A Scalable Asynchronous Distributed Algorithm for Topic Modeling

Hsiang-Fu Yu
Univ. of Texas, Austin
rofuyu@cs.utexas.edu

Cho-Jui Hsieh
Univ. of Texas, Austin
cjhsieh@cs.utexas.edu

Hyokun Yun
Amazon
yunhyoku@amazon.com

S.V.N Vishwanathan
Univ. of California, Santa Cruz
vishy@ucsc.edu

Inderjit S. Dhillon
Univ. of Texas, Austin
inderjit@cs.utexas.edu

ABSTRACT

Learning meaningful topic models with massive document collections which contain millions of documents and billions of tokens is challenging because of two reasons. First, one needs to deal with a large number of topics (typically on the order of thousands). Second, one needs a scalable and efficient way of distributing the computation across multiple machines. In this paper, we present a novel algorithm F+Nomad LDA which simultaneously tackles both these problems. In order to handle large number of topics we use an appropriately modified Fenwick tree. This data structure allows us to sample from a multinomial distribution over T items in $O(\log T)$ time. Moreover, when topic counts change the data structure can be updated in $O(\log T)$ time. In order to distribute the computation across multiple processors, we present a novel asynchronous framework inspired by the Nomad algorithm of [25]. We show that F+Nomad LDA significantly outperforms recent state-of-the-art topic modeling approaches on massive problems which involve millions of documents, billions of words, and thousands of topics.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—Parameter learning

General Terms

Algorithms, Experimentation

Keywords

Topic Models; Scalability; Sampling

1. INTRODUCTION

Topic models provide a way to aggregate vocabulary from a document corpus to form latent "topics." In particular, Latent Dirichlet Allocation (LDA) [3] is one of the most popular topic modeling approaches. Learning meaningful topic models with massive document collections which contain millions of documents and billions of tokens is challenging because of two reasons. First, one needs to deal with a large number of topics (typically on the order of thousands). Second, one needs a scalable and efficient way of distributing the computation across multiple machines.

Unsurprisingly, there have been significant attempts at developing scalable inference algorithms for LDA. To tackle large number of topics, [23] proposed a clever sparse sampling trick that is widely used in packages like MALLET and Yahoo! LDA. More recently, [11] proposed using the Alias table method to speed up sampling from the multinomial distribution. At the same time, there has also been significant effort towards distributing the computation across multiple processors. Some early efforts in this direction include [22] and [10], where the basic idea is to partition the documents across processors. During each inner iteration the words in the vocabulary are partitioned across processors and each processor only updates the latent variables associated with the subset of documents and words that it owns. After each inner iteration, a synchronization step is used to update global counts and to re-partition the words across processors. In fact, a very similar idea was independently discovered in the context of matrix completion by [6] and [15]. However, in the case of LDA we need to keep a global count synchronized across processors which significantly complicates matters as compared to matrix completion. Arguably, most of the recent efforts towards scalable LDA such as [16, 13] have been focused on the global count issue either implicitly or explicitly. Recently there is also a growing trend in machine learning towards asynchronous algorithms which avoid bulk synchronization after every iteration. For example, in the context of LDA see the work of [1], and in the more general machine learning context see *e.g.*, [7, 12].

In this paper, we propose a new asynchronous distributed topic modeling algorithm called F+Nomad LDA which simultaneously tackles the twin problems of large number of documents and large number of topics. In order to handle large number of topics we use an appropriately modified Fenwick tree. This data structure allows us to sample from a multinomial distribution over T items in $O(\log T)$ time. Moreover, when topic counts change, the data structure can be updated in $O(\log T)$ time. In order to distribute the computation across multiple processors, we present a novel asynchronous framework inspired by the Nomad algorithm

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741682.

of [25]. While we believe that our framework can handle variable update schedules of many different methods, in this paper we will primarily focus on Collapsed Gibbs Sampling (CGS). Our technical contributions can be summarized as follows:

- We identify the following key property of various inference methods for topic modeling: only a single vector of size k needs to be synchronized across multiple processors.
- We present a variant of the Fenwick tree which allows us to efficiently encode a multinomial distribution using $O(T)$ space. Sampling can be performed in $O(\log T)$ time and maintaining the data structure requires only $O(\log T)$ work.
- F+Nomad LDA: we propose a novel parallel framework for various types of inference methods for topic modeling. Our framework utilizes the concept of nomadic tokens to avoid locking and conflict at the same time. Our parallel approach is fully asynchronous with non-blocking communication, thus leading to good speedups. Moreover, our approach minimizes the staleness of the variables (at most k variables can be stale) for distributed parallel computation.
- We demonstrate the scalability of our methods by performing extensive empirical evaluation on large datasets which contain millions of documents and **billions** of words.

2. NOTATION AND BACKGROUND

We begin by very briefly reviewing Latent Dirichlet Allocation (LDA) [3]. Suppose we are given I documents denoted as $d_1, d_2, \ldots, d_I$, and let J denote the number of words in the vocabulary. Moreover, let n_i denote the number of words in a document d_i. Let w_j denote the j-th word in the vocabulary and $w_{i,j}$ denote the j-th word in the i-th document. Assume that the documents are generated by sampling from T topics denoted as $\phi_1, \phi_2, \ldots, \phi_T$; a topic is simply a J dimensional multinomial distribution over words. Each document includes some proportion of the topics. These proportions are latent, and we use the T dimensional probability vector θ_i to denote the topic distribution for a document d_i. Moreover, let $z_{i,j}$ denote the latent topic from which $w_{i,j}$ was drawn. Let α and β be hyper parameters of the Dirichlet distribution. The generative process for LDA can be described as follows:

1. Draw T topics $\phi_k \sim \texttt{Dirichlet}(\beta)$, $k = 1, \ldots, T$.
2. For each document $d_i \in \{d_1, d_2, \ldots, d_I\}$:
   - Draw $\theta_i \sim \texttt{Dirichlet}(\alpha)$.
   - For $j = 1, \ldots, n_i$
     - Draw $z_{i,j} \sim \texttt{Discrete}(\theta_i)$.
     - Draw $w_{i,j} \sim \texttt{Discrete}(\phi_{z_{i,j}})$.

2.1 Inference

The inference task for LDA is to characterize the posterior distribution $Pr(\phi_i, \theta_i, z_{i,j} \mid w_{i,j})$. In the Bayesian setting, we want an efficient way to draw samples from this posterior distribution. Collapsed Gibbs Sampling (CGS) [8] is a popular inference scheme for LDA. Define

$$n_{z,i,w} := \sum_{j=1}^{n_i} I\left(z_{i,j} = z \text{ and } w_{i,j} = w\right), \qquad (1)$$

$n_{z,i,*} = \sum_w n_{z,i,w}$, $n_{z,*,w} = \sum_i n_{z,i,w}$, and $n_{z,*,*} = \sum_{i,w} n_{z,i,w}$, where $I(\cdot)$ is the indicator function. The update rule for CGS can be written as follows

1. Decrease $n_{z_{i,j},i,*}$, $n_{z_{i,j},*,w_{i,j}}$, and $n_{z_{i,j},*,*}$ by 1.
2. Resample $z_{i,j}$ according to

$$\Pr\left(z_{i,j} | w_{i,j}, \alpha, \beta\right) \propto \frac{\left(n_{z_{i,j},i,*} + \alpha_{z_{i,j}}\right)\left(n_{z_{i,j},*,w_{i,j}} + \beta_{w_{i,j}}\right)}{n_{z_{i,j},*,*} + \sum_{j=1}^{J} \beta_j}. \qquad (2)$$

3. Increase $n_{z_{i,j},i,*}$, $n_{z_{i,j},*,w_{i,j}}$, and $n_{z_{i,j},*,*}$ by 1.

Although in this paper we will focus on CGS, note that there are many other inference techniques for LDA such as collapsed variational Bayes, stochastic variational Bayes, or expectation maximization which essentially follow a very similar update pattern [2]. We believe that the parallel framework proposed in this paper will apply to this wider class of inference techniques as well.

2.2 Review of Multinomial Sampling

Given a T-dimensional discrete distribution characterized by unnormalized parameters $\mathbf{p}$ with $p_t \geq 0$ such as in (2), many sampling algorithms can be applied to draw a sample z such that $\Pr(z = t) \propto p_t$.

