \boldsymbol{\phi}_K]$ be the $K \times V$ topic matrix. We further denote $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_d\}$ and $\mathbf{Z} = \{\mathbf{z}_d\}$, where $\mathbf{z}_d = \{z_{dn}\}$.

To train an LDA model, one must infer the posterior distribution of latent variables $(\boldsymbol{\Theta}, \boldsymbol{\Phi}, \mathbf{Z})$ given $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{W})$ or its marginal version. Unfortunately, the exact posterior inference is intractable. Thus approximate techniques including variational Bayes and Markov Chain Monte Carlo (MCMC) methods are adopted. As mentioned before, Collapsed Gibbs Sampling (CGS) [11] is the most popular because of its simplicity and the availability of fast sampling algorithms. Given $(\mathbf{W}, \boldsymbol{\alpha}, \boldsymbol{\beta})$, CGS integrates out $(\boldsymbol{\Theta}, \boldsymbol{\Phi})$ by conjugacy and samples $\mathbf{Z}$ from the following conditional distribution

$$\begin{aligned} &p(z_{dn} = k | \mathbf{Z}_{\neg dn}, w_{dn} = w, \mathbf{W}_{\neg dn}) \\ &\quad \propto (C_{dk}^{\neg dn} + \alpha_k) \frac{C_{wk}^{\neg dn} + \beta_w}{C_k^{\neg dn} + \bar{\beta}}, \end{aligned} \tag{1}$$

where $C_{dk} = \sum_{n=1}^{L_d} \mathbb{I}(z_{dn} = k)$ is the number of tokens that are assigned to topic $k$ in document $d$; $C_{wk} = \sum_{d=1}^{D} \sum_{n=1}^{L_d} \mathbb{I}(z_{dn} = k, w_{dn} = w)$ is the number of times that word $w$ has topic $k$; $C_k = \sum_d C_{dk} = \sum_w C_{wk}$; and $\bar{\beta} = \sum_v \beta_v$. The superscript or subscript $^{\neg dn}$ stands for excluding $(z_{dn}, w_{dn})$ from the corresponding count.

By sampling on a collapsed space, CGS often converges faster than a standard Gibbs sampler on a space with all the variables. The time complexity to sample a token with Eq. (1) is $O(K)$ by a vanilla enumeration of all $K$ possible topic assignments, which can be too expensive for large-scale applications where $K$ can be in the order of $10^6$. Many works exist to speed this up, including SparseLDA [28], AliasLDA [15] and LightLDA [29]. We first introduce some general algorithms, which are used by both previous works and our algorithms, then introduce existing works. To save space, we only introduce LightLDA, which is the fastest (See Table 1) and most relevant to WarpLDA.

## 2.1 General algorithms

**Metropolis-Hastings (MH):** Let $p(x)$ be a (unnormalized) target distribution. We consider the nontrivial case that it is hard to directly draw samples from $p(x)$. Metropolis-Hastings (MH) methods construct a Markov chain with an easy-to-sample *proposal distribution* $q(\hat{x}_t | x_{t-1})$ at each step $t$. Starting with an arbitrary point $x_0$, MH repeatedly generates samples from the proposal distribution $\hat{x}_t \sim q(\hat{x}_t | x_{t-1})$, and accepts the proposal with an *acceptance rate* $\pi_t = \min\{1, \frac{p(\hat{x}_t) q(x_{t-1} | \hat{x}_t)}{p(x_{t-1}) q(\hat{x}_t | x_{t-1})}\}$. Under some mild conditions, it is guaranteed that $p(x_t)$ converges to $p(x)$ as t $\to \infty$, regardless of $x_0$ and $q(\hat{x}|x)$ (Alg. 1). In LDA, $p(x)$ is the distribution of topic assignment in Eq. (1), whose sampling is $O(K)$, and $q(\hat{x}_t | x_{t-1})$ is a cheap approximation of $p(x)$, as will be clear soon.

**Algorithm 1** Metropolis-Hastings algorithm

```
Require: Initial state $x_0$, $p(x)$, $q(\hat{x}|x)$, number of steps $T$
  for $t \leftarrow 1$ to $T$ do
    Draw $\hat{x} \sim q(\hat{x}|x_{t-1})$
    Compute the acceptance rate $\pi = \min\{1, \frac{p(\hat{x}) q(x_{t-1}|\hat{x})}{p(x_{t-1}) q(\hat{x}|x_{t-1})}\}$
    $x_t = \begin{cases} \hat{x} & \text{with probability } \pi \\ x_{t-1} & \text{otherwise} \end{cases}$
  end for
```

**Mixture of multinomial:** If a categorical distribution has the following form

$$p(x = k) \propto A_k + B_k,$$

it can be represented by a mixture of two distributions,

$$p(x = k) = \frac{Z_A}{Z_A + Z_B} p_A(x = k) + \frac{Z_B}{Z_A + Z_B} p_B(x = k),$$

where $Z_A = \sum_k A_k, Z_B = \sum_k B_k$ are the normalizing coefficient, and $p_A(x = k) = \frac{A_k}{Z_A}, p_B(x = k) = \frac{B_k}{Z_B}$ are the normalized mixture distributions. By introducing an auxiliary variable $u$, where $u \sim \mathrm{Bern}(\frac{Z_A}{Z_A + Z_B})$, and defining $p(x|u = 1) = p_A(x), p(x|u = 0) = p_B(x)$, one can confirm that $p(x)$ is a marginal distribution of $p(u)p(x|u)$. Therefore, a sample from $p(x)$ can be drawn via ancestral sampling:

Draw $u \sim \mathrm{Bern}(\frac{Z_A}{Z_A + Z_B})$

Draw $x \sim \begin{cases} p_A(x) & \text{If } u = 1 \\ p_B(x) & \text{If } u = 0 \end{cases}$

This principle is useful when $p_A(x)$ is a simple distribution to sample from and $\frac{Z_B}{Z_A + Z_B} \ll 1$, in which case we only need to draw samples from the simple $p_A(x)$ with a high probability.

## 2.2 LightLDA

LightLDA is an MH-based algorithm that speeds up drawing topic assignments from the target distribution in Eq. (1). Unlike traditional MH methods, LightLDA uses two proposal distributions

$$\begin{aligned} q^{\text{doc}}(z_{dn} = k) &\propto C_{dk}^{\neg dn} + \alpha_k \\ q^{\text{word}}(z_{dn} = k) &\propto \hat{\phi}_{wk}, \end{aligned}$$

where $\hat{\phi}_{wk} = \frac{C_{wk}^{\neg dn} + \beta_w}{C_k^{\neg dn} + \bar{\beta}}$ is the stale version of $\phi_{wk}$, and is computed and stored at the beginning of each iteration. The correct stationary distribution is guaranteed by alternating between the two proposal distributions [29]. The rationale of using two proposal distributions instead of one is that a high probability state of $p(z_{dn} = k | \mathbf{Z}_{\neg dn}, w_{dn} = w, \mathbf{W}_{\neg dn})$ is also the high probability state of at least one of $q^{\text{doc}}$ and $q^{\text{word}}$. Hence samples from the two proposal distributions can cover most of the high probability states in the target distribution.

**Sampling from $q^{\text{word}}$:** LightLDA applies Walker's alias method [23] to draw samples from the proposal $q^{\text{word}}$, which is a categorical distribution. The time complexity of generating a single sample is $O(1)$, with an auxiliary structure called alias table, which can be precomputed in $O(K)$ time. LightLDA periodically computes and stores $\hat{\phi}_{wk} = \frac{C_{wk}^{\neg dn}+\beta_w}{C_k^{\neg dn}+\bar{\beta}}$, and rebuilds the alias table.

