Parallelized Variational EM for Latent Dirichlet Allocation: An Experimental Evaluation of Speed and Scalability

Ramesh Nallapati, William Cohen and John Lafferty
Machine Learning Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
{nmramesh,wcohen,lafferty}@cs.cmu.edu

Abstract

Statistical topic models such as the Latent Dirichlet Allocation (LDA) have emerged as an attractive framework to model, visualize and summarize large document collections in a completely unsupervised fashion. Considering the enormous sizes of the modern electronic document collections, it is very important that these models are fast and scalable. In this work, we build parallel implementations of the variational EM algorithm for LDA in a multiprocessor architecture as well as a distributed setting. Our experiments on various sized document collections indicate that while both the implementations achieve speed-ups, the distributed version achieves dramatic improvements in both speed and scalability. We also analyze the costs associated with various stages of the EM algorithm and suggest ways to further improve the performance.

1 Introduction

Recently, statistical topic modeling has been proposed as a completely unsupervised method to help summarize and visualize the contents of large document collections [1, 2]. These models use simple surface features such as word occurrences within documents to reveal suprisingly meaningful semantic content of documents.

The basic version of this family of models is called *Latent Dirichlet Allocation* (LDA)[1]. In this model, each document d is assumed to be generated from a K-component mixture model, where the mixing probabilities $\boldsymbol{\theta}_d$ for each document are governed by a global Dirichlet distribution with parameter α. Each of the words w_{di} in the document is generated by a repetitive process of sampling a mixture component z_{di} from $\boldsymbol{\theta}_d$ and then sampling the word itself from a multinomial distribution over the entire vocabulary $\boldsymbol{\beta}_{z_{di}}$, associated with the mixture component z_{di}. The generative process and the graphical representation of LDA are shown in Figure 1 and Table 1 respectively.

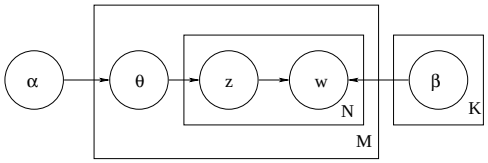

Figure 1. Graphical representation of LDA: each plates represents repeated sampling as many times as the number in the right-hand bottom corner of the plate.

- For each document $d = 1, \cdots, M$
  - sample mixing proportions $\boldsymbol{\theta}_d \sim \text{Dir}(\cdot|\alpha)$
  - For each position $i = 1, \cdots, N_d$, sample:
    - topic $z_{di} \in \{1, \cdots, K\} \sim \text{Mult}(\cdot|\boldsymbol{\theta}_d)$
    - word $w_{di} \in \{1, \cdots, V\} \sim \text{Mult}(\cdot|\boldsymbol{\beta}_{z_{di}})$

Table 1. Generative process of LDA: V is the vocabulary size and K is the number of topics while α is the symmetric Dirichlet parameter and $\{\boldsymbol{\beta}_1, \cdots, \boldsymbol{\beta}_K\}$ are the multinomial topic parameters. The only input to the model is K, the number of topics. All the parameters can be estimated by approximate variational techniques or Gibbs sampling.

Each of the K multinomial distributions $\boldsymbol{\beta}_k$ assigns a high probability to a specific set of words that are semantically coherent. Hence these distributions over the vocabulary are referred to as *topics* and represent a concise summary of the contents of the document collection. A few representative topics learnt from the AP corpus are displayed in figure 2. Note that learning the parameters of the LDA model is intractable in general [1]. However, approximations have been proposed that allow efficient unsupervised parameter estimation. One of them is the mean-field variational EM algorithm [1] and the other is stochastic EM Gibbs sampling [2]. The former is a deterministic approximation method that results in a biased estimate, but is com-

"ARTS"	"BUDGET"	"CHILDREN"	"EDUCATION"
New	Million	Children	School
Film	Program	Women	Students
Show	Tax	People	Schools
Music	Budget	Child	Education
Movie	Billion	Years	Teachers
Play	Federal	Families	High
Musical	Year	Work	Public
Best	Spending	Parent	Teacher
Actor	New	Says	Bennett
First	State	Family	Manigat
York	Plan	Welfare	Namphy
Opera	Money	Men	State
Theater	Programs	Percent	President
Actress	Government	Care	Elementary
Love	Congress	Life	Haiti

Figure 2. Most likely words from 4 topics in LDA from the AP corpus: the topic titles in quotes are not part of the algorithm. (Courtesy: Blei *et al*, [1])

putationally efficient with well defined numerical convergence criteria. The latter is a probabilistic technique that asymptotically converges to an unbiased solution, but its convergence has to be assessed in terms of stochastic convergence. In this work, we use the mean-field variational EM technique.