- LSearch: Linear search on $\mathbf{p}$. **Initialization:** Compute the normalization constant $c_T = \sum_t p_t$. **Generation:** First generate $u = \texttt{uniform}(c_T)$, a uniform random number in $[0, c_T)$, and perform a linear search to find $z = \min\left\{t : \left(\sum_{s \leq t} p_s\right) > u\right\}$.
- BSearch: Binary search on $\mathbf{c} = \texttt{cumsum}(\mathbf{p})$. **Initialization:** Compute $\mathbf{c} = \texttt{cumsum}(\mathbf{p})$ such that $c_t = \sum_{s:s \leq t} p_s$. **Generation:** First generate the cumulated sum $u = \texttt{uniform}(c_T)$ and perform a binary search on $\mathbf{c}$ to find $z = \min\{t : c_t > u\}$.
- Alias method. **Initialization:** Construct an Alias table [19] for $\mathbf{p}$, which contains two arrays of length T: *alias* and *prob*. See [18] for a linear time construction scheme. **Generation:** First generate $u = \texttt{uniform}(T)$, $j = \lfloor u \rfloor$, and

$$z = \begin{cases} j+1 & \text{if } (u - j) \leq \textit{prob}[j+1] \\ \textit{alias}[j+1] & \text{otherwise} \end{cases}.$$

See Table 1 for a comparison of the time/space requirements of each of the above sampling methods.

3. FENWICK TREE SAMPLING

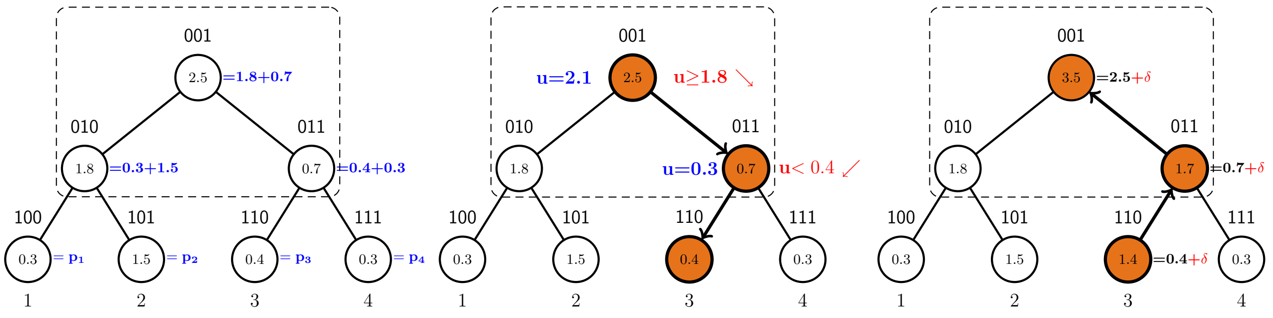
In this section, we first describe a binary tree structure F+tree for fast T-dimensional multinomial sampling. The initialization of an F+tree is linear in T and the cost to generate a sample is logarithmic in T. Furthermore, F+tree can also be maintained in logarithmic time for a single parameter update of p_t. We will explain how such properties of F+tree can be explored to significantly accelerate LDA sampling.

3.1 F+tree Sampling

F+tree, first introduced for weighted sampling without replacement [21], is a simplified and generalized version of Fenwick tree [5], which supports both efficient sampling and update procedures. In fact, Fenwick tree can be regarded as a compressed version of the F+tree studied in this paper.

Table 1: Comparison of samplers for a T-dimensional multinomial distribution $\mathbf{p}$ described by unnormalized parameters $\{p_t : t = 1, \ldots, T\}$.

	Data Structure	Initialization		Generation	Parameter Update
	Space	Time	Space	Time	Time
LSearch	$c_T = \mathbf{p}^\top \mathbf{1}$: $O(1)$	$O(T)$	$O(1)$	$O(T)$	$O(1)$
BSearch	$\mathbf{c} = \texttt{cumsum}(\mathbf{p})$: $O(T)$	$O(T)$	$O(1)$	$O(\log T)$	$O(T)$
Alias Method	$prob, alias$: $O(T)$	$O(T)$	$O(T)$	$O(1)$	$O(T)$
F+tree Sampling	$\texttt{F.initialize}(\mathbf{p})$: $O(T)$	$O(T)$	$O(1)$	$O(\log T)$	$O(\log T)$



(a) F+tree for $\mathbf{p} = [0.3, 1.5, 0.4, 0.3]^\top$ (b) Sampling (c) Updating (with $\delta = 1.0$)

Figure 1: Illustration of sampling and updating using F+tree in logarithmic time.

For simplicity, we assume T is a power of 2. F+tree is a complete binary tree with $2T - 1$ nodes for a given $\mathbf{p}$, where

- each leaf node corresponds to a dimension t and stores p_t as its value, and
- each internal node stores the sum of the values of all of its leaf descendants, or equivalently the sum of values of its two children due to binary tree structure.

See Figure 1a for an example with $\mathbf{p} = [0.3, 1.5, 0.4, 0.3]$ and $T = 4$. Nodes in the dotted rectangle are internal nodes. Similar to the representation used in a heap [4], an array F of length $2T$ can be used to represent the F+tree structure. Let i be the index of each node, and $\texttt{F}[i]$ be the value stored in the i-th node. The index of the left child, right child, and parent of the i-th node is $2i$, $2i + 1$, and $\lfloor i/2 \rfloor$, respectively. The 0/1 string along each node in Figure 1 is the binary number representation of the node index.

Initialization. By the definition of F+tree, given $\mathbf{p}$, the values of F be defined as follows:

$$\texttt{F}[i] = \begin{cases} p_{i-T+1} & \text{if } i \geq T, \\ \texttt{F}[2i] + \texttt{F}[2i+1] & \text{if } i < T. \end{cases} \tag{3}$$

Thus, F can be constructed in $O(T)$ by initializing elements using (3) in reverse. Unlike the Alias method, there is no extra space required in the F+tree initialization in addition to F.

Sample Generation. Sampling using a F+tree can be carried out as a simple top-down traversal procedure to locate $z = \min\left\{t : \left(\sum_{s:s\leq t} p_s\right) > u\right\}$ for a number uniformly sampled between $[0, \sum_t p_t)$. Note that $\sum_t p_t$ is stored in $\texttt{F}[1]$, which can be directly used to generate $u = \texttt{uniform}(\texttt{F}[1])$. Let $\texttt{leaves}(i)$ be the set of all leaf descendant of the i-th node. We can consider a general recursive step in the traversal with the current node i and $u \in [0, \texttt{F}[i])$. The definition of F+tree guarantees that

$$u \geq \texttt{F}[i.\texttt{left}] \Rightarrow z \in \texttt{leaves}(i.\texttt{right}),$$
$$u < \texttt{F}[i.\texttt{left}] \Rightarrow z \in \texttt{leaves}(i.\texttt{left}),$$

This provides a guideline to determine which child to go next. If right child is chosen, $\texttt{F}[i.\texttt{left}]$ should be subtracted from u to ensure $u \in [0, \texttt{F}[i.\texttt{right}])$. Note that as half of the possible t values are removed from the set of candidates, it is clear that this sampling procedure costs only $O(\log T)$ time. The detailed procedure, denoted by $\texttt{F.sample}(u)$, is described in Algorithm 1. A toy example with $u = 2.1$ is illustrated in Figure 1b.

