**Parallelism for LDA**

*Yang Ruan, Changsi An*

*(yangruan@indiana.edu, anch@indiana.edu)*

## 1. Overview

As parallelism is very important for large scale of data, we want to use different technology to parallelize the latent dirichlet allocation algorithm. Latent dirichlet allocation (LDA) is a generative model that allows sets of observations to be explained by unobserved groups which explain why some parts of the data are similar.

There are two different implementation of this model, one is Expectation Maximum (EM method) and the other is Gibbs Sampling. Both of them use iterative algorithm to do the calculation, however, Gibbs Sampling is easier to implement in sequential code but harder to parallel while the EM method is easier to parallel as (Nallapati, Cohen et al.) has already use MapReduce technology to parallelize that. And we are going to parallel the Gibbs sampling implementation this time using iterative MapReduce and message passing interface.

## 2. Algorithm

### 2.1 Guideline of Sequential Algorithm with LDA

LDA is a generative probabilistic model of a corpus. The basic idea is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words(Blei, Ng et al. 2003).

To clarify this model, assume we have the following convention:

A corpus D : $\{d_1, d_2, .. d_m\}$. Each corpus $d_i$ : $\{w_1, w_2, ..., w_{ni}\}$ is a sequence of words. In contrast to words, we have vocabulary V : $\{v_1, v_2, ..., v_l\}$ which is a non-duplicated projection of all words. Also we assume there exist K topics T = $\{t_1, t_2, ..., t_k\}$.

Apart from the above notation, we also have two variable parameters: $\alpha$ which is used as a parameter in Dirichlet distribution to generate topics distribution and $\beta$ is a conditional probability matrix, which records the probability of a word given a specific topic.

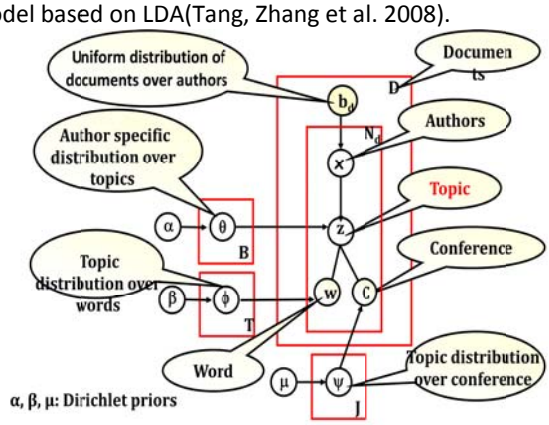
The algorithm with LDA model has two goals. One goal is to generate a topics distribution and associated words list for each topic, and give out the probability of the generated case. Another goal is to update the parameters $\alpha$ and $\beta$ in the generation process.

The algorithm can be described in the following steps: first choose $\theta \sim Dirchlet(\alpha)$. Then for each word $w_i$ of document $d_i$, sample a topic $T_n$ which follows multinomial distribution with parameter $\theta$, after that, sample a word $w_i'$ according to $p(w \mid T_n, \beta)$. After that, if $w_i'$ is not $w_i$, we will update $\beta$ by augmenting the probability of word $w_i'$ to topic $T_n$ and diminishing the one of word $w_i$.

The probability of the generated case can be calculated by multiplying the joint probabilities of occurred events in the sampling processes.

### 2.2 ACT model

Author Conference Topic model (ACT) is an extension model based on LDA(Tang, Zhang et al. 2008).




Because this model also adds conference and author as two new variables, so the size of the computation increased from 2 2-D matrixes to 3 2-D matrixes. We will have topic-author, word-topic and conference-topic matrixes to do the Gibbs sampling. And the sampled result (matrix contains the frequency) can be used in the computation to the final probability result: $\theta$ is the probability of a topic given an author; $\phi$ is the probability of a word given a topic; $\psi$ is the probability of a conference given a topic.

## 3. Technology

This time we are going to use an iterative MapReduce framework (twister) and message passing interface (MPI). The reason for choosing twister is the algorithm is done in iterative pattern and it will be much faster to be parallelized under twister which is specially designed for it than normal MapReduce framework like Hadoop. This has already been demonstrated by Jaliya et al. And MPI has always be a standard parallelism method for years. By using it we can have a control group where there will be a comparison between this two technologies.

Twister is an enhanced MapReduce runtime with an extended programming model that supports iterative MapReduce computations efficiently(J.Ekanayake, H.Li et al. 2010).