**Sampling from $q^{\text{doc}}$:** $q^{\text{doc}}$ can be represented as a mixture of multinomial:

$$q^{\text{doc}}(z_{dn}=k) = \pi A_k + (1-\pi)B_k$$

where $\pi = \frac{L_d-1}{L_d-1+\bar{\alpha}}$, $A_k = \frac{1}{L_d-1}C_{dk}^{\neg dn}$, $B_k = \frac{\alpha_k}{\bar{\alpha}}$ and $\bar{\alpha} = \sum_k \alpha_k$. As mentioned before, one only needs to draw samples from $p_A(z_{dn})$ and $p_B(z_{dn})$, where $p_A(z_{dn}=k) \propto A_k$ and $p_B(z_{dn}=k) \propto B_k$. Drawing a sample from $p_A(z_{dn})$ can be done in $O(1)$ by randomly picking a topic assignment $z_{di}$ in this document $d$, since $A_k \propto C_{dk}^{\neg dn}$ is just a count of $z_{dn}$. Samples from $p_B(z_{dn})$ can be drawn in $O(1)$ with an alias table.

**Acceptance rate** Let $k$ be the old topic assignment and $k'$ be the topic assignment generated from the proposal distribution. Then, the acceptance rate for sampling from $q^{\text{word}}$ is

$$\pi_{k\to k'}^{\text{word}} = \min\{1, \frac{C_{dk'}^{\neg dn}+\alpha_{k'}}{C_{dk}^{\neg dn}+\alpha_k}\frac{C_{wk'}^{\neg dn}+\beta_w}{C_{wk}^{\neg dn}+\beta_w}\frac{C_k^{\neg dn}+\bar{\beta}}{C_{k'}^{\neg dn}+\bar{\beta}}\frac{\hat{\phi}_{wk}}{\hat{\phi}_{wk'}}\}.$$

Similarly, the acceptance rate for sampling from $q^{\text{doc}}$ is

$$\pi_{k\to k'}^{\text{doc}} = \min\{1, \frac{C_{wk'}^{\neg dn}+\beta_w}{C_{wk}^{\neg dn}+\beta_w}\frac{C_k^{\neg dn}+\bar{\beta}}{C_{k'}^{\neg dn}+\bar{\beta}}\}. \qquad (2)$$

Despite being an $O(1)$ algorithm, the speed of LightLDA is not satisfactory. According to Yuan et. al [29], the throughput of LightLDA is less than 4M token/s per machine, which is approximately 10,000 CPU cycles per token per iteration, or 600 CPU cycles per token per iteration per MH step, with 16 MH steps. The high per-token overhead attributes to the random access to the topic term count matrix (i.e., $C_{wk'}$ and $C_{wk}$ in Eq. (2)). Because the size of the model can be tens of Gigabytes, almost all of the accesses will result in cache misses and main memory accesses, which is notoriously slow (See Table 2). To mitigate the problem, Yuan et. al use a large $MH$ (e.g., 16) to improve the locality. Since the convergence speed only improves sublinearly as $MH$ grows, this is not a fundamental treatment.

In this paper, we design WarpLDA to minimize cache misses. WarpLDA scans the data either document by document or word by word, eliminating random accesses to the large topic term count matrix $C_{wk}$. We also carefully design the layout of data so that the scope of random access is comparable to the size of the L3 cache, as detailed below.

# 3. WARPLDA

WarpLDA is a novel MH-based algorithm which utilizes two techniques: 1) delayed update: the update of $C_{dk}$ and $C_{wk}$ are delayed until the end of the iteration; and 2) simpler proposal distribution. With these we are able to derive a simple and efficient $O(1)$ sampling algorithm, which is consistently faster than LightLDA.

## 3.1 Data representation

The basic element of WarpLDA is *token*. We define a token as a triple of a word $w_{dn}$, its topic assignment $z_{dn}$, and $MH$ samples $\mathbf{z}'_{dn} \in \{1,\dots,K\}^{MH}$ from the proposal distribution, where $MH$ is a constant for the length of the $MH$ chain. We define $\mathbf{x}_d := \{(w_{dn}, z_{dn}, \mathbf{z}'_{dn})\}_{n=1}^{L_d}$ to be the vector of all tokens in a document, and define $\mathbf{y}_w := \{(w_{dn}, z_{dn}, \mathbf{z}'_{dn}) | \forall d, n, \text{s.t. } w_{dn} = w\}$ to be the vector of all tokens whose words are $w$, define the length of $\mathbf{y}_w$ as $L_w$. For a token $t$, let $t.w, t.k$ and $t.\mathbf{k}'$ be its word, topic assignment, and samples from proposal distributions respectively. $\mathbf{y}_w.k$ is a convenient notation of the vector $\{t.k | \forall t \in \mathbf{y}_w\}$.

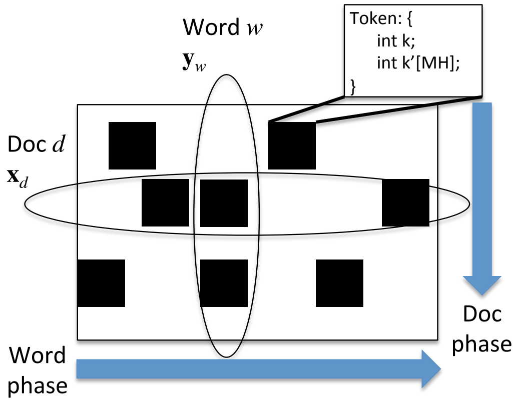


Figure 1: Data representation of WarpLDA.

WarpLDA views the data as a $D \times V$ *sparse matrix* $\mathbf{X}$, where a token $t \in \mathbf{x}_d$ is stored at the position $X_{d,t.w}$ (See Fig. 1). Because a word can occur more than once in one document, the definition of the sparse matrix is extended so that it can store multiple tokens in a single entry. In practice, $t.w$ needs not to be stored because it is the coordinate. We can observe some symmetric structure of the sparse matrix. For example, each row of the matrix $\mathbf{X}$ is a document with $\mathbf{x}_d$ as a vector of all the tokens within this row, and each column of the matrix $\mathbf{X}$ is a word with $\mathbf{y}_w$ as the vector of all the tokens within this column.

Besides the data matrix $\mathbf{X}$, WarpLDA also stores the topic count vector $C_k$. Unlike previous work such as Yahoo!LDA and LightLDA, WarpLDA does not store the document topic counts $C_{dk}$ and $C_{wk}$, because it scans documents and words sequentially, the counts can be computed on the fly (Section 3.4).

## 3.2 Delayed update

In WarpLDA, the counts $C_{dk}$ and $C_{wk}$ are not updated immediately after the change of a single $z_{dn}$, but are updated after finishing the iteration of sampling $\mathbf{Z}$. This delayed update scheme decouples the computation related with documents and the computation related with words, and is the key for our algorithm to be efficient. The rationale of delayed update is that the counts $C_{dk}$ and $C_{wk}$ are changing slowly, so a slightly staled count will not impact the convergence much. Delayed update has been widely used in existing algorithms [20, 1].