Algorithm 1 Logarithmic time sampling: $\texttt{F.sample}(u)$.

```
Input: F: an F+tree for $\mathbf{p}$, $u = \texttt{uniform}(\texttt{F}[1])$.
Output: $z = \min\left\{t : \left(\sum_{s\leq t} p_s\right) > u\right\}$
• $i \leftarrow 1$
• While $i$ is not a leaf
    – If $u \geq \texttt{F}[i.\texttt{left}]$,
        ∗ $u \leftarrow u - \texttt{F}[i.\texttt{left}]$
        ∗ $i \leftarrow i.\texttt{right}$
    – Else
        ∗ $i \leftarrow i.\texttt{left}$
• $z \leftarrow i - T + 1$
```

Maintenance for Parameter Updates. A simple and efficient maintenance routine to deal with slight changes on the multinomial parameters $\mathbf{p}$ can be very useful in CGS for LDA (See details in Section 3.2). F+tree structure supports a logarithmic time maintenance routine for a single element change on $\mathbf{p}$. Assume the t-th component is updated by δ:

$$\bar{\mathbf{p}} \leftarrow \mathbf{p} + \delta \mathbf{e}_t,$$

Algorithm 2 Logarithmic time F+tree maintenance for a single parameter update: $\texttt{F.update}(t, \delta)$

```
    Input: a F+tree F for $\mathbf{p}$, $t$, $\delta$.
    Output: F+tree F is updated for $\bar{\mathbf{p}} \equiv \mathbf{p} + \delta \mathbf{e}_t$
• $i \leftarrow \texttt{leaf}[t]$
• While $i$ is a valid node
    – $\texttt{F}[i] = \texttt{F}[i] + \delta$
    – $i \leftarrow i.\texttt{parent}$
```

where $\mathbf{e}_t$ is the t-th column of the identity matrix of order T. A simple bottom-up update procedure to modify a F+tree F for the current $\mathbf{p}$ to a F+tree for $\bar{\mathbf{p}}$ can be carried out as follows. Let $\texttt{leaf}[t]$ be the leaf node corresponding to t. For all the ancestors i of $\texttt{leaf}[t]$ (self included), perform the following delta update:

$$\texttt{F}[i] = \texttt{F}[i] + \delta.$$

See Figure 1c for an illustration with $t = 3$ and $\delta = 1.0$. The detailed procedure, denoted by $\texttt{F.update}(t, \delta)$, is described in Algorithm 2. The maintenance cost is linear to the depth of the F+tree, which is $O(\log T)$. Note that to deal with a similar change in $\mathbf{p}$, LSearch can update its normalization constant $c_T \leftarrow c_T + \delta$ in a constant time, while both BSearch and Alias method require to re-construct the entire data structure (either $\mathbf{c} = \texttt{cumsum}(\mathbf{p})$ or the Alias table: *alias* and *prob*), which costs $O(T)$ time in general.

See Table 1 for a summary of the complexity analysis for each multinomial sampling approach. Clearly, LSearch has the smallest update cost but the largest generation cost, and Alias method has the best generation cost but the worst maintenance cost. In contrast, F+tree sampling has a logarithmic time procedure for both operations.

Algorithm 3 F+LDA with word-by-word sampling

```
• $\texttt{F.initialize}(\mathbf{q})$, with $q_t = \frac{\beta}{n_t + \bar{\beta}}$
• For each word $w$
    – $\texttt{F.update}(t, n_{tw}/(n_t + \bar{\beta})) \quad \forall t \in T_w$
    – For each occurrence of $w$, say $w_{i,j} = w$ in $d_i$
        * $t \leftarrow z_{i,j}$
        * Decrease $n_t$, $n_{td_i}$, $n_{tw}$ by one
        * $\texttt{F.update}(t, \delta)$ with $\delta = \frac{n_{tw} + \beta}{n_t + \bar{\beta}} - \texttt{F}[\texttt{leaf}(t)]$
        * $\mathbf{c} \leftarrow \texttt{cumsum}(\mathbf{r})$ (on $T_w$ only)
        * $t \leftarrow \texttt{discrete}(\mathbf{p}, \texttt{uniform}(\alpha \texttt{F}[1] + \mathbf{r}^\top \mathbf{1}))$ by
          (6)
        * Increase $n_t$, $n_{td_i}$, $n_{tw}$ by one
        * $\texttt{F.update}(t, \delta)$ with $\delta = \frac{n_{tw} + \beta}{n_t + \bar{\beta}} - \texttt{F}[\texttt{leaf}(t)]$
        * $z_{i,j} \leftarrow t$
    – $\texttt{F.update}(t, -n_{tw}/(n_t + \bar{\beta})) \quad \forall t \in T_w$
```

3.2 F+LDA = LDA with F+tree Sampling

In this section, we give details on applying F+tree sampling to CGS for LDA. Let us focus on a single CGS step in LDA with the current document id d_i, the current word w, and the current topic assignment t_{cur}. For simplicity of presentation, we further denote $n_{td} = n_{t,d_i,*}$, $n_{tw} = n_{t,*,w}$, and $n_t = n_{t,*,*}$ and assume $\alpha_t = \alpha, \forall t$, $\beta_j = \beta, \forall j$, and $\bar{\beta} = J \times \beta$. The multinomial parameter $\mathbf{p}$ of the CGS step in (2) can be decomposed into two terms as follows.

$$\begin{aligned} p_t &= \frac{(n_{td} + \alpha)(n_{tw} + \beta)}{n_t + \bar{\beta}}, \quad \forall t = 1, \ldots, T. \\ &= \beta \left(\frac{n_{td} + \alpha}{n_t + \bar{\beta}} \right) + n_{tw} \left(\frac{n_{td} + \alpha}{n_t + \bar{\beta}} \right). \end{aligned} \tag{4}$$

Let $\mathbf{q}$ and $\mathbf{r}$ be two vectors with $q_t = \frac{n_{td} + \alpha}{n_t + \bar{\beta}}$ and $r_t = n_{tw} q_t$. Some facts and implications about this decomposition:

(a) $\mathbf{p} = \beta \mathbf{q} + \mathbf{r}$. This leads to a simple two-level sampling for $\mathbf{p}$

$$\texttt{discrete}(\mathbf{p}, u) = \begin{cases} \texttt{discrete}(\mathbf{r}, u) & \text{if } u \leq \mathbf{r}^\top \mathbf{1}, \\ \texttt{discrete}(\mathbf{q}, \frac{u - \mathbf{r}^\top \mathbf{1}}{\beta}) & \text{otherwise}, \end{cases}$$

where $\mathbf{1}$ is the all-ones vector and $\mathbf{p}^\top \mathbf{1}$ denotes the normalization constant for $\mathbf{p}$, and $u = \texttt{uniform}(\mathbf{p}^\top \mathbf{1})$. This means that sampling for $\mathbf{p}$ can be very fast if $\mathbf{q}$ and $\mathbf{r}$ can be sampled efficiently.

(b) $\mathbf{q}$ is always dense but only two elements will be changed at each CGS step if we follow a document-by-document sampling sequence. Note $\mathbf{q}$ only depends on n_{td}. Decrement or increment of a single n_{td} only changes a single element of $\mathbf{q}$. We propose to apply F+tree sampling for $\mathbf{q}$ for its logarithmic time sampling and maintenance. At the beginning of CGS for LDA, a F+tree F for $\mathbf{q}$ with $q_t = \frac{\alpha}{n_t + \bar{\beta}}$ is constructed in $O(T)$ time. When CGS switches to a new document d_i, perform the following updates

$$\texttt{F.update}(t, \frac{n_{td}}{n_t + \bar{\beta}}) \quad \forall t \in T_d := \{t : n_{td} \neq 0\}.$$

When CGS finishes sampling for this document, we can perform $\texttt{F.update}(t, \frac{-n_{td}}{n_t + \bar{\beta}})$ $\forall t \in T_d$. Both updates can be done in $O(|T_d| \log T)$. As $|T_d|$ is upper bounded by the number of words in this document, the amortized sampling cost for each word in the document remains $O(\log T)$.

(c) $\mathbf{r}$ is T_w sparse, where $T_w := \{t : n_{tw} \neq 0\}$. Unlike $\mathbf{q}$, all the elements of $\mathbf{r}$ change when we switch from one word to another word in the same document. Moreover, $\mathbf{r}$ is only used once to compute $\mathbf{r}^\top \mathbf{1}$ and to generate at most one sample. Thus, we propose to use BSearch approach to perform the sampling for $\mathbf{r}$. In particular, we only calculate the cumulative sum on nonzero elements in T_w. Thus, the initialization cost of BSearch is $O(|T_w|)$ and the sampling cost is $O(\log |T_w|)$.

Word-by-word CGS for LDA. Other than the traditional document-by-document CGS for LDA, we can also consider CGS using a word-by-word sampling sequence. For this sequence, we consider another decomposition of (4) as follows.

$$p_t = \alpha \left(\frac{n_{tw} + \beta}{n_t + \bar{\beta}} \right) + n_{td} \left(\frac{n_{tw} + \beta}{n_t + \bar{\beta}} \right), \quad \forall t. \tag{5}$$

For this decomposition (5), $\mathbf{q}$ and $\mathbf{r}$ have analogous definitions such that $q_t = \frac{n_{tw} + \beta}{n_t + \bar{\beta}}$ and $r_t = n_{td} q_t$, respectively. The corresponding three facts for (5) are as follows.