2 Variational EM in brief

In broad terms, the variational EM algorithm simplifies the problem by approximating the posterior distribution of the hidden variables given the data $P(\boldsymbol{\theta}, \mathbf{z}|\mathbf{w}, \alpha, \boldsymbol{\beta})$ by a more tractable, conditionally independent variational distribution as shown below:

$$Q(\boldsymbol{\theta}, \mathbf{z}|\boldsymbol{\gamma}, \boldsymbol{\phi}) = \prod_{d=1}^{M} \left\{ q(\boldsymbol{\theta}_d|\boldsymbol{\gamma}_d) \prod_{i=1}^{N_d} P(z_{di}|\boldsymbol{\phi}_{di}) \right\} \quad (1)$$

It can be shown using Jensen's inequality that the log-likelihood of the observed data $\log P(\mathbf{w}|\alpha, \boldsymbol{\beta})$ is lower-bounded by the sum of the expected-log-likelihood of the complete data with respect to the variational distribution $E_Q[\log P(\mathbf{w}, \mathbf{z}, \boldsymbol{\theta}|\alpha, \boldsymbol{\beta})]$ and the entropy of the variational distribution $H(Q)$, as shown below:

$$\log P(\mathbf{w}|\alpha, \boldsymbol{\beta}) \geq E_Q[\log P(\mathbf{w}, \mathbf{z}, \boldsymbol{\theta}|\alpha, \boldsymbol{\beta})] + H(Q) \quad (2)$$

The variational EM proceeds iteratively in two steps as follows: In the E-step, we estimate the variational parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\phi}$ by maximizing the RHS of Eq. (2) with respect to them. Since $\boldsymbol{\phi}_d$ and $\boldsymbol{\gamma}_d$ for a given document d depend on each other, we estimate them iteratively until the lower-bound on the log-likelihood of the document converges to a local maximum. The algorithm for E-step is shown in Table 2. In the M-step, the lower-bound is maximized with respect to the parameters $\boldsymbol{\beta}$ and α to obtain their new values. The M-step is summarized in table 3.

0.	**Input:** Model parameters $\boldsymbol{\beta}, \alpha$
1.	For each document $d \in 1, \cdots, M$
2.	For each topic $k \in 1, \cdots, K$
3.	Initialize $\gamma_{dk} = N_d/K$
4.	Repeat until convergence:
5.	For each position $i = 1, \cdots, N_d$
6.	For each topic $k = 1, \cdots, K$
7.	$\phi_{dik} = \beta_{kw_i} \exp(\psi(\gamma_{dk}))$
8.	Normalize $\boldsymbol{\phi}_{di}$
9.	For each topic $k = 1, \cdots, K$
10.	$\gamma_{dk} = \alpha + \sum_{i=1}^{N_d} \phi_{dik}$
11.	**Return**:
	Sufficient statistics matrix S where $S_{kv} = \sum_{d=1}^{N_d} \sum_{i=1}^{N_d} \delta_v(w_{di}) \phi_{dik}$
	α-sufficient statistics S_α where $S_\alpha = \sum_{d=1}^{M} \sum_{k=1}^{K} (\psi(\gamma_{dk}) - K\psi(\sum_{k=1}^{K} \gamma_{dk}))$

Table 2. Summary of E-step: ψ is the Digamma function and $\delta_v(w) = 1$ if $w = v$ and 0 otherwise. $\boldsymbol{\phi}$ and $\boldsymbol{\gamma}$ are the variational parameters as defined in Eq. 1.

