IEICE TRANS. INF. & SYST., VOL.E96–D, NO.5 MAY 2013

PAPER Special Section on Data Engineering and Information Management

MPI/OpenMP Hybrid Parallel Inference Methods for Latent Dirichlet Allocation — Approximation and Evaluation

Shotaro TORA[†a)], ***Nonmember and*** **Koji EGUCHI**[†], ***Member***

SUMMARY Recently, probabilistic topic models have been applied to various types of data, including text, and their effectiveness has been demonstrated. Latent Dirichlet allocation (LDA) is a well known topic model. Variational Bayesian inference or collapsed Gibbs sampling is often used to estimate parameters in LDA; however, these inference methods incur high computational cost for large-scale data. Therefore, highly efficient technology is needed for this purpose. We use parallel computation technology for efficient collapsed Gibbs sampling inference for LDA. We assume a symmetric multiprocessing (SMP) cluster, which has been widely used in recent years. In prior work on parallel inference for LDA, either MPI or OpenMP has often been used alone. For an SMP cluster, however, it is more suitable to adopt hybrid parallelization that uses message passing for communication between SMP nodes and loop directives for parallelization within each SMP node. We developed an MPI/OpenMP hybrid parallel inference method for LDA, and evaluated the performance of the inference under various settings of an SMP cluster. We further investigated the approximation that controls the inter-node communications, and found out that it achieved noticeable increase in inference speed while maintaining inference accuracy.

key words: *probabilistic topic models, Latent Dirichlet allocation, Gibbs sampling, MPI/OpenMP hybrid parallelization*

1. Introduction

Topic modeling is one of the most successful approaches to analyzing large document collections. Topic models are based on the idea that each document is generated from a mixture of word distributions, each of which is called a "topic". Latent Dirichlet allocation (LDA) [1] is a well known topic model. We can use the LDA model or its variants for not only text data but also image data [2], [3], network data [4], [5], and others. However, inference of unknown parameters of the LDA model on large-scale data brings significant challenges in terms of computation time and memory requirements. For this purpose, some methods for increasing the inference speed have been proposed via parallelization such as approximate distributed inference for LDA (AD-LDA) [6] and asynchronous distributed learning algorithm for LDA (Async-LDA) [7].

Parallel computing architectures can be divided into three classes, distributed memory, symmetric multiprocessing (SMP), and SMP cluster. To optimize the performance and resources of these architectures, a promising approach is hybrid parallel computing, which uses message passing for the communication between SMP nodes in a cluster and parallel computing based on shared memories in each node. We use Message Passing Interface (MPI) for the message passing between SMP nodes and Open Multi-Processing (OpenMP) for parallelization within each SMP node for the inference of the LDA model.

In parallel computing using only MPI, processor cores communicate with each other by message passing, even though they belong to the same node, just like on distributed memory systems. In the MPI/OpenMP hybrid parallel method, each processor core in an SMP node refers to data on a shared memory, and only one processor core on each node communicates with the others. In general, it is more efficient for processor cores to communicate via a shared memory than by message passing. When data are transferred by the MPI function, such as 'MPI_Allreduce', the fewer processors that take part in communication, the less time it takes. Therefore, using MPI/OpenMP hybrid parallelization on an SMP cluster is expected to improve the speed of learning the LDA model. However, there have been no published reports, to our knowledge, on detailed performance evaluation using MPI/OpenMP hybrid parallelization with an SMP cluster, for the inference of the LDA model.

We develop an MPI/OpenMP hybrid parallel inference method for the LDA model, and evaluate the performance of the inference under various settings of an SMP cluster, compared with using only an MPI or a single processor. We further investigate the approximation that controls the inter-node communications, and demonstrate that it achieves significant increase in inference speed while maintaining inference accuracy.

2. Related Work

2.1 LDA

LDA is a widely accepted topic model and is based on the idea that each document is generated from a mixture of multinomial distributions over words, each of which is called a 'topic' and represents a cluster of words that co-occur across different documents. In this model, Dirichlet priors are assumed to correspond to each document's topic multinomial and each topic's word multinomial. Figure 1 shows a graphical model representation of LDA, and the following is the process of generating documents. Here, the notations are given Table 1.

1. For document d, multinomial parameters θ_d are drawn from Dirichlet prior distribution $Dir(\alpha)$.

Manuscript received July 3, 2012.
Manuscript revised October 29, 2012.

[†]The authors are with the Kobe University, Kobe-shi, 657–8501 Japan.
a) E-mail: tora@cs25.scitec.kobe-u.ac.jp

DOI: 10.1587/transinf.E96.D.1006

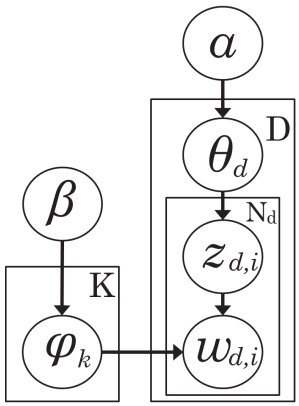


Fig. 1 Graphical model representation of LDA.