(a) $\mathbf{p} = \alpha \mathbf{q} + \mathbf{r}$. The two-level sampling for $\mathbf{p}$ is

$$\texttt{discrete}(\mathbf{p}, u) = \begin{cases} \texttt{discrete}(\mathbf{r}, u) & \text{if } u \leq \mathbf{r}^\top \mathbf{1}, \\ \texttt{discrete}(\mathbf{q}, \frac{u - \mathbf{r}^\top \mathbf{1}}{\alpha}) & \text{otherwise}, \end{cases} \tag{6}$$

Table 2: Comparison of various sampling methods for LDA. We use $\#MH$ to denote number of Metropolis-Hasting steps for Alias LDA. Note that in this table only the time complexity is presented—there are some hidden constants that also play important roles in practice. For example, the initialization cost of the Alias table is much more than linear search although they have the same time complexity.

	F+LDA		F+LDA		Sparse-LDA			Alias-LDA	
Sample Sequence	Word-by-Word		Doc-by-Doc		Doc-by-Doc			Doc-by-Doc	
Exact Sampling	Yes		Yes		Yes			No	
Decomposition	$\alpha\left(\frac{n_{tw}+\beta}{n_t+\bar{\beta}}\right)+$	$n_{td}\left(\frac{n_{tw}+\beta}{n_t+\bar{\beta}}\right)$	$\beta\left(\frac{n_{td}+\alpha}{n_t+\bar{\beta}}\right)+$	$n_{tw}\left(\frac{n_{td}+\alpha}{n_t+\bar{\beta}}\right)$	$\frac{\alpha\beta}{n_t+\bar{\beta}}$	$+\beta\left(\frac{n_{td}}{n_t+\bar{\beta}}\right)+$	$n_{tw}\left(\frac{n_{td}+\alpha}{n_t+\bar{\beta}}\right)$	$\alpha\left(\frac{n_{tw}+\beta}{n_t+\bar{\beta}}\right)+$	$n_{td}\left(\frac{n_{tw}+\beta}{n_t+\bar{\beta}}\right)$
Sampling method	F+tree	BSearch	F+tree	BSearch	LSearch	LSearch	LSearch	Alias	Alias
Fresh samples	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Initialization	$O(\log T)$	$O(\|T_d\|)$	$O(\log T)$	$O(\|T_w\|)$	$O(1)$	$O(1)$	$O(\|T_w\|)$	$O(1)$	$O(\|T_d\|)$
Sampling	$O(\log T)$	$O(\log \|T_d\|)$	$O(\log T)$	$O(\log \|T_w\|)$	$O(T)$	$O(\|T_d\|)$	$O(\|T_w\|)$	$O(\#MH)$	$O(\#MH)$

(b) $\mathbf{q}$ is always dense but only very few elements will be changed at each CGS step using word-by-word sampling sequence. A F+tree structure $\mathtt{F}$ is maintained for $\mathbf{q}$. The amortized update time for each occurrence of a word is $O(\log T)$ and the sampling generation for $\mathbf{q}$ using $\mathtt{F}$ also costs $O(\log T)$. Thus, $\mathtt{discrete}(\mathbf{q}, u) := \mathtt{F.sample}(u)$.

(c) $\mathbf{r}$ is a sparse vector with $|T_d|$ non-zeros. BSearch is used to construct $\mathbf{c} = \mathtt{cumsum}(\mathbf{r})$ in $O(T_d)$ space and time. $\mathbf{c}$ is used to perform binary search to generate a sample required by CGS for the occurrence of the current word. Thus, $\mathtt{discrete}(\mathbf{r}, u) := \mathtt{binary\_search}(\mathbf{c}, u)$.

The detailed procedure of using word-by-word sampling sequence is described in Algorithm 3. Let us analyse the performance difference of F+LDA between two sampling sequences of a large number of documents. The amortized cost for each CGS step is $O(|T_d| + \log T)$ for the word-by-word sequence and $O(|T_w|+\log T)$ for the document-by-document sequence. Note that $|T_d|$ is always bounded by the number of words in a document, which is usually a much smaller number than a large T (say 1024). In contrast, $|T_w|$ approaches to T when the number of documents increases. Thus, we can expect that F+LDA with the word-by-word sequence is faster than the document-by-document sequence. Empirical results in Section 5.1 also confirm our analysis.

3.3 Related Work

SparseLDA [23] is the first sampling method which considered decomposing $\mathbf{p}$ into a sum of sparse vectors and a dense vector. In particular, it considers a three-term decomposition of p_t as follows.

$$p_t = \frac{\alpha\beta}{n_t + \bar{\beta}} + \beta\left(\frac{n_{td}}{n_t + \bar{\beta}}\right) + n_{tw}\left(\frac{n_{td} + \alpha}{n_t + \bar{\beta}}\right),$$

where the first term is dense, the second term is sparse with $|T_d|$ non-zeros, and the third term is sparse with $|T_w|$ non-zeros. In both SparseLDA implementations (Yahoo! LDA [16] and Mallet LDA [23]), LSearch is applied to all of these three terms. As SparseLDA follows the document-by-document sequence, very few elements will be changed for the first two terms at each CGS step. Sampling procedures for the first two terms have very low chance to be performed due to the observation that most mass of p_t is contributed from the third term. The choice of LSearch, whose normalization constant c_T can be updated in $O(1)$ time, for the first two terms is reasonable. Note that $O(T)$ and $O(|T_d|)$ initialization costs for the first two terms can be amortized. The overall amortized cost for each CGS step is $O(|T_w| + |T_d| + |T|)$.

AliasLDA [11] is a recently proposed approach which reduces the amortized cost of each step to $O(|T_d|)$. AliasLDA considers the following decomposition of $\mathbf{p}$:

$$p_t = \alpha\left(\frac{n_{tw} + \beta}{n_t + \bar{\beta}}\right) + n_{td}\left(\frac{n_{tw} + \beta}{n_t + \bar{\beta}}\right).$$

Instead of the "exact" multinomial sampling for $\mathbf{p}$, AliasLDA considers a proposal distribution $\mathbf{q}$ with a very efficient generation routine and performs a series of Metropolis-Hasting (MH) steps using this proposal to simulate the true distribution $\mathbf{p}$. In particular, the proposal distribution is constructed using the latest second term and a stale version of the first term. For both terms, Alias method is applied to perform the sampling. $\#MH$ steps decides the quality of the sampling results. The overall amortized cost for each CGS step is $O(|T_d| + \#MH)$. Note the initialization cost $O(|T|)$ for the first term can be amortized as long as the same Alias table can be used to generate T samples.

See Table 2 for a detailed summary for LDA using various sampling methods. Note that the hidden coefficient ρ_A in the $O(|T_d|)$ notation for the construction of the Alias table is larger than the coefficient ρ_B for the construction of BSearch and the coefficient ρ_F for the maintenance and sampling of F+tree. Thus as long as $T < 2^{\frac{\rho_A - \rho_B}{\rho_F}|T_d|}$, F+LDA using the word-by-word sampling sequence is faster than AliasLDA. Empirical results in Section 5.1 also show the superiority of F+LDA over AliasLDA for real-world datasets even using $T = 50,000$.

4. PROPOSED PARALLEL APPROACH

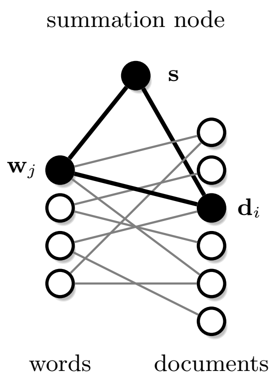
In this section we present our second innovation—a novel parallel framework for CGS. Note that the same technique can also be used for other inference techniques for LDA such as collapsed variational Bayes and stochastic variational Bayes [2] since they follow similar update patterns.