0.	**Input:** Sufficient statistics S and S_α
1.	Compute α using Newton-Raphson with S_α as input.
2.	Compute $\boldsymbol{\beta} = S$-normalized-row-wise.
3.	**Return:** Model parameters $\boldsymbol{\beta}, \alpha$.

Table 3. Summary of the M-step

A pair of E and M-steps comprises a single iteration of the EM algorithm. Each iteration of the variational EM algorithm is guaranteed to increase the lower-bound on the log-likelihood of the observed data. The iterations are terminated when the lower-bound of the observed data log-likelihood (RHS of Eq. (2)) converges to a local maximum. Interested readers may refer to [1] for more technical details of the algorithm.

3 Parallelized LDA

Note that for a document d of length N_d, each E-step takes $\mathcal{O}((N_d+1)K)$ computations[1]. Empirical experiments suggest that the number of variational iterations required is roughly on the order of N_d for each document [1]. Hence, the total complexity of the E-step for each iteration is on the order of MN_{max}^2K where M is the corpus size and N_{max} is the maximum document length in the corpus. Clearly, the E-step is a computational challenge as its complexity grows linearly with the collection size M and quadratically with maximum document length N_{max}.

The complexity of the M-step, on the other hand, is $\mathcal{O}(VK)$ where V is the observed vocabulary size, as it involves normalizing the *sufficient statistics* S_{kv} =

[1] Specifically, $\mathcal{O}(N_dK)$ to compute $\boldsymbol{\phi}_d$ and $\mathcal{O}(K)$ to compute $\boldsymbol{\gamma}_d$.

$\sum_{d=1}^{M} \sum_{i=1}^{N_d} \phi_{dik} \delta_v(w_{di})$ that are already computed in the E-step, to obtain β_{kv} for each word and topic pair (v, k). This is not a serious computational challenge since elementary array operations are extemely fast. The other part of M-step, involving a linear time Newton-Raphson method to compute α, is also computationally inexpensive.

Thus, the main computational bottleneck in the EM algorithm is the E-step, making the algorithm hard to scale to large document collections. Fortunately, there is a way to address this issue: the variational parameters $\boldsymbol{\gamma}_d$ and $\boldsymbol{\phi}_d$ in each document d are independent of those in other documents and therefore can be computed independently. In other words, the E-step computations of the documents can be speeded up through parallelization. This is the key fact we use in parallelizing LDA.

3.1 Multiprocessor implementation

In a multiprocessor architecture, we have multiple CPUs that share the same RAM. In this setting, variational EM can be implemented efficiently using parallel threads. In our implementation, the main program first assigns pointers to one unique subset of the corpus to each thread. A single iteration of the EM algorithm proceeds as follows. Each thread performs the E-step on its subset and returns its sufficient statistics matrix. The main program then aggregates these statistics and estimates the model parameters in the M-step. The thread-architecture makes the memory management efficient by allowing a single copy of $\boldsymbol{\beta}$ and D matrices to be shared across all the threads. A detailed step-by-step algorithm in presented in Table 4.

3.2 Distributed implementation

In the distributed setting, we have a cluster of machines that share the same disk but do not have a shared memory. One of the machines plays the role of a *master* and the other nodes act as *workers*. In this architecture, the master node physically splits the data into W subsets and writes them to disk. In each EM iteration, the worker nodes load their respective data subsets and the mdeol computed in the last iteraion into their memories. Upon performing the E-step on their respective subsets, they collect their sufficient statistics and print them in a sparse manner onto the disk. The master node then aggregates the outputs of all the worker nodes. Finally, the master node performs the M-step and prints out the new model onto the disk. This iterative process is repeated until convergence as shown in Table 5.

4 Experimental Details

4.1 Hardware

For our multiprocessor implementation, we used a Linux machine with 4 CPUs, each of which is an Intel Xeon 2.40GHz processor. The processors share a RAM of size 4GB and a 512KB cache.

Our distributed implementation used a cluster with 96 nodes that share the same disk and each of which is a Linux machine equipped with a Transmetta Efficeon TM8000 1.2GHz processor with 1MB of cache and 1GB RAM.

4.2 Software

For our parallel LDA implementations, we used the C-code of Blei[2] as the core. We extended this code to the multiprocessor architecture using *pthreads*.