We further provide a more rigorous justification of the delayed update, by viewing it as a Monte-Carlo EM algorithm for MAP estimation of LDA. The goal is to get a *maximum a posterior* (MAP) estimation of LDA

$$\hat{\boldsymbol{\Theta}}, \hat{\boldsymbol{\Phi}} = \operatorname*{argmax}_{\boldsymbol{\Theta},\boldsymbol{\Phi}} \log p(\boldsymbol{\Theta}, \boldsymbol{\Phi} | \boldsymbol{\alpha}', \boldsymbol{\beta}', \mathbf{W}),$$

where $\boldsymbol{\alpha}'$ and $\boldsymbol{\beta}'$ are the Dirichlet hyper-parameters. Let $q(\mathbf{Z})$ be a variational distribution. Then, by Jensen's inequality, we get

$$\begin{aligned}\log p(\boldsymbol{\Theta}, \boldsymbol{\Phi} | \boldsymbol{\alpha}', \boldsymbol{\beta}', \mathbf{W}) \geq & \mathbb{E}_q[\log p(\mathbf{W}, \mathbf{Z} | \boldsymbol{\Theta}, \boldsymbol{\Phi}) - \log q(\mathbf{Z})] \\ & + \log p(\boldsymbol{\Theta} | \boldsymbol{\alpha}') + \log p(\boldsymbol{\Phi} | \boldsymbol{\beta}') \\ = & \mathcal{J}(\boldsymbol{\Theta}, \boldsymbol{\Phi}, q(\mathbf{Z})).\end{aligned}$$

We can derive an EM algorithm, where in the E-step we maximize $\mathcal{J}$ with respect to the variational distribution $q(\mathbf{Z})$ and in the M-step we maximize $\mathcal{J}$ with respect to $(\boldsymbol{\Theta}, \boldsymbol{\Phi})$, keeping $q(\mathbf{Z})$ fixed. One can prove that the optimal solution at E-step is $q(\mathbf{Z}) =$

$p(\mathbf{Z}|\mathbf{W}, \mathbf{\Theta}, \mathbf{\Phi})$ without further assumption on $q$. Unfortunately the expectation in $\mathcal{J}$ is intractable. A Monte-Carlo EM estimates the expectation via samples and has the following steps:

- E-step: sample $\mathbf{Z}$ from $p(\mathbf{Z}|\mathbf{W}, \mathbf{\Theta}, \mathbf{\Phi})$
- M-step: maximize $\log p(\mathbf{W}, \mathbf{Z}|\mathbf{\Theta}, \mathbf{\Phi}) + \log p(\mathbf{\Theta}|\boldsymbol{\alpha}') + \log p(\mathbf{\Phi}|\boldsymbol{\beta}')$

By setting $\partial\mathcal{J}/\partial\mathbf{\Theta}$ and $\partial\mathcal{J}/\partial\mathbf{\Phi}$ to zero, we get the update rules:

$$\hat{\theta}_{dk} \propto C_{dk} + \alpha'_k - 1 \tag{3}$$

$$\hat{\phi}_{wk} = \frac{C_{wk} + \beta'_w - 1}{C_k + \bar{\beta}' - V} \tag{4}$$

which are executed at the end of each iteration of sampling $\mathbf{Z}$ (i.e., after E-step). A Gibbs sampler draws each $z_{dn}$ according to the local conditional:

$$p(z_{dn} = k|w_{dn} = w, \mathbf{W}_{\neg dn}, \mathbf{\Theta}, \mathbf{\Phi}) \propto \theta_{dk}\phi_{wk}. \tag{5}$$

To set connection with the CGS with delayed update, let $\alpha_k = \alpha'_k - 1, \beta_w = \beta'_w - 1$, and plug Eq.s (3, 4) in Eq. (5). Then, we have

$$p(z_{dn} = k|\mathbf{Z}_{\neg dn}, w_{dn} = w, \hat{\mathbf{\Theta}}, \hat{\mathbf{\Phi}}) \propto (C_{dk} + \alpha_k)\frac{C_{wk} + \beta_w}{C_k + \bar{\beta}}. \tag{6}$$

Note that $C_{dk}$ and $C_{wk}$ in Eq. (6) are updated at the end of each iteration of sampling $\mathbf{Z}$ because they actually come with $\mathbf{\Phi}$ and $\mathbf{\Theta}$. Comparing Eq. (6) with Eq. (1), we conclude that CGS with delayed update actually finds an MAP estimate with hyper-parameters $\alpha'_k = \alpha_k + 1, \beta'_w = \beta_w + 1$.[4] Asuncion et al. [3] have shown that this MAP solution is almost identical with the solution of CGS.

## 3.3 Simple proposal distribution

Despite one can draw samples from an arbitrary categorical distribution in amortized $O(1)$ time with Walker's alias method, it increases the overhead for building the alias table. Moreover, if the number of tokens is small and the model is large, it is challenging to amortize the overhead of building alias table, which does not conform with our need of building a sampler that is consistently fast under various settings. We eliminate all needs for alias sampling in WarpLDA by the careful choice of proposal distributions

$$q^{\text{doc}}(z_{dn} = k) \propto C_{dk}^{\neg dn} + \alpha$$
$$q^{\text{word}}(z_{dn} = k) \propto C_{wk}^{\neg dn} + \beta_w. \tag{7}$$

Here, we assume symmetric Dirichlet prior, i.e., $\forall k, \alpha_k = \alpha$. Although Wallach et al. [24] show that using asymmetric prior can lead to better perplexity results, it is non-trivial to learn the prior for a very large model efficiently and most times people use a symmetric prior due to its simplicity. Because the prior is symmetric, the topic counts $C_k$ are close to each other. Fig. 2 provides an empirical study, where we can see that the minimum and maximum of $C_k$ are within the same level of magnitude. So after normalization, the WarpLDA proposal $\frac{1}{Z}(C_{wk}^{\neg dn} + \beta_w)$ is not much different with the LightLDA proposal $\frac{1}{Z}\frac{C_{wk}^{\neg dn} + \beta_w}{C_k + \bar{\beta}}$, but we eliminate the need for alias sampling, as explained below.

Note that $q^{\text{doc}}$ and $q^{\text{word}}$ have the same form, so we can draw samples from them similarly. For simplicity we only discuss the $q^{\text{word}}$ case, and the $q^{\text{doc}}$ case can be derived by replacing $C_{wk}^{\neg dn}$ with $C_{dk}^{\neg dn}$, $L_d$ with $L_w$, $\alpha_k$ with $\beta_w$, and $\mathbf{y}_w^{\neg dn}.k$ with $\mathbf{z}_d^{\neg dn}$.

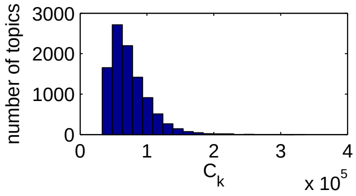


Figure 2: Distribution of $C_k$. The model is learnt from the PubMed dataset, 10000 topics, $\alpha = \beta = 0.01$.

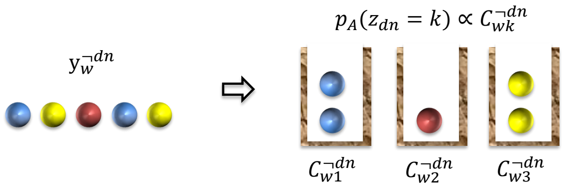


Figure 3: Graphic illustration of drawing a sample from $q^{\text{word}}$.

By the technique in Section 2.1, $q^{\text{word}}$ can be represented as a mixture of two distributions

$$p_A(z_{dn} = k) = \frac{1}{Z_A}C_{wk}^{\neg dn}, \;\; p_B(z_{dn} = k) = \frac{\beta_w}{Z_B},$$

with the mixing proportion $\frac{Z_A}{Z_A + Z_B}$, where $Z_A = \sum_k C_{wk}^{\neg dn} = L_w - 1, Z_B = K\beta_w$. Then, an ancestral sampler can be used. Since $p_B$ is a uniform distribution, a sample can be easily obtained. The sample from $p_A$ can be obtained as follows, without the need of an alias table.