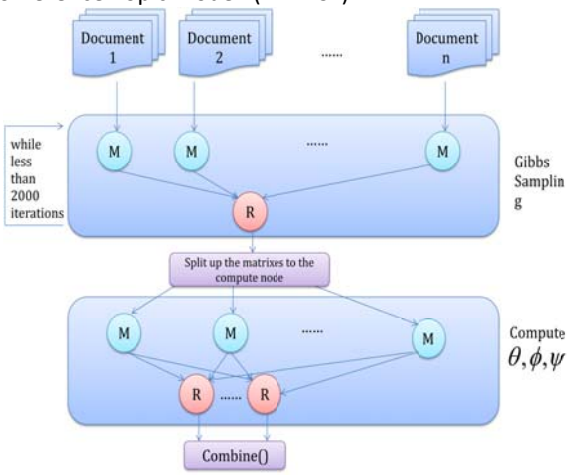
To use that feature, the iterative part of the algorithm needs to be done within the program control of the iteration. Here is a figure of the parallelism of the program.

There are two main parts of this program, the first part is the Gibbs sampling which is an iterative part of the program and the second none-iterative part is to use this 3 2-D matrixes to calculate the final probability distribution.

In the first part, for each mapper, there will be 3 split up 2-D matrixes in each mappers' constant memory which are generated directly by each document chunk. In each

iteration, the reducer will broadcast the merged matrixes onto each mapper.

We call this method Approximate Distributed Author Conference Topic model. (AD-ACT)

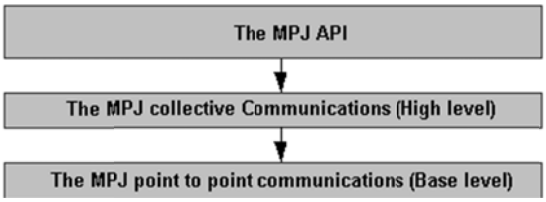



By using this pattern, we can save the memory space on each mapper by exchange the time of sending the 1-D matrix to each node.

MPJ Express is a MPI extension of Java implementation. It is capable to run on distributed machines and multicore processors. The usage of it is also daemon control similar to Twister and with command line launcher.

Here is a figure and some API of it:

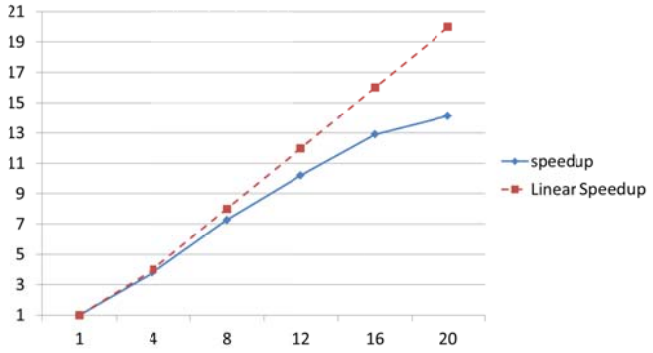
*P2P Communication*
*Comm.Send() , Comm.Recv()*
*Comm.Isend(), Comm.Irecv()*
*Collective Communcation*
*Intracomm.Bcast()*
*Intracomm.Gather()*
*Process Management*
*Intra-communication(within same Group) : Intracomm*
*Inter-communication(server-client, Cartesian topology): Cartcomm*




## 4. Conclusion

The performance of Twister-ACT is not as good as expected, since while we implement this parallel program, we found out that every time we sample a single element inside the matrixes, there will be some influence to the following sampling. So the results can never be as same as the sequential one if we do parallel computing on the ACT gibbs sampling. Basically, to minimize this problem, we have to do iterative mapreduce. And keep the each block of document on each mapper's memory cache. Still we have a large quantity of message to pass since every mapper requires the full sampling matrixes. Still, we can solve some of the memory problem by splitting up the documents, so each mapper only gets a partition of the full document.

And Here is the performance chart:




*Speed up Test of 8k documents running on PolarGrid in Indiana University using 1 to 20 mappers*

**References:**

*Blei, D. M., A. Y. Ng, et al. (2003). "Latent Dirichlet allocation." Journal of Machine Learning Research **3**: 993-1022.*

*J.Ekanayake, H.Li, et al. (2010). Twister: A Runtime for iterative MapReduce. Proceedings of the First International Workshop on MapReduce and its Applications of ACM HPDC 2010 conference June 20-25, 2010. Chicago, Illinois, ACM.*

*Nallapati, R., W. Cohen, et al. Parallelized Variational EM for Latent Dirichlet Allocation: An Experimental Evaluation of Speed and Scalability. Pittsburgh, USA, Carnegie Mellon University.*

*Tang, J., J. Zhang, et al. (2008). ArnetMiner: Extraction and Mining of Academic Social Networks. KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.*