The distributed implementation uses a main process written in *Perl* that co-ordinates the worker nodes. This process runs on the master node and uses *rsh* connections to invoke the worker nodes. The worker nodes execute the E-step (implemented in C) and signal their completion by creating empty files on the disk, which are inspected for by the main process periodically. Upon signals from all the workers, it calls the M-step (implemented in C) at the master node.

4.3 Data

We used a subset of the *PubMed*[3] collection consisting of about 300,000 documents as our primary dataset. We indexed the collection using *Lemur*[4] after stopword removal and stemming of the collection. The vocabulary of this collection consists of about 100,000 unique words. We then randomly sampled from this primary collection to generate sub-collections of various sizes. The vocabulary size of all these smaller collections remains the same.

4.4 Experimental setup

In our experiments, we primarily studied the effect the number of threads (nodes) has on the runtime of the LDA algorithm for various collection sizes, keeping the vocabulary size V fixed at nearly 100,000 and number of topics K fixed at 50. We also fixed the variational convergence factor at 10^{-6} and the EM convergence factor[5] at 10^{-3}. In the multiprocessor setting, we varied the number of threads from 1 to 4, since we had only 4 CPUs in the hardware. In the distributed setting, we varied the number to nodes starting from 1 and reaching as high as 90.

For our experiments on a given sub-collection, we first used a randomly initialized model and ran LDA for just one EM iteration to obtain a new estimate of the model. We

[2]Downloadable from http://www.cs.princeton.edu/∼blei/lda-c/

[3]http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed

[4]http://www.lemurproject.org

[5]These factors represent the stopping criteria for the inner variational iterations and outer EM iterations respectively: when the ratio of log-likelihood of observed data in two consecutive iterations is smaller than the factor, the iterations are terminated.

0. **Input:** Sparse Doc-Term Matrix D of size $(M \times V)$, num. topics K, num. threads P.
1. Randomly Initialize model parameters $\boldsymbol{\beta}$ of size $(K \times V)$ and α .
2. Split D row-wise into P equal sized disjoint subsets $\{D_1, \cdots, D_P\}$ each of maximum size $(\lceil M/P \rceil \times V)$.
3. Do *Until* Convergence
4. For each thread $p \in \{1, \cdots, P\}$
5. Initialize sufficient statistics matrix S_p of size $(K \times V)$ to **0**.
6. Perform the variational *E-step* for the subset D_p as described in table 2.
7. return S_p and S_{α_p}.
8. Aggregate $S = \sum_{p=1}^{P} S_p$ and $S_\alpha = \sum_{p=1}^{K} S_{\alpha_p}$.
9. Perform *M-step* as described in table 3.
10. **Return:** $\boldsymbol{\beta}, \alpha$.

Table 4. Parallelization of LDA in a multiprocessor setting

MASTER NODE

0. **Input:** Sparse Doc-Term Matrix D of size $(M \times V)$, num. topics K, num. worker nodes W.
1. Randomly Initialize model parameters $\boldsymbol{\beta}$ of size $(K \times V)$ and α.
2. Split D row-wise into W equal sized disjoint subsets $\{D_1, \cdots, D_W\}$ each of maximum size $(\lceil M/W \rceil \times V)$ and write them to disk.
3. Do *Until* Convergence
   Call the W worker nodes and wait until each of them finishes its job.
4. Read from disk and aggregate $S = \sum_{w=1}^{W} S_w$ and $S_\alpha = \sum_{w=1}^{W} S_{\alpha_w}$.
5. Perform *M-step* as described in table 3 and write $\boldsymbol{\beta}, \alpha$ to disk.
6. **Return:** $\boldsymbol{\beta}, \alpha$.

WORKER NODE w $\in \{1, \cdots, W\}$

1. Load Doc-Term matrix D_w into memory.
2. Load Model $\boldsymbol{\beta}$ and α into memory.
3. Initialize local sufficient statistics matrix S_w of size $(K \times V)$ to **0**.
4. Perform the variational *E-step* for the subset D_w as described in table 2 .
5. Write sparse S_w and S_{α_w} to disk.
6. Signal Completion.