Given that $C_{wk}^{\neg dn}$ is the count of $\mathbf{y}_w^{\neg dn}.k$, drawing a sample from the distribution $p_A$ can be done by randomly selecting an index $i$ (excluding $n$) and get the corresponding topic $y_{wi}.k$. Fig. 3 illustrates a simple example, where $L_w = 6$ and $\mathbf{y}_w^{\neg dn}.k$ is shown in Fig. 3(left). We use different colors to denote different topics. Then, the corresponding counts $C_{wk}^{\neg dn}$, or the (unnormalized) distribution $p_A$, is shown in Fig. 3(right). It is clear that sampling from $p_A$ is equivalent to sampling uniformly an index $i$ (e.g., $i = 4$) and getting the topic $y_{wi}$, which is topic 1 in this case when $i = 4$.

With our proposals $q^{\text{doc}}$ and $q^{\text{word}}$, the corresponding acceptance rates of WarpLDA are as follows:

$$\pi_{k \to k'}^{\text{doc}} = \min\{1, \frac{C_{wk'}^{\neg dn} + \beta_w}{C_{wk}^{\neg dn} + \beta_w}\frac{C_k^{\neg dn} + \bar{\beta}}{C_{k'}^{\neg dn} + \bar{\beta}}\}$$
$$\pi_{k \to k'}^{\text{word}} = \min\{1, \frac{C_{dk'}^{\neg dn} + \alpha_{k'}}{C_{dk}^{\neg dn} + \alpha_k}\frac{C_k^{\neg dn} + \bar{\beta}}{C_{k'}^{\neg dn} + \bar{\beta}}\}. \tag{8}$$

## 3.4 Organizing the computation

With delayed update, we can decouple the access to document specific data and the access to word specific data to improve locality and restrict random access within a small scope.

Specifically, WarpLDA processes each token by starting from an arbitrary initial topic $k^0$, sequentially draws samples from $q^{\text{doc}}$ or $q^{\text{word}}$ in Eq. (7), and computes acceptance rates using Eq. (8). Because of delayed update, the counts $C_{dk}^{\neg dn}$ and $C_{wk}^{\neg dn}$ do not change, we can therefore first draw $MH$ samples from the proposal distribution and then simulate the chain by computing the corresponding acceptance rate. Thus, the sampling is equivalent to a two phase procedure:

- word phase: compute $\pi^{\text{doc}}$ and draw samples from $q^{\text{word}}$;

[4] There is actually one subtle difference: we are not excluding $z_{dn}$ itself from the count, however that impacts little since counts are often much larger than one.

**Algorithm 2** The WarpLDA algorithm (word phase).

```
for $w \leftarrow 1$ to $V$ do
    // Compute $C_{wk}$ on the fly
    $C_{wk} \leftarrow 0, \forall k$
    for token $k \in \mathbf{y}_w$ do
        $++C_{wk}$
    end for
    // Simulate $q^{\text{doc}}$ chain with samples from last iteration
    for each token $t \in \mathbf{y}_w$ do
        for $i \leftarrow 1$ to $MH$ do
            $\pi \leftarrow \min\{1, \frac{C^{\neg dn}_{w,t.k'^i}+\alpha}{C^{\neg dn}_{w,t.k}+\alpha} \frac{C^{\neg dn}_{t.k}+\beta V}{C^{\neg dn}_{t.k'^i}+\beta V}\}$
            $t.k \leftarrow t.k'^i$ with probability $\pi$
        end for
    end for
    // Draw samples from $q^{\text{word}}$
    for each token $t \in \mathbf{y}_w$ do
        for $i \leftarrow 1$ to $MH$ do
            $t.k'^i \leftarrow \begin{cases} y_{w,\text{unidrnd}(L_w)}.k & \text{with probability } \frac{L_w-1}{L_w-1+K\beta} \\ \text{unidrnd}(K) & \text{otherwise} \end{cases}$
        end for
    end for
end for
```

- document phase: compute $\pi^{\text{word}}$ and draw samples from $q^{\text{doc}}$.

The word phase only requires word-specific data $\mathbf{y}_w$ and $C_{wk}$, and can be implemented by scanning $\mathbf{y}_w$ by $w$ with $C_{wk}$ computed on the fly. The document phase is similar that it requires only $\mathbf{x}_w$ and $C_{dk}$, and the latter is computed on the fly while scanning $\mathbf{x}_w$.

The scope of random access of this approach is small. In the word phase, $C_{wk}$ is random accessed for computing $\pi^{\text{doc}}$, and $\mathbf{y}_w$ is random accessed for drawing samples from $q^{\text{word}}$, where $C_{wk}$ is a $O(K)$ *vector* and $\mathbf{y}_w$ is a $O(L_w)$ vector. Thus, the scope of random access is $O(\max\{K, L_w\})$ and similarly, in the document phase the scope of random access is $O(\max\{K, L_d\})$. These scopes are significantly smaller than that of LightLDA, which randomly accesses the $O(KV)$ *matrix* $C_{wk}$. Algorithm 2 is the pseudocode of WarpLDA, which also shows the simplicity of WarpLDA: no alias tables or complicated data structures. These simplicities make WarpLDA efficient, simple to implement, and easy to be optimized.

# 4. SYSTEM

System side optimization is required to maximize the efficiency and scalability of WarpLDA. In this section, we present cache aware optimizations and a scalable implementation based on MPI.

## 4.1 Single Machine

We start with the single machine version with optimizations for both single thread and multi-threading implementations.

### *4.1.1 Single Thread*

As stated in Section 3.4, WarpLDA alternately performs the word and document phases, so that the scope of random access is small. One question left unanswered: *how to store the data so that we can scan the data by both document (row) and word (column) efficiently*? So far, we assume that in both document and word phases the tokens with each document / word are continuous, but some careful treatments are required to ensure this.

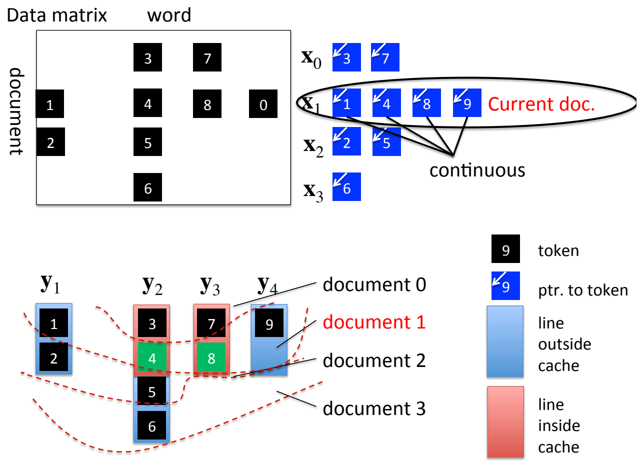


Figure 4: Storage format for WarpLDA. The tokens from $\mathbf{y}_w$ are cached for better hit rate (Please see text for details).

One possible solution is to store both $\mathbf{x}_d$ and $\mathbf{y}_w$ in the memory, and scan $\mathbf{x}_d$ for the document phase and $\mathbf{y}_w$ for the word phase. While scanning, the tokens within each document / word are continuous, so the locality is good. However, since $\mathbf{x}_d$ and $\mathbf{y}_w$ are different copies of the data matrix, one must update $\mathbf{y}_w$ with $\mathbf{x}_d$ after the document phase and vice versa, which requires an extra pass through data.

In WarpLDA, we only store $\mathbf{y}_w$, so the word phase always accesses continuous tokens, which is memory efficient. For the document phase, we store pointers to tokens in $\mathbf{x}_d$ instead of actual tokens and access the tokens through pointers, so that $\mathbf{x}_d$ and $\mathbf{y}_w$ are updated simultaneously. An access to a specific token $x_{dn}$ in the document phase would require an access of the pointer followed by an access of the actual token in $\mathbf{y}_w$, where the latter access may be discontinuous.