To explain our proposed approach, we find it instructive to consider a hypergraph G. Let $G = (V, E)$ be a hypergraph with $(I + J + 1)$ nodes:

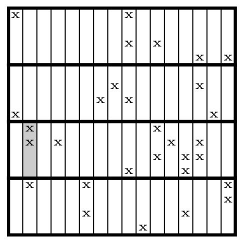
$$V = \{\mathbf{d}_i : i = 1, \ldots, I\} \cup \{\mathbf{w}_j : j = 1, \ldots, J\} \cup \{\mathbf{s}\},$$

and hyperedges:

$$E = \{e_{ij} = \{\mathbf{d}_i, \mathbf{w}_j, \mathbf{s}\}\},$$



(a) Access graph for LDA



(b) Task and Data Partition

Figure 2: Abstract access graph for LDA

where $|E| = \sum_i n_i$. Note that G contains multi-edges, which means that the same hyperedge can appear more than once in E just as a single word can appear multiple times in a document. Clearly, G is equivalent to a bag-of-words representation of the corpus $\{d_1, \ldots, d_I\}$; each $\mathbf{d}_i$ is associated with the i-th document, each $\mathbf{w}_j$ is associated with the j-th vocabulary, and each hyperedge e_{ij} corresponds to one occurrence of the vocabulary w_j in the i-th document d_i. See Figure 2 (a) for a visual illustration; here, each gray edge corresponds to an occurrence of a word and the black triangle highlights a particular hyperedge $e_{ij} = \{\mathbf{d}_i, \mathbf{w}_j, \mathbf{s}\}$.

To further connect G to the update rule of CGS, we associate each node of G with a T-dimensional vector. In many inference methods, an update based on a single occurrence w_{ij} can be realized as a graph operation on G which accesses values of nodes in a single hyperedge e_{ij}. More concretely, let us define the t-th coordinate of each vector as follows:

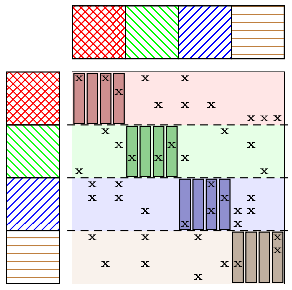
$$(\mathbf{d}_i)_t := n_{t,i,*}, \quad (\mathbf{w}_j)_t := n_{t,*,w_j}, \text{ and } \quad (\mathbf{s})_t := n_{t,*,*}.$$

Based on the update rule of CGS, we can see that the update for the occurrence of w_{ij} only reads from and writes to the values stored in $\mathbf{d}_i$, $\mathbf{w}_{w_{ij}}$, and $\mathbf{s}$.

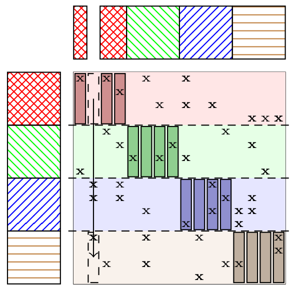
Interestingly, this property of the updates is reminiscent of that of the stochastic gradient descent (SGD) algorithm for matrix completion model. Similarly to LDA, matrix completion model has two sets of parameters $\mathbf{w}_1, \ldots, \mathbf{w}_J$ and $\mathbf{d}_1, \ldots, \mathbf{d}_I$, and each SGD update requires only one of $\mathbf{w}_j$ and one of $\mathbf{d}_i$ to be read and modified. Since each update is highly localized, there is considerable parallelism available; [25] exploits this property to propose an efficient asynchronous parallel SGD algorithm for matrix completion.

The crucial difference in the case of LDA, however, is that there is an additional variable $\mathbf{s}$ which participates in every hyperedge of the graph. Thus, if we change the update sequence from $(e_{ij}, e_{i'j'})$ to $(e_{i'j'}, e_{ij})$, then even if $i \neq i'$ and $j \neq j'$ the result of updates will not be the same since the value of $\mathbf{s}$ changes in the first update. Fortunately, this dependency is very weak; each element of $\mathbf{s}$ is a large number because it is a summation over the whole corpus and each update changes its value at most by one, therefore the relative change of $\mathbf{s}$ made in a short period of time is often negligible.

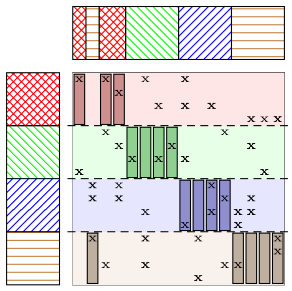
While existing approaches such as Yahoo! LDA [16] exploit this observation by introducing a parameter server and let each machine query the server to retrieve recent updates, it is certainly not desirable in a large scale system that every machine has to query the same central server. Motivated by the "nomadic" algorithm introduced by [25] for matrix completion, we propose a new parallel framework for LDA that is decentralized, asynchronous and lock-free.



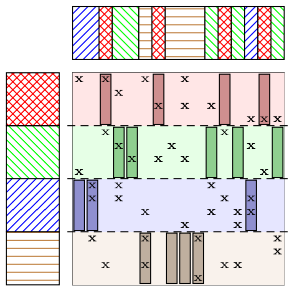
(a) Initial assignment of $\mathbf{w}_j$. Each worker works only on the diagonal active area in the beginning.



(b) After a worker finishes processing j, it sends the corresponding $\mathbf{w}_j$ to another worker. Here, $\mathbf{w}_2$ is sent from worker 1 to 4.



(c) Upon receipt, the $\mathbf{w}_j$ is processed by the new worker. Here, worker 4 can now process $\mathbf{w}_2$ since it owns it.



(d) During the execution of the algorithm, the ownership of the $\mathbf{w}_j$ changes.

Figure 3: Illustration of the Nomad LDA algorithm

Algorithm 4 The basic Nomad LDA algorithm

```
Given: initialized $\mathbf{s}_l$, $\bar{\mathbf{s}}$, and local queue $\mathbf{q}_l$
• While stop signal has not been received
    – If receive a token $\tau$, $push(\mathbf{q}_l, \tau)$
    – $\tau \leftarrow pop(\mathbf{q}_l)$
    – If $\tau = \tau_s$
        ∗ $\mathbf{s} \leftarrow \mathbf{s} + (\mathbf{s}_l - \bar{\mathbf{s}})$
        ∗ $\mathbf{s}_l \leftarrow \mathbf{s}$
        ∗ $\bar{\mathbf{s}} \leftarrow \mathbf{s}$
        ∗ Send $\tau_s$ to another worker
    – Else if $\tau = \tau_j := (j, \mathbf{w}_k)$
        ∗ Perform the $j$-th subtask
        ∗ Send $\tau_s$ to another worker
```

4.1 Nomadic Framework for Parallel LDA

Let p be the number of parallel workers, which can be a thread in a shared-memory multi-core machine or a processor in a distributed memory multi-machine system.

Data Partition and Subtask Split. The given document corpus is split into p portions such that the l-th worker owns the l-th partition of the data, $D_l \subset \{1, \ldots, J\}$. Unlike the other parallel approach where each unit subtask is a document owned by the worker, our approach uses a fine-