Table 5. Parallelization of LDA in a distributed setting

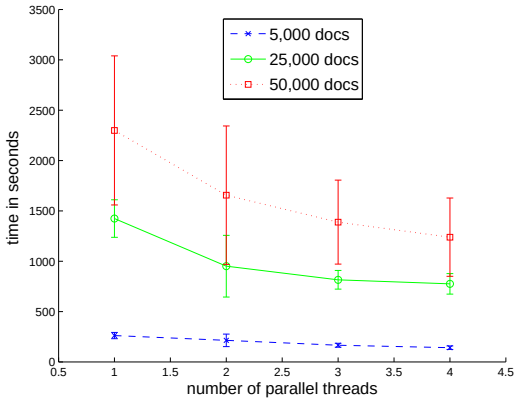

Figure 3. Multiprocessor LDA: Runtime as a function of number of threads

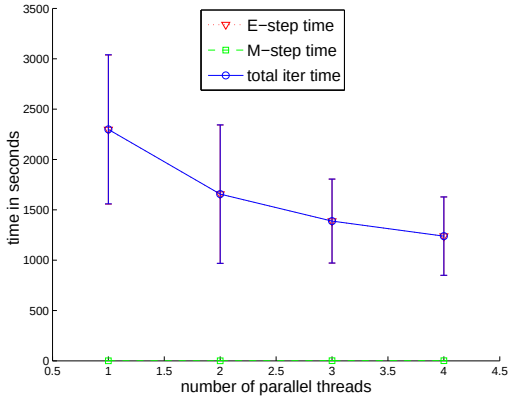

Figure 4. Multiprocessor LDA: contribution of E and M steps to runtime for the sub-collection of size 50,000 documents. Note that the E-step time and the total iteration time are nearly the same and hence cannot be distinguished in the plot.

used this model as the starting point for all our runs on this sub-collection [6]. For each run, we reported the average run-time per EM iteration instead of total run-time until convergence[7]. Run-time of each EM iteration is measured as the sum of the average E-step times of all the worker nodes and the M-step time at the master node. We also report error-bars of width twice the standard deviation in our plots.

5 Results and Discussion

Figure 3 displays the average EM iteration time for the multi-processor implementation for subcollections of 3 different sizes, as a function of number of parallel threads. Notice that iteration time decreases monotonically as we go from 1 to 4. However, this decrease is not linear as one would expect. We believe this is primarily because of the conflict between the threads in read-accessing the model β located in the main memory, during the E-step.

Figure 4 shows the break-up of the runtime between E and M steps for a specific sub-collection. The plot confirms our expectation in section 3 that E-step is the main (and perhaps the only) bottleneck in the algorithm.

Figure 5 shows the average EM iteration time as a function of number of nodes in the distributed implementation. In this case, we were able to achieve substantial reduction in computation time due to the larger number of nodes available. For example, for the collection of size 50,000 documents, the multiprocessor implementation achieved a speedup of only 1.85 from 1 to 4 threads while the

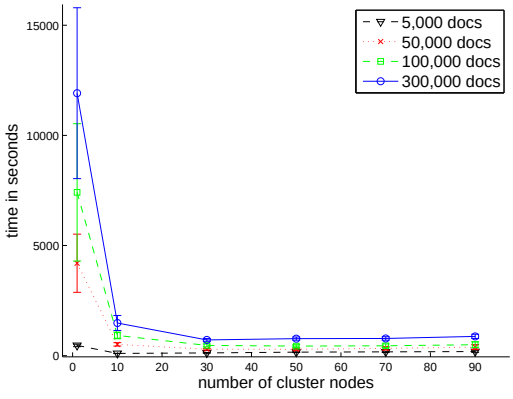

Figure 5. Distributed LDA: Runtime as a function of number of nodes

[6] Since the objective function of LDA is non-convex, the final solution and the number of EM iterations required for convergence depends on the starting point. Starting from the same initial point ensures that the final output and the amount of computation is same for all runs.

[7] Since the number of iterations until convergence may vary across sub-collections, total run-time is not directly comparable across sub-collections of different sizes.