Despite the discontinuity, the document phase is still cache efficient. Firstly, the access to the pointer matrix is continuous, hence efficient. Let $t.d$ be the document number, to which the token $t$ belongs, we can sort $\mathbf{y}_w$ by $t.d$ in the preprocessing stage. In each document phase, documents are considered sequentially, so the access of $\mathbf{y}_w$ is continuous. Once a token $y_{wi}$ is accessed, the next few tokens in the same cache line $y_{wi+1}, y_{wi+2}, \dots, y_{wi+m}$ will be also kept in cache, so that the next $m$ accesses to $\mathbf{y}_w$ will hit. If the cache is large enough to fit one cache line (64 Bytes) for each of the most high frequency words, the access to $\mathbf{y}_w$ for all these words enjoys a good cache hit rate. If the L3 cache is 30MB, it is enough to store 30MB/64B=491520 cache lines, i.e., supporting 491,520 high frequency words. A natural corpus often satisfies power law [14], that is, a small fraction of words covers most of the occurrences, so the actual number of high frequency words is much smaller.

Fig. 4 is a concrete example, where the small squares are tokens, with a unique id on each, to make the presentation clear. The actual tokens are stored in $\mathbf{y}_w$, and $\mathbf{x}_d$ is an array of pointers to the actual tokens. The dashed red lines separate the tokens of each document in $\mathbf{y}_w$. We also assume a cache line can fit two tokens, shown by the tall red or blue rectangle. While visiting document 0, tokens 3 and 7 (i.e., $y_{21}$ and $y_{31}$) are accessed, so $y_{22}$ and $y_{32}$ are also in the cache because they are in the same cache line of the accessed tokens. Hence, the access to tokens 4 and 8 while scanning document 1 will enjoy a cache hit.

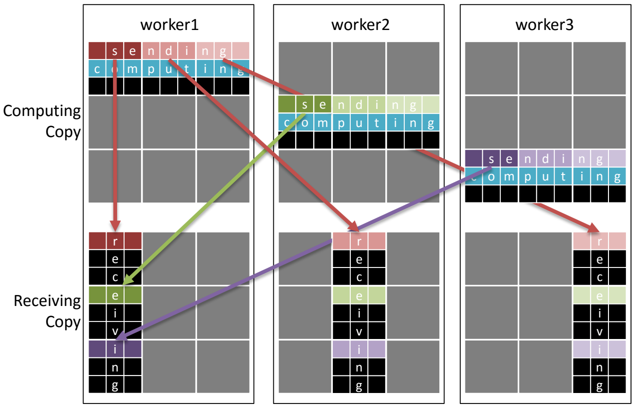


Figure 5: Data partition and communication of multiple workers.

When $K$ is large, it is likely to have $L_d \ll K$ and $L_w \ll K$ for low frequency words. It is more effective to use a hash table rather than a dense array for the counts $C_{dk}$ and $C_{wk}$, because the size of a hash table is smaller. We choose an open addressing hash table with linear probing for hash collisions. The hash function is a simple `and` function, and the capacity is set to the minimum power of 2 that is larger than $\min\{K, 2L_d\}$ or $\min\{K, 2L_w\}$. We find that the hash table is almost as fast as a dense array even when $L_d > K$. Although LightLDA [29] also uses a hash table to store the counts, it is mainly for reducing the storage overhead instead of improving cache locality, since LightLDA randomly accesses the entire count matrix $C_{wk}$ instead of a single row.

### 4.1.2 *Multi-threading and NUMA*

WarpLDA is embarrassingly parallel: different threads work at disjoint sets of tokens, so there are no data races. In contrast, traditional frameworks such as Yahoo!LDA [1] and LightLDA [29] need to update the global $C_{wk}$ matrix in parallel, and require extra treatment such as locks. The computation for $C_k$ follows a classical *reduce* operation: we store a local count vector in each thread, and aggregate them at the end of each iteration.

There is one subtlety here. We mentioned that in the document phase, the required cache size to ensure a good cache hit rate is $64V$. When we access documents in parallel, the access to documents is no longer strictly sequential. If the fastest thread is working on document $D_2$ and the slowest thread is working on document $D_1$, all tokens in document $[D_1, D_2]$ should be in cache to ensure a good hit rate; but this requires an additional $(D_2 - D_1)L_d$ storage, where $L_d$ is typically a few hundreds. To minimize $D_2 - D_1$, i.e., make all threads to work within a small range of documents, we assign documents to threads in a round-robin fashion, that is, document $d$ is assigned to thread $d \bmod \Gamma$, where $\Gamma$ is the number of threads. Since the lengths of documents distribute uniformly, all the threads work roughly at the same pace.

Finally, we also note that modern computers have a non-uniform memory access (NUMA) pattern, where each main memory DIMM belongs to a specific CPU socket. If one CPU needs to access data in the memory belonging to another CPU, the data flows through another CPU, resulting in degraded performance. For better performance, we partition $\mathbf{y}_w$ by column and $\mathbf{z}_d$ by row, and bind each slice to a different CPU.

## 4.2 Distributed implementation with MPI

For distributed training, we first split the $D \times V$ data matrix into *$P \times P$ partitions*, where $P$ is the number of MPI workers. In the document phase, each worker has all the $P$ partitions in the same row in its memory; in the word phase, each worker has all the $P$ partitions in the same column in its memory. In each document / word phase, each worker first scans the documents / words in its memory, then sends the blocks to workers who are in charge of the blocks in next word / document phase.

By overlapping communication and computation, one can utilize the CPU and network simultaneously, and can achieve an up to 2x speedup of program when the time for communication is equal to the time of computation. This technique is called pipelining and is widely used in large-scale LDA literature [1, 25]. For WarpLDA, we further divide each partition into *$B \times B$ blocks*, where $B \in [2, 10]$ is a small constant. During training, each of the blocks may be in one of the four states: 1) not started; 2) computing; 3) sending; 4) finished. In each document / word phase, workers scan its blocks by row or column, and each block can be immediately sent to the corresponding worker after the whole computation of row or column is finished. Two copies of the data must be stored—a computing copy that the workers are processing, and a receiving copy for preparing data for the next phase. For this communication pattern of WarpLDA, where each pair of nodes need to communicate, we use `MPI_Ialltoall`, which is a highly-optimized asynchronous implementation of this generalization for this communication pattern. Fig. 5 is an example, where there are 3 workers, the data is partitioned into $3 \times 3$ partitions, and each partition is further divided into $3 \times 3$ blocks. The current phase is the document phase. So each worker is assigned to the whole row of partitions. The workers simultaneously process the whole row of blocks and send previous finished blocks to the corresponding workers. The receiving copy is partitioned by column for the next word phase.

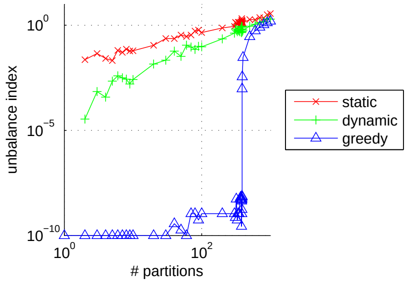


Figure 6: Comparison of partition strategies for the ClueWeb12 dataset.

To minimize the wait time, the number of tokens within each worker should be roughly the same, which implies that the sum of the number of tokens of all partitions within each row or column should be roughly the same. Note that the row / column can be treated independently: we first partition $\mathbf{x}_d$ and $\mathbf{y}_w$ as $P$ slices each, afterwards the $P \times P$ partition arises naturally by combining the $P$ slices of $\mathbf{x}_d$ and the $P$ slices of $\mathbf{y}_w$.