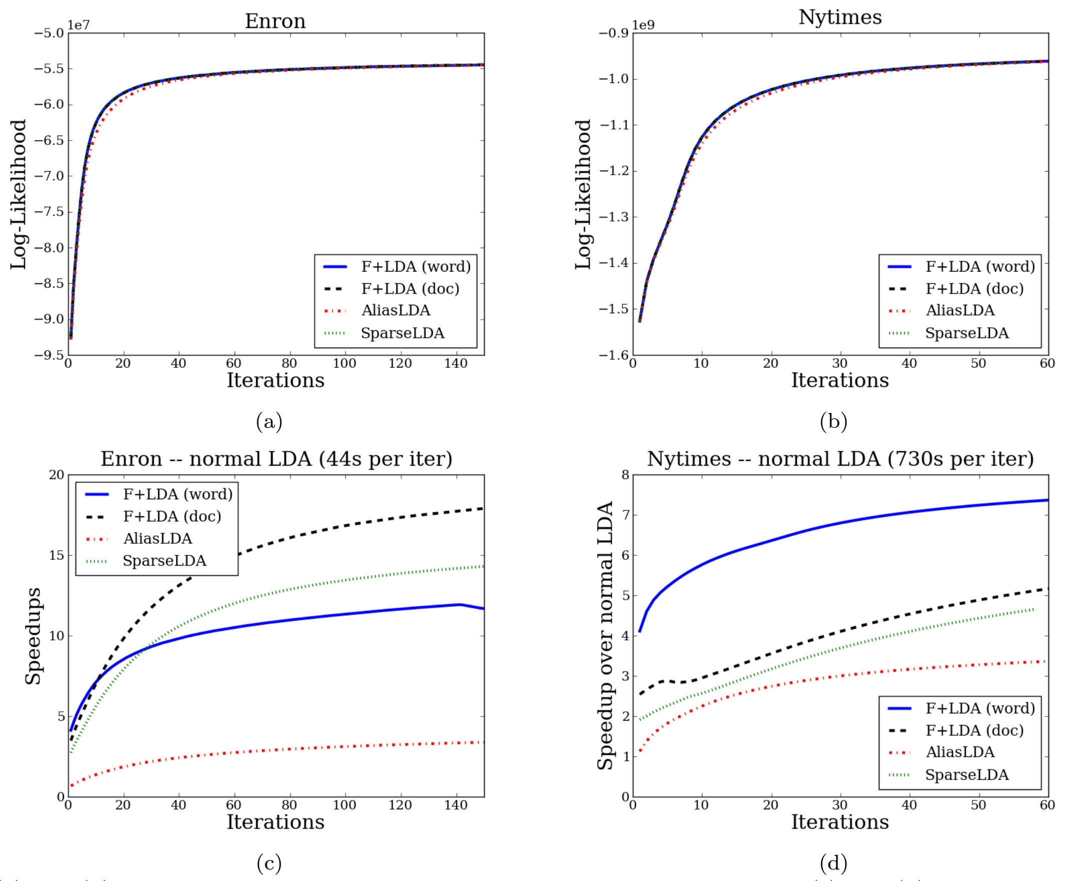


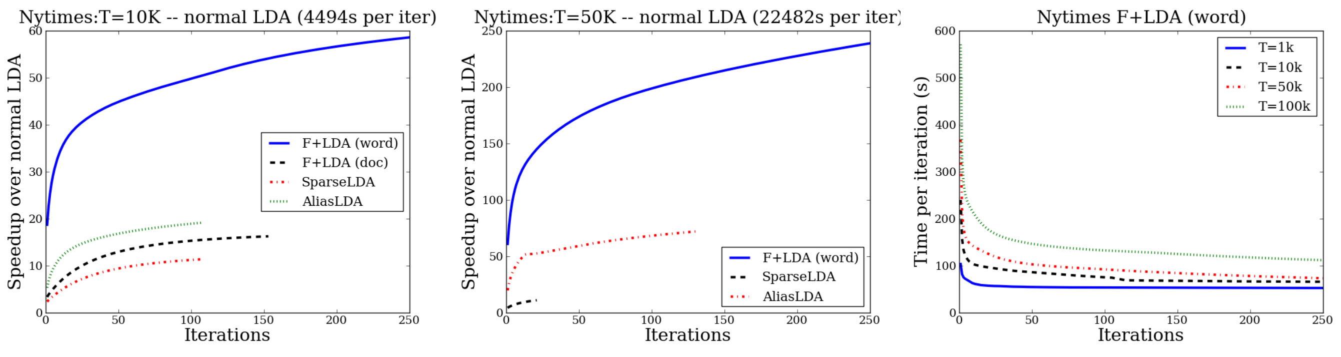
Figure 4: (a) and (b) present the convergence speed in terms of number of iterations. (c) and (d) present the sampling speed of each iteration—the y-axis is the speedup over the normal LDA implementation which takes $O(T)$ time to generate one sample. We observe all the sampling algorithms have similar convergence speed, while F+LDA(doc) is the fastest compared to other document-wise sampling approaches. Also, F+LDA(word) is faster than F+LDA(doc) for larger datasets, which confirms our analysis in Section 3.2.

grained split for tasks. Note that in the inference for LDA, each word occurrence corresponds to an update. Thus, we consider a unit subtask $\mathbf{t}_j$ as all occurrences of word w_j in all documents owned by the worker. See Figure 2b for an illustration in the data partition and task split. Each "x" denotes an occurrence of a word. Each block row (bigger rectangle) represents a data partition owned by a worker, while each smaller rectangle stands for a unit subtask for the worker.

Asynchronous Computation. It is known that synchronous computation would suffer from *the curse of last reducer* when the load-balance is poor. In this work, we aim to develop an asynchronous parallel framework where each worker maintains a local job queue $\mathbf{q}_l$ such that the worker can keep performing the subtask popped from the queue without worrying about data conflict and synchronization. To achieve this goal, we first study the characteristics of subtasks. The subtask $\mathbf{t}_j$ for the l-th worker involves updates on all occurrences of $\mathbf{w}_j$ in D_l, which means that to perform $\mathbf{t}_j$, the l-th worker must acquire permission to access $\{\mathbf{d}_i : i \in D_l\}$, $\mathbf{w}_j$, and $\mathbf{s}$. Our data partition scheme has guaranteed that two workers will never need to access a same $\mathbf{d}_i$ simultaneously. Thus we can always keep the ownership of $\mathbf{d}_i, \forall i \in D_l$ to l-th worker. The difficulty for parallel execution comes from the access to $\mathbf{w}_j$ and $\mathbf{s}$ which can be accessed by different workers at the same time. To overcome this difficulty, we propose to use a *nomadic token passing* scheme to avoid access conflicts. Token passing is a standard technique used in telecommunications to avoid conflicting access to a resource shared by many members. The idea is "owner computes": only the member with the ownership of the token has the permission to access the resource. Here we borrow the same idea to avoid the situation where two workers require access to the same $\mathbf{w}_j$ and $\mathbf{s}$.

Nomadic Tokens for $\mathbf{w}_j$. We have a token τ_j dedicated for the ownership of each word $\mathbf{w}_j$. These J tokens are nomadically passed among p workers. The ownership of a token τ_j means the worker can perform the subtask $\mathbf{t}_j$. Each token τ_j is a tuple $(j, \mathbf{w}_j)$, where the first entry is the index for the token, and the second entry is the latest value of $\mathbf{w}_j$. For a worker, a token τ means the activation of the corresponding inference subtask. Thus, we can guarantee that 1) the values of $\mathbf{w}_j$ used in each subtask is always up-to-date; 2) no two workers require access to a same $\mathbf{w}_j$.

Nomadic Token for $\mathbf{s}$. So far we have successfully kept the values of $\mathbf{d}_i$ and $\mathbf{w}_j$ used in each subtask latest, and avoid access conflicts by nomadic token passing. However, all subtasks depend on each other due to the need to access $\mathbf{s}$. Based on the summation property, we propose to deal with this issue by creating a special nomadic token $\tau_s = (0, \mathbf{s})$ for $\mathbf{s}$, where 0 is the token index for τ_s, and have two copies of $\mathbf{s}$ in each worker: $\mathbf{s}_l$ and $\bar{\mathbf{s}}$. $\mathbf{s}_l$ is a local shadow node for $\mathbf{s}$. The



(a) (b) (c)

Figure 5: (a) and (b) present the sampling speed for $T = 10,000$ and $T = 50,000$. The superiority of F+LDA(word) is more significant when T is large. (c) shows the sampling speed for various T. The increase in sampling time is much smaller than the increase in the number of topics T. Note that the sampling time per iteration for $T = 100,000$ is only about twice as much as the time required for $T = 1,000$.

l-th worker always uses the values of $\mathbf{s}_l$ to perform updates and makes the modification to $\mathbf{s}_l$. $\bar{\mathbf{s}}$ was the snapshot of $\mathbf{s}$ from the last arrival of τ_s. Due to the additivity of $\mathbf{s}$, the delta $\mathbf{s}_l - \mathbf{s}$ can be regarded as the effort that has been made since the last arrival of τ_s. Thus, each time that τ_s arrives, the worker can perform the following operations to accumulate its local effort to the global $\mathbf{s}$ and update its local $\mathbf{s}_l$.

1. $\mathbf{s} \leftarrow \mathbf{s} + (\mathbf{s}_l - \bar{\mathbf{s}})$
2. $\bar{\mathbf{s}} \leftarrow \mathbf{s}$
3. $\mathbf{s}_l \leftarrow \mathbf{s}$

We present the general idea of Nomad LDA in Algorithm 4 and its illustration in Figure 3.