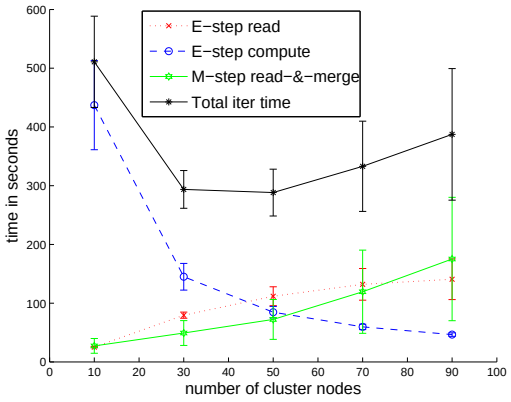

Figure 6. Distributed LDA: contribution of various components to runtimefor the sub-collection of size 50,000 documents.

distributed implementation achieved a significantly higher speedup of 14.5 from 1 to 50 nodes. As a result, we were able to scale-up our experiments on the distributed implementation to larger collection sizes as shown in figure 5.

Although not clear from the plots in Figure 5, our data indicates that the optimum number of nodes required for maximum speedup increases with the collection size. Hence it is desirable to scale the cluster size as the data size grows.

We noticed that the run-time in the distributed setting decreases monotonically until we reach an optimal number of nodes and then increases steadily again (not very discernible from figure 5). In order to understand this phenomenon, we plotted the average run-times of various components of the EM algorithm that consume most of the runtime, in figure 6. The plot shows that while the E-step computation time (step 4 of the worker node in Table 5) goes down with increasing number of nodes as expected, two other components namely, the E-step read time (steps 1 and 2 of the worker node in Table 5) and the M-step read time (step 4 of the master node in Table 5) increase with higher number of nodes. We believe this is due to the conflict between the nodes in disk read-access.

Notice also that the rate of increase of M-step read time is higher that that of the E-step read time. Our investigation revealed an additional overhead in this case, which can be explained as follows: as shown in step 4 of the master node in Table 5, the master node reads files containing sparse sufficient statistics S_w output from all the worker nodes $w \in \{1, \cdots W\}$ and aggregates them together. Each S_w file is sparse, which means it contains an entry for a word v only if it occurs in the corresponding document subset D_w. Thus, the number of entries to be read for a given word is equal to the number of document subsets D_w it occurs in. As the number of worker nodes W grows, each word occurs in increasing number of subsets, thereby increasing the total number of S_w entries for itself. Hence the cumulative size of the S_w files grows with increasing W, which explains the steeper increase in read-time compared to E-step. In the latter case, the input file size, that of β, α is constant with respect to W.

6 Conclusion and Future Work

Our experiments demonstrate that while the multiprocessor implementation speeds up LDA, the gain is not significant because of the read-conflict between various threads. In addition, this implementation stores the entire data in memory, so it may not scale to large collections in its existing form. Finally, the prohibitive costs of multiple processor machines also make them less desirable.

The distributed implementation on the other hand, shows tremendous promise of scalability and speed, achieving speedup ratio of 16 or more on a document collection of size 300,000 documents. It scales well to large collections because it splits the data between its worker nodes. It is also the more desirable architecture due to its low costs.

We note that more efficient distributed implementations are possible using sophisticated message passing protocols, but we used this rudimentary disk-based interaction as a proof-of-concept for LDA.

One trick to reduce the steep overhead in the M-step read-time is to split the document term matrix between the worker nodes (in step 2 of the master node in Table 5) by using a word-similarity based clustering of the documents, instead of doing a random split. This would act as a natural compression by reducing occurrences of any given word in multiple S_w matrices, thereby reducing the disk read-access time in the M-step.

On similar lines, one could reduce the E-step read-time at a worker node w by loading only the model parameters that are relevant to the words that occur in the corresponding document subset D_w. These are the only model parameters that would be used in estimating the variational parameters in the E-step (see Table 2).

Another trick to achieve scalability is not to load the entire document matrix D_w into memory at each worker node w, but to read one document at a time, perform E-step on it and output its sufficient statistics to the disk, in a pipelined fashion.

We hope to implement these improvements as part of our future work and scale our experiments to larger collections.

References

[1] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3, 2003.

[2] T. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 2004.