As we mentioned in Section 4.1.1, words from a natural corpus often satisfy the power law. For example, the most frequent word in the ClueWeb12 corpus occupies 0.257% of all occurrences, after removal of stop words. This is a very large proportion, because each slice may only have 1% of the tokens if we have 100 slices. We propose a greedy algorithm for more balanced partitioning. First, we sort all words by their frequency in a decreasing order. Then from the most frequent token, we put each token in the partition with the least number of tokens. Because there are

Table 3: Statistics of various datasets, where $T$ is the total number of words in the corpus.

| Dataset | $D$ | $T$ | $V$ | $T/D$ |
|---|---|---|---|---|
| NIPS | 1.5K | 1.9M | 12.4K | 1288 |
| 20NG | 19K | 2.2M | 60.7K | 116 |
| NYTimes | 300K | 100M | 102K | 332 |
| PubMed | 8.2M | 738M | 141K | 90 |
| ClueWeb12 | 150M | 55B | 1M | 360 |

many low frequency words (the long tail), this algorithm can produce very balanced results. We compared the *unbalance index* of our greedy algorithm with two randomized algorithms: *static* first random shuffle the words, then partition so that each partition has equal number of words; *dynamic* allows each partition has different number of words, but each slice is continuous. Unbalance index is defined as

$$\frac{\text{number of tokens in the largest partition}}{\text{average number of tokens of each partition}} - 1.$$

In the ideal case the unbalance index should be zero. Fig. 6 shows the experimental results on the ClueWeb12 corpus, where we can see that the greedy algorithm is much better than both randomized algorithms in hundreds of partitions.

Finally, while doing the word phase, the tokens are not continuous but continuous within each block, so an extra `memcpy` is required to make it continuous. Since `memcpy` accesses memory continuously, it is still cache efficient.

# 5. EXPERIMENTS

We now present empirical studies of WarpLDA, by comparing with two strong baselines LightLDA [29] and Yahoo!LDA [1]. LightLDA is the fastest sampler, while Yahoo!LDA is a scalable implementation of the exact SparseLDA. We compare with them in both time efficiency and the quality of convergence.

## 5.1 Datasets and Setups

Table 3 summarizes the datasets. NIPS and 20NG are two small-scale datasets. NYTimes and PubMed are standard datasets from the UCI machine learning repository [2]; they consist of news articles and biomedical literature abstracts, respectively. ClueWeb12 a 1/4 subset of the ClueWeb12 dataset, which is a large crawl of web pages.[5] While NYTimes and PubMed are already tokenized, we extract text from ClueWeb12 using JSoup, remove everything except alphabets and digits, convert letters to lower case, tokenize the text by space and remove stop words.

The experiments are done on the Tianhe-2 supercomputer. Each node is equipped with two Xeon E5-2692 CPUs (2 × 12 2.2GHz cores), and 64GB memory. Nodes are connected with InfiniBand, and single machine experiments are done with only one node.

We set the hyper-parameters $\alpha = 50/K$ and $\beta = 0.01$. We follow the previous work [1, 29] and measure the model quality by the widely adopted log joint likelihood (log likelihood in short):

$$\mathcal{L} = \log p(\mathbf{W},\mathbf{Z}|\boldsymbol{\alpha},\boldsymbol{\beta}) = \log \prod_d [\frac{\Gamma(\bar{\alpha})}{\Gamma(\bar{\alpha}+L_d)} \prod_k \frac{\Gamma(\alpha_k+C_{dk})}{\Gamma(\alpha_k)}] \prod_k [\frac{\Gamma(\bar{\beta})}{\Gamma(\bar{\beta}+C_k)} \prod_w \frac{\Gamma(\beta_w+C_{kw})}{\Gamma(\beta_w)}].$$

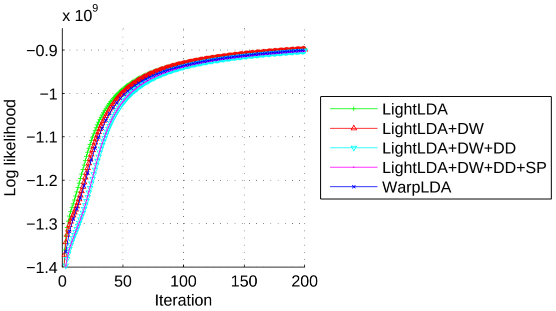


Figure 7: Impact of approximations on NYTimes, $K = 1000$.

## 5.2 Impact of Approximations

We show that delayed update and our simple proposal distributions have little impact on convergence speed, by comparing with LightLDA, which is similar with WarpLDA but updates the counts more frequently and uses a more complicated proposal distribution. We use the NYTimes corpus as an example and set $K = 1,000$. The compared algorithms are

- LightLDA: LightLDA with $MH = 1$. $C_{dk}$ is updated instantly, $C_{wk}$ is updated every 300 documents. (Theoretically $C_{wk}$ can be updated immediately as well, but practical implementations still have slightly delayed update for efficiency. [6])

- LightLDA+DW: LightLDA, $C_{dk}$ is updated instantly; the update of $C_{wk}$ is delayed to the end of each iteration.

- LightLDA+DW+DD: LightLDA, updates of both $C_{dk}$ and $C_{wk}$ are delayed to the end of each iteration.

- LightLDA+DW+DD+SP: LightLDA, updates of both $C_{dk}$ and $C_{wk}$ are delayed to the end of each iteration. Use WarpLDA's proposal distributions.

- WarpLDA: WarpLDA with $MH = 1$. This is similar as LightLDA+DW+DD+SP but the order of loops are different: for each token, LightLDA tries the document proposal and the word proposal before proceeding to the next token; but WarpLDA firstly tries the document proposal for every token, and then tries the word proposal for those tokens.

Fig. 7 shows the results. We can see that delayed update and simple proposal distributions result in little degrade of convergence speed, and WarpLDA is slightly faster than LightLDA+DW+DD+SP because of different orders of loops.

## 5.3 Convergence Results

We now analyze the convergence behaviors. Fig. 8 presents the results on the moderate sized corpora, including NYTimes (first two rows) and PubMed (last two rows), WarpLDA with a fixed $MH = 2$ is compared with LightLDA with the best $MH$ selected from $\{1, 2, 4, 8, 16, 32\}$, and Yahoo!LDA. Each algorithm is run for a fixed number of iterations. Yahoo!LDA is not included into comparison for PubMed $K = 100,000$ because it is too slow.

To have a full understanding, we consider a diverse range of evaluation metrics, including log-likelihood w.r.t the number of iterations (*1st column*), log-likelihood w.r.t running time (*2nd column*), the ratio of iteration number of LightLDA (or Yahoo!LDA) over

[5] `http://www.lemurproject.org/clueweb12.php/`

[6] Personal communication with the authors.