4.2 Related Work

Unlike the situation in the serial case, the latest values of $n_{z,*,w}$ and $n_{z,*,*}$ can be distributed among different machines in the distributed setting. The existing parallel approaches focus on developing mechanisms to communicate these values. We briefly review two approaches for parallelizing CGS in distributed setting: AdLDA [13] and Yahoo! LDA [16]. In both approaches, each machine has a local copy of the *entire* $n_{z,*,w}$ and $n_{z,*,*}$. AdLDA uses bulk synchronization to update its local copy after each iteration. At each iteration, each machine just uses the snapshot from last synchronization point to conduct Gibbs sampling. On the other hand, Yahoo! LDA creates a central parameter server to maintain the latest values for $n_{z,*,w}$ and $n_{z,*,*}$. Every machine asynchronously communicates with this machine to send the local update to the server and get new values to update its local copy. Note that the communication is done asynchronously in Yahoo! LDA to avoid expensive network locking. The central idea of Yahoo! LDA is that modest stale values would not affect the sampler significantly. Thus, there is no need to spend too much effort to synchronize these values. Note that for these two approaches, both values of $n_{z,*,w}$ and $n_{z,*,*}$ used in Gibbs sampling could be stale. In contrast, our proposed Nomad LDA has the following advantages:

- No copy of the entire $n_{z,*,w}$ is required in each machine.
- The value of $n_{z,*,w}$ used in the Gibbs sampling is always up-to-date in each machine.
- The computation is both asynchronous and decentralized.

Our Nomad LDA is close to a parallel approach for matrix completion [25], which also utilized the concept of nomadic variables. However, the application is completely different. [25] concentrates on parallelizing stochastic gradient descent for matrix completion. The access graph for this problem is a bipartite graph, and there is no variable that needs to be synchronized across processors.

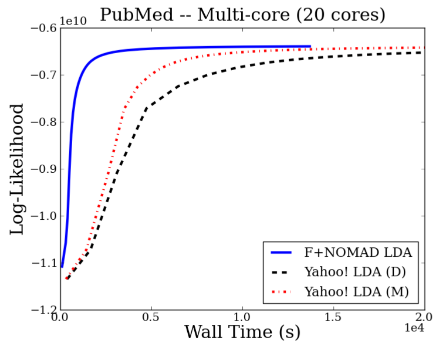
5. EXPERIMENTAL EVALUATION

In this section we investigate the performance and scaling of our proposed algorithms. We demonstrate that our proposed F+tree sampling method is very efficient in handling large number of topics compared to the other approaches in Section 5.1. When the number of documents is also large, in Section 5.2 we show our parallel framework is very efficient in multi-core and distributed systems.

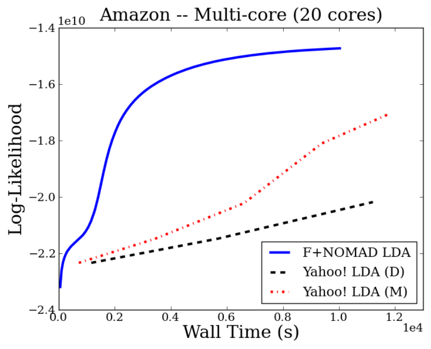
Datasets. We work with five real-world large datasets—Enron, NyTimes, PubMed, Amazon, and UMBC. The detailed data set statistics are listed in Table 3. Among them, Enron, NyTimes and PubMed are bag-of-word datasets in the UCI repository[1]. These three datasets have been used to demonstrate the scaling behavior of topic modeling algorithms in many recent papers [2, 16, 11]. In fact, the PubMed dataset stretches the capabilities of many implementations. For instance, we tried to use LDA code from `http://www.ics.uci.edu/~asuncion/software/fast.htm`, but it could not handle PubMed.

To demonstrate the scalability of our algorithm, we use two more large-scale datasets—Amazon and UMBC. The Amazon dataset consists of approximately 35 million product reviews from Amazon.com, and was downloaded from the Stanford Network Analysis Project (SNAP) home page. Since reviews are typically short, we split the text into words, removed stop words, and using Porter stemming [14]. After this pre-processing we discarded words that appear fewer than 5 times or in 5 reviews. Finally, any reviews that were left with no words after this pre-processing were discarded.

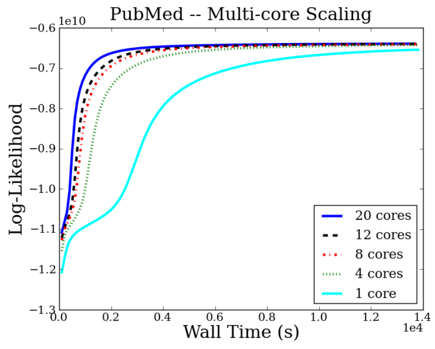
[1]https://archive.ics.uci.edu/ml/datasets/Bag+of+Words



(a)



(b)



(c)

Figure 6: (a) and (b) show the comparison between Nomad LDA and Yahoo! LDA using 20 cores on a single machine. (c) shows the scaling performance of Nomad LDA as a function of number of cores.

Table 3: Data statistics.

	# documents (I)	# vocabulary (J)	# words
Enron	37,861	28,102	6,238,796
NyTimes	298,000	102,660	98,793,316
PubMed	8,200,000	141,043	737,869,083
Amazon	29,907,995	1,682,527	1,499,602,431
UMBC	40,599,164	2,881,476	1,483,145,192

This resulted in a corpus of approximately 30 million documents and approximately **1.5 billion** words.

The UMBC WebBase corpus is downloaded from `http://ebiquity.umbc.edu/blogger/2013/05/01/`. It contains a collection of pre-processed paragraphs from the Stanford WebBase[2] crawl on February 2007. The original dataset has approximately 40 million paragraphs and 3 billion words. We further processed the data by stemming and removing stop words following the same procedure in LibShortText [24]. This resulted in a corpus of approximately **1.5 billion** words.

Hardware. The experiments are conducted on a parallel platform at the Texas Advanced Computing Center (TACC), called Maverick[3]. Each node contains 20 Intel Xeon E5-2680 CPUs and 256 GB memory. Each job can run on at most 32 nodes (640 cores) for at most 12 hours.

Parameter Setting. Throughout the experiments we set the hyper parameters $\alpha = 50/T$ and $\beta = 0.01$, where T is the number of topics. Previous papers showed that this parameter setting gives good model qualities [9], and many widely-used software such as Yahoo! LDA and Mallet-LDA also use this as the default parameters. To test the performance with a large number of topics, we set $T = 1024$ in all the experiments except the ones in Figure 5.

Evaluation. Our main competitor is Yahoo! LDA in large-scale distributed setting. To have a fair comparison, we use the same training likelihood routine to evaluate the quality of model (see Eq. (2) in [16] for details).

5.1 Comparison of sampling methods: handling large number of topics

In this section, we compare various sampling strategies used for LDA in the serial setting. We include the following sampling strategies into the comparison (see Section 3 for details):

1. F+LDA: our proposed sampling scheme. Document-wise and word-wise sampling order are denoted by F+LDA(doc) and F+LDA(word), respectively.
2. SparseLDA: the approach that uses linear search on PDF to conduct document-wise sampling. This approach is used in Yahoo! LDA and Mallet-LDA.
3. AliasLDA: the approach that uses Alias method to do the sampling with document-wise sampling order. This approach is proposed very recently in [11].

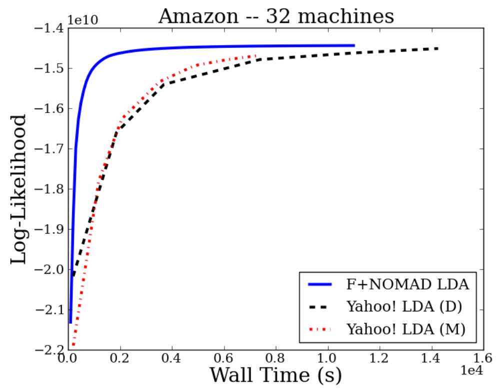
To have a fair comparison focusing on different sampling strategies, we implemented the above three approaches to use the same data structures. We use two smaller datasets—Enron and NyTimes to conduct the experiments. Note that [11] also conducts the comparison of different sampling approaches using these two datasets after further preprocessing. Figure 4 presents the comparison results using $T = 1,024$, while in Figure 5 we show the results by varying T from $1,000$ to $100,000$.

We first compare F+LDA(doc), Sparse LDA, and Alias LDA, where all of the three approaches have the same document-wise sampling ordering. F+LDA(doc) and Sparse LDA follow the exact sampling distribution of the normal Gibbs sampling; as a result, we can observe in Figure 4a and 4b that they have the same convergence speed in terms of number of iterations. On the other hand, Alias LDA converges slightly slower than other approaches because it does not sample from the exact same distribution. Note that we found that this phenomenon becomes more clear when T is large. In terms of efficiency, Figures 4c and 4d indicate that F+LDA(doc) is faster than Sparse-LDA and Alias-LDA, which confirms our analysis in Section 3.