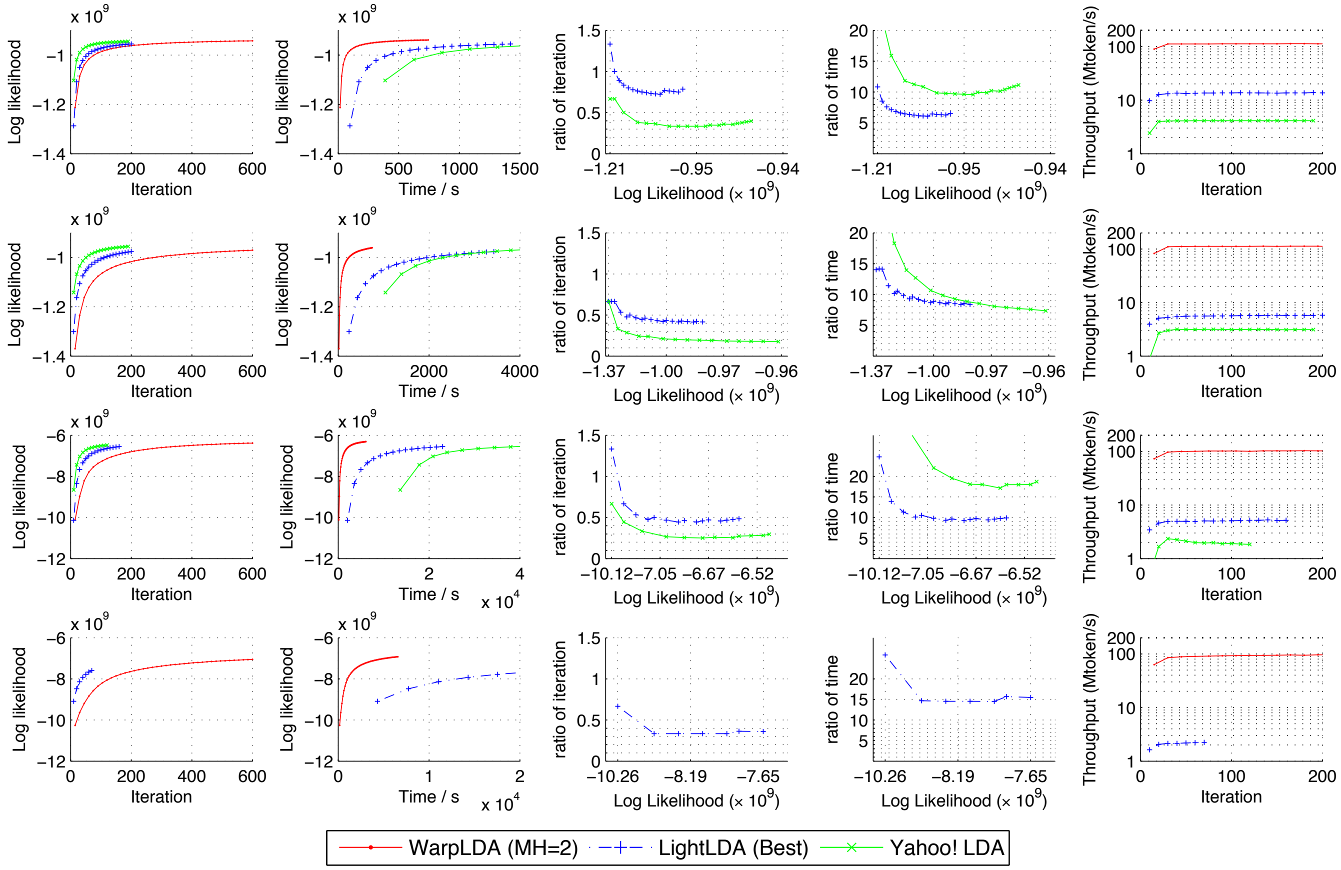


Figure 8: Convergence results on NYTimes (1st row: $K = 1,000$; 2nd row: $K = 10,000$) and PubMed (3rd row: $K = 10,000$; 4th row: $K = 100,000$). Each column corresponds to an evaluation metric (Please see text for details). The x-axis of column 3 and 4 are distorted for better resolution.

that of WarpLDA to get a particular log-likelihood (*3rd column*), the ratio of running time of LightLDA (or Yahoo!LDA) over that of WarpLDA to get a particular log-likelihood (*4th column*), and finally throughput w.r.t the number of iterations (*5th column*). From the results, we have the following observations.

- WarpLDA converges to the same log-likelihood as other baselines (1st and 2nd columns).
- WarpLDA converges slower than SparseLDA and LightLDA in terms of the number of iterations (1st column), but much faster in terms of running time (2nd column). Overall, WarpLDA is 6-15x faster than the second best algorithm (4th column).
- The speedup of WarpLDA over baseline algorithms comes from faster iterations: e.g., WarpLDA is 110M token/s on the PubMed dataset with $K = 10,000$, while the best LightLDA ($MH = 8$) is about 4M token/s (5th column). WarpLDA converges about 2.5x slower than LightLDA in terms of iteration number (3th column), as a result WarpLDA is about 10x faster than LightLDA in terms of running time (4th column).
- WarpLDA is cache efficient: converting the 110M token/s and 4M token/s throughput to cycles, WarpLDA takes 240 CPU cycles per iteration per MH step, while LightLDA takes 1,650 CPU cycles. According to Table 2, 240 cycles is only enough for a single memory access, indicating that there are at most one cache miss per token per MH step on average.

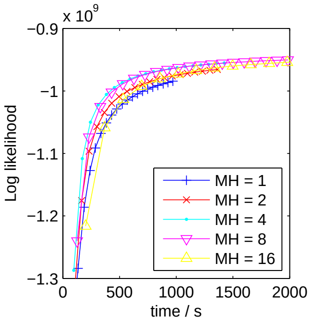
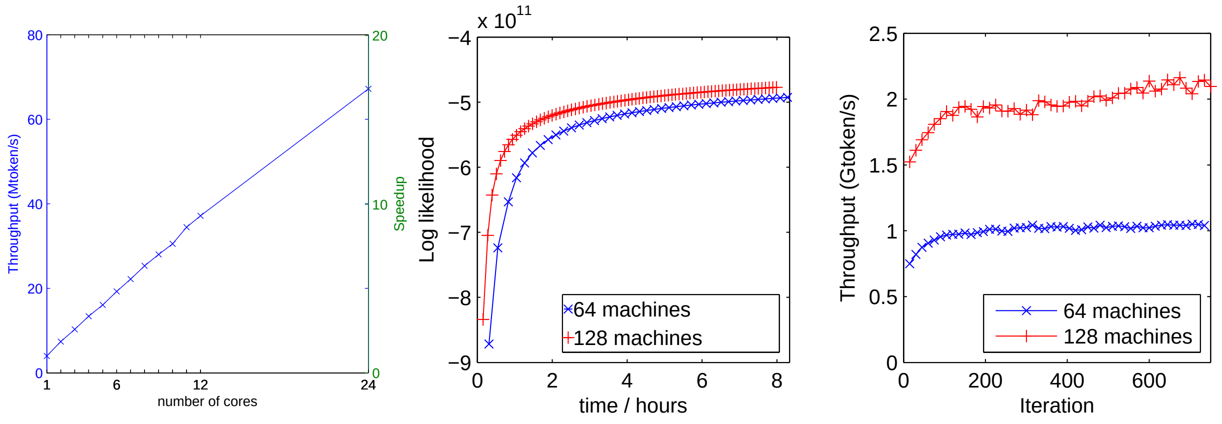


Figure 9: Impact of different $MH$ steps of WarpLDA.

Overall, we observe that without tuning the parameters, WarpLDA converges 6-15x faster than the second best algorithm, without loss of model quality. Fig. 9 shows the impact of $MH$. As $MH$ gets larger the converged log-likelihood is better. This may be because of the bias introduced by the finite-length MH chain. However, we find the topics extracted by algorithms of different $MH$ are all make sense despite the different log-likelihood. Therefore, we stick to small $MH$ such as 2 or 4 to avoid the storage overhead by a large $MH$.



(a) Single machine

(b) 32 machines, convergence

(c) 32 machines, throughput

Figure 11: Scalability results. a) multi-threading speedup on NYTimes, $K = 1,000$, $MH = 4$; b, c) convergence and throughput on ClueWeb12, $K = 10^6$, $MH = 4$.

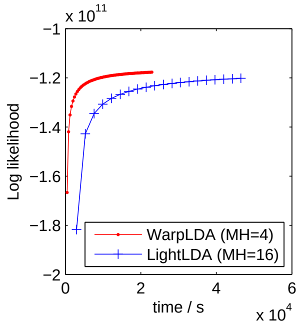


Figure 10: Convergence result on a 40-million-document subset of ClueWeb12, $K = 10000$.

Table 4: Time required to converge to a given log likelihood on small-scale corpus.

| Dataset | WarpLDA (s) | LightLDA (s) | Yahoo!LDA (s) |
|---|---|---|---|
| NIPS | 1.6 | 14 | 28 |
| 20NG | 1 | 7 | 39 |

We also compared WarpLDA ($MH = 4$) with LightLDA ($MH = 16$) in the distributed setting (See Fig. 10). Both algorithms run on 32 machines, and we can see that WarpLDA is 5x faster than LightLDA.

WarpLDA is also fast in the small-scale setting. Specifically, we set the number of cores to 4 to emulate a desktop PC, and compare the time required to converge to a specific log-likelihood, which is produced by Yahoo!LDA after 50 iterations. Both WarpLDA and LightLDA use $MH = 2$. From the results in Table 4, we can see that WarpLDA is 7 - 9 times faster than the second best algorithm (i.e., LightLDA), and can learn the model on both corpora in 2 seconds. This short training time allows users to tune their parameters and get results immediately. Furthermore, it enables real time topic modeling for small-scale data.

## 5.4 Scalability Results

Fig. 11(a) shows the multi-threading speedup result for WarpLDA with $MH = 4$ on NYTimes with $K = 1,000$. The throughput for single thread, single CPU, and 2 CPUs are 4, 37, and 67M token/s, respectively. The speedup of the 24-core version against the single core version is 17x, which is good for such a memory intensive task. The 2-CPU (24 cores) version is faster than the single CPU (12 cores) version by 1.8x, implying that our NUMA strategy is successful.

To demonstrate our capacity of learning large-scale topic models, we learned $K = 10^6$ topics on the 1.5-hundred-million-document ClueWeb12 corpus, on 64 and 128 machines. The hyper-parameter $\beta$ is set to 0.001 for finer grained topics, $MH$ is set to 4, and the number of iterations is set to 1,200. The convergence results are shown in Fig. 11(b). We can see that both runs produce meaningful results after eight hours.[7] Fig. 11(c) shows the throughput is 2G tokens/s with 128 machines, which is never reported in the literature. The speedup of 128 machines over 64 machines is approximately 2x, demonstrating good scalability.

# 6. CONCLUSIONS AND FUTURE WORK

We propose WarpLDA, a simple and efficient $O(1)$ sampler for Latent Dirichlet Allocation. We design WarpLDA to minimize cache misses which are the bottleneck of $O(1)$ LDA samplers. With delayed update and simple proposal distributions, WarpLDA decouples the access to document specific and word specific data, so that the random memory access is restricted within a small scope. We further design a scalable distributed system to handle large-scale datasets and models.

We test WarpLDA on various datasets ranging from thousands of documents to hundreds of millions of documents. Our results show that WarpLDA is consistently 5 - 15x faster than other strong baselines. With WarpLDA, users can learn large topic models with millions of topics and hundreds of million documents in hours, which is both fast and cost effective. WarpLDA can also extract topics from small-scale corpora in seconds, enabling rapid researching and real time topic modeling.

In the future, we plan to exploit the single instruction multiple data (SIMD) nature of WarpLDA to develop $O(1)$ GPU samplers for LDA. Integrating WarpLDA with recent advances in streaming training of LDA is also promising for reducing the memory and network consumption and better scalability.

[7] The learned 1 million topics are available at `http://ml.cs.tsinghua.edu.cn/~jianfei/warplda.html`.

## 7. ACKNOWLEDGMENTS

We would like to thank the authors of LightLDA for the very helpful discussions on the experiments.

## 8. REFERENCES

[1] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola. Scalable inference in latent variable models. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 123–132. ACM, 2012.

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

[3] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh. On smoothing and inference for topic models. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 27–34. AUAI Press, 2009.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[5] J. L. Boyd-Graber, D. M. Blei, and X. Zhu. A topic model for word sense disambiguation. In *EMNLP-CoNLL*, pages 1024–1033, 2007.

[6] T. Broderick, N. Boyd, A. Wibisono, A. C. Wilson, and M. I. Jordan. Streaming variational bayes. In *Advances in Neural Information Processing Systems*, pages 1727–1735, 2013.

[7] J. Canny and H. Zhao. Bidmach: Large-scale learning with zero memory allocation. In *BigLearn Workshop, NIPS*, 2013.

[8] J. Chang and D. Blei. Relational topic models for document networks. In *AISTATS*, 2009.

[9] N. Chen, J. Zhu, F. Xia, and B. Zhang. Discriminative relational topic models. *IEEE Trans. on PAMI*, 37(5):973–986, 2015.

[10] W.-Y. Chen, J.-C. Chu, J. Luan, H. Bai, Y. Wang, and E. Y. Chang. Collaborative filtering for orkut communities: discovery of user latent behavior. In *Proceedings of the 18th international conference on World wide web*, pages 681–690. ACM, 2009.

[11] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[12] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[13] T. Iwata, T. Yamada, and N. Ueda. Probabilistic latent semantic visualization: topic model for visualizing documents. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 363–371. ACM, 2008.

[14] Z. G. Kingsley. Selective studies and the principle of relative frequency in language, 1932.

[15] A. Q. Li, A. Ahmed, S. Ravi, and A. J. Smola. Reducing the sampling complexity of topic models. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 891–900. ACM, 2014.

[16] 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 *Operating Systems Design and Implementation (OSDI)*, 2014.

[17] S. Liu, X. Wang, J. Chen, J. Zhu, and B. Guo. Topicpanorama: a full picture of relevant topics. In *Proceedings of IEEE VAST*, 2014.

[18] E. Meeds, R. Hendriks, S. a. Faraby, M. Bruntink, and M. Welling. Mlitb: Machine learning in the browser. *arXiv preprint arXiv:1412.2432*, 2014.

[19] T. Minka and J. Lafferty. Expectation-propagation for the generative aspect model. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 352–359. Morgan Kaufmann Publishers Inc., 2002.

[20] D. Newman, P. Smyth, M. Welling, and A. U. Asuncion. Distributed inference for latent dirichlet allocation. In *Advances in neural information processing systems*, 2007.

[21] S. Patterson and Y. W. Teh. Stochastic gradient riemannian langevin dynamics on the probability simplex. In *Advances in Neural Information Processing Systems*, pages 3102–3110, 2013.

[22] Y. W. Teh, D. Newman, and M. Welling. A collapsed variational bayesian inference algorithm for latent dirichlet allocation. In *Advances in neural information processing systems*, pages 1353–1360, 2006.

[23] A. J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):253–256, 1977.

[24] H. M. Wallach, D. Minmo, and A. McCallum. Rethinking lda: Why priors matter. 2009.

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

[26] 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. *arXiv preprint arXiv:1405.4402*, 2014.

[27] X. Wei and W. B. Croft. Lda-based document models for ad-hoc retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 178–185. ACM, 2006.

[28] L. Yao, D. Mimno, and A. McCallum. Efficient methods for topic model inference on streaming document collections. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 937–946. ACM, 2009.

[29] J. Yuan, F. Gao, Q. Ho, W. Dai, J. Wei, X. Zheng, E. P. Xing, T.-Y. Liu, and W.-Y. Ma. Lightlda: Big topic models on modest compute clusters. In *Proceedings of the 24th International Conference on World Wide Web*, 2015.

[30] H. Zhao, B. Jiang, and J. Canny. Same but different: Fast and high-quality gibbs parameter estimation. *arXiv preprint arXiv:1409.5402*, 2014.

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