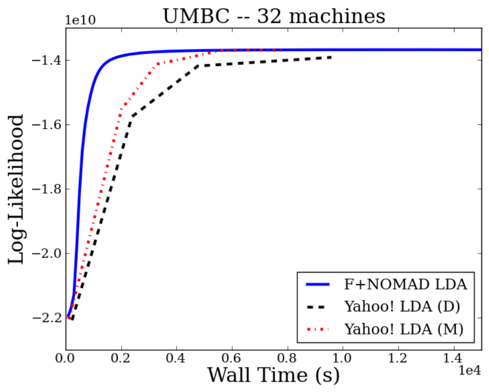
Next we compare the performance of document-wise and word-wise sampling for F+LDA. Figure 4a and 4b indicate that both orderings give similar convergence speed. As discussed in Section 3.2, using the F+tree sampling approach, the word-wise ordering is expected to be faster than document-wise ordering as the number of documents increases. This phenomenon is confirmed by our experimental results in Figures 4c, 4d, and 5a as F+LDA(word) is faster than F+LDA(doc) on the NyTimes dataset, which has a larger number of documents comparing to Enron. The experimental results also justify our use of word-wise sampling

[2]Stanford WebBase project: `http://dbpubs.stanford.edu:8091/~testbed/doc2/WebBase/`

[3]`https://portal.tacc.utexas.edu/user-guides/maverick`



(a)



(b)

Figure 7: The comparison between F+Nomad LDA and Yahoo! LDA on 32 machines with 20 cores per machine.

when applying the Nomad approach in multi-core and distributed systems.

Figure 5 shows the results for even larger T. As the length of the computing time is limited to 12 hours by the Maverick system, the number of iterations is different for all methods. We first observe that when $T \geq 10,000$, AliasLDA starts to outperform SparseLDA in Figures 5a and 5b. However, F+LDA(word) still outperforms these two methods significantly. In 5c, we can see that when T is increased from $1,000$ to $100,000$, the sampling time required by F+LDA(word) increases only by a factor of two. This can be explained by the logarithmic time complexity of F+LDA(word).

5.2 Multi-core and Distributed Experiments

Now we combine our proposed F+tree sampling strategy with the nomadic parallelization framework. This leads to a new F+Nomad LDA sampler that can handle huge problems in multi-core and distributed systems.

5.2.1 *Competing Implementations.*

We compare our algorithm against Yahoo! LDA for three reasons: a) It is one of the most efficient open source implementations of CGS for LDA, which scales to large datasets. b) [16] claims that Yahoo! LDA outperforms other open source implementation such as AD-LDA [13] and PLDA [20]. c) Yahoo! LDA uses a parameter server, which has become a generic approach for distributing large-scale learning problems. It is therefore interesting to see if a different asynchronous approach can outperform the parameter server on this specific problem. Yahoo! LDA is a disk-based implementation that assumes the latent variables associated with tokens in the documents are streamed from disk at each iteration. To have a fair comparison, in addition to running the disk-based Yahoo! LDA (denoted by Yahoo! LDA(D)), we further ran it on the `tmpfs` file system [17] which resides on RAM for the intermediate storage used by Yahoo! LDA. This way we eliminate the cost of disk I/O, and can make a fair comparison with our own code which does not stream data from disk; we use Yahoo! LDA(M) to denote this version.

5.2.2 *Multi-core Experiments*

Both F+Nomad LDA and Yahoo! LDA support parallel computation on a single machine with multiple cores. Here we conduct experiments on two datasets, Pubmed and Amazon, and the comparisons are presented in Figure 6. As can be seen from Figures 6a and 6b, F+Nomad LDA handsomely outperforms both memory and disk version of Yahoo! LDA, and gets to a better quality solution within the same time budget. Given a desired log-likelihood level, F+Nomad LDA is approximately 4 times faster than Yahoo! LDA.

Next we turn out attention to the scaling of F+Nomad LDA as a function of the number of cores. In Figure 6c we plot the convergence of F+Nomad LDA as the number of cores is varied. Clearly, as the number of cores increases the convergence speed is faster.

5.2.3 *Distributed Memory Experiments*

In this section, we compare the performance of F+Nomad LDA and Yahoo! LDA on two huge datasets, Amazon and UMBC, in a distributed memory setting. The number of machines is set to 32, and the number of cores per machine is 20. As can be seen from Figure 7, F+Nomad LDA dramatically outperforms both memory and disk version of Yahoo! LDA and obtains significantly better quality solution (in terms of log-likelihood) within the same wall clock time.

6. CONCLUSIONS

In this paper, we present a novel F+Nomad LDA algorithm that can handle large number of topics as well as large number of documents. In order to handle large number of topics we use an appropriately modified Fenwick tree. This data structure allows us to sample from and update a T-dimensional multinomial distribution in $O(\log T)$ time. In order to handle large number of documents, we propose a novel asynchronous and non-locking parallel framework, which leads to impressive speedups in multi-core and distributed systems. The resulting algorithm is faster than Yahoo! LDA and is able to handle datasets with billions of words. In future work we would like to include the ability to stream documents from disk, just like Yahoo! LDA does. It is also interesting to study how our ideas can be transferred to other sampling schemes such as CVB0.

Acknowledgments. We thank the anonymous reviewers for their constructive comments. We thank Alex Smola for help in implementing AliasLDA. We thank the Texas Advanced Computing Center for providing infrastructure and timely support for our experiments. This research was supported by NSF grants IIS-1219015, CCF-1320746, and CCF-1117055.

7. REFERENCES

[1] A. Asuncion, P. Smyth, and M. Welling. Asynchronous distributed learning of topic models. In *NIPS*, pages 81–88, 2008.

[2] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh. On smoothing and inference for topic models. In *UAI*, pages 27–34, 2009.

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

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[5] P. M. Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24(3):327–336, 1994.

[6] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *KDD*, 2011.

[7] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, 2012.

[8] T. Griffiths and M. Steyvers. Finding scientific topics. *PNAS*, 101:5228–5235, 2004.

[9] G. Heinrich. Parameter estimation for text analysis. Technical report, 2008.

[10] A. Ihler and D. Newman. Understanding errors in approximate distributed latent Dirichlet allocation. *IEEE TKDE*, 24(5):952–960, May 2012.

[11] A. Q. Li, A. Ahmed, S. Ravi, and A. J. Smola. Reducing the sampling complexity of topic models. In *KDD*, 2014.

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

[13] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed algorithms for topic models. *Journal of Machine Learning Research*, 10:1801–1828, 2009.

[14] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[15] B. Recht and C. Ré. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*, 5(2):201–226, June 2013.

[16] A. J. Smola and S. Narayanamurthy. An architecture for parallel topic models. *Proceedings of the VLDB Endowment*, 3(1):703–710, 2010.

[17] P. Snyder. tmpfs: A virtual memory file system. In *Proceedings of the Autumn 1990 European UNIX Users' Group Conference*, 1990.

[18] M. D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering*, 17(9):972–975, 1991.

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

[20] Y. Wang, H. Bai, M. Stanton, W. Chen, and E. Chang. PLDA: Parallel latent Dirichlet allocation for large-scale applications. In *International Conference on Algorithmic Aspects in Information and Management*, 2009.

[21] C. K. Wong and M. C. Easton. An efficient method for weighted sampling without replacement. *SIAM Journal on Computing*, 9(1):111–113, 1980.

[22] F. Yan, N. Xu, and Y. Qi. Parallel inference for latent Dirichlet allocation on graphics processing units. In *NIPS*, pages 2134–2142, 2009.

[23] L. Yao, D. Mimno, and A. McCallum. Efficient methods for topic model inference on streaming document collections. In *KDD*, 2009.

[24] H.-F. Yu, C.-H. Ho, Y.-C. Juan, and C.-J. Lin. Libshorttext: A library for short-text classification and analysis. Technical report, 2013.

[25] H. Yun, H.-F. Yu, C.-J. Hsieh, S. V. N. Vishwanathan, and I. S. Dhillon. NOMAD: Non-locking, stOchastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion. *Proceedings of the VLDB Endowment*, 7(11):975–986, 2014.