Table 1 Description of variables for LDA.

Variable	Description
D	Number of documents in collection
K	Number of topics
V	Number of word types (vocabulary size) in collection
N_d	Number of words in document d
$w_{d,i}$	i^{th} observed word in document d
$z_{d,i}$	Topic assigned to $w_{d,j}$
ϕ_k	Multinomial parameters associated with topic k
θ_d	Multinomial parameters associated with document d
α	Hyperparameter of Dirichlet prior for θ.
β	Hyperparameter of Dirichlet prior for ϕ.

2. For topic k, multinomial parameters ϕ_k are drawn from Dirichlet prior distribution $Dir(\beta)$.
3. For the i-th word $w_{d,i}$ in document d:
   - topic $z_{d,i}$ is drawn from multinomial distribution $Mult(\boldsymbol{\theta}_d)$
   - word $w_{d,i}$ is drawn from multinomial distribution $Mult(\boldsymbol{\phi}_{z_{d,i}})$

The posterior distributions can be estimated such as using the collapsed Gibbs sampling method, which updates each topic assignment $z_{d,i}$ by sampling with the following full conditional posterior probability in each iteration [8], [9]:

$$p(z_{d,i} = k | w_{d,i} = v, \mathbf{W}_{-(d,i)}, \mathbf{Z}_{-(d,i)}, \alpha, \beta) \propto (C^{doc}_{d,k} - 1 + \delta_{t \neq t'} + \alpha) \cdot \frac{C^{word}_{k,v} - 1 + \delta_{t \neq t'} + \beta}{\sum_{v'} C^{word}_{k,v'} - 1 + \delta_{t \neq t'} + V\beta} \quad (1)$$

Here, we assume that the Dirichlet priors α and β are symmetric. $C^{word}_{k,v}$ and $C^{doc}_{d,k}$ stand for the (k, v)-element of word-topic count matrix $\mathbf{C}^{word}$ and the (d, k)-element of document-topic count matrix $\mathbf{C}^{doc}$, respectively. $\mathbf{W}_{-(d,i)}$ denotes the document-word matrix in the document collection excluding word $w_{d,i}$, and $\mathbf{Z}_{-(d,i)}$ represents the corresponding topic assignments of $\mathbf{W}_{-(d,i)}$. The indicator function δ takes the value 1 when the designated event occurs; otherwise, 0. The sampling procedure starts with random topic assignments. Then the posterior distributions given by the procedure above with a sufficient number of iterations will converge according to the dependencies between random variables.

2.2 Fast Inference Methods for LDA

The computational complexity of collapsed Gibbs sampling is given by the number of topics multiplied by the vocabulary size in the document collection. There have been prior studies that attempted to improve the inference speed for the LDA model at various strategic points.

- Newman et al. [6] proposed approximate distributed inference methods for LDA: AD-LDA and HD-LDA, using collapsed Gibbs sampling for distributed memory systems. AD-LDA increases inference speed without losing accuracy, even though it is an approximate method. HD-LDA is theoretically equivalent to estimating a mixture of LDA models; however, it incurs high computational cost. Our inference method described in Sect. 3 can be positioned as an extension of AD-LDA.
- Yi Wang et al. [10] implemented Parallel LDA (PLDA) on multiple processors by MPI and by MapReduce. PLDA's algorithm is equivalent to AD-LDA's. They reported that MPI-PLDA was faster than MapReduce-PLDA because the latter involves machine scheduling and disk I/O between iterations. PLDA can considerably reduce the total running time; however, it is assumed that all processors are independent within and across nodes; thus, requiring global synchronization between them at each iteration. Therefore, the communication cost increases as the number of processors increases. We address this problem by MPI/OpenMP hybrid parallelization and assuming the use of an SMP cluster comprising multi-core processors.
- Asuncion et al. [7] investigated Async-LDA. The data are distributed to processors, each of which independently performs collapsed Gibbs sampling, but the processors communicate with each other asynchronously. The global synchronization phase of AD-LDA in each Gibbs sweep (each iteration in Gibbs sampling) may force fast processors to wait for the slowest processor. In contrast, Async-LDA requires no global synchronization; therefore, each processor can start its next Gibbs sweep without having to wait for slower processors. Async-LDA can perform effectively in a distributed environment with heterogeneous machines having different specifications. However, this is not the case for machines having the same specifications; therefore, we focus on conditions with a homogeneous environment.
- Smola et al. [16] developed a parallel inference method for LDA on a cluster system. It introduces a blackboard-style architecture to facilitate simultaneous communication and sampling between different computers in a cluster environment. They experimented

with a Hadoop cluster, which was being used for production work during their experiments. The parallel computing architecture that they used is different from that in this paper. Moreover, they did not show how the performance is changed when varying the number of CPU cores and/or the number of computers used for computation, while we focus on this kind of performance evaluation in this paper.

- Yi Wang also implemented MPI/OpenMP based parallel implementation of LDA[†], similarly to our work. However, to our knowledge, there are no published reports in which the performance evaluation is carried out using this implementation, while this paper includes some original findings through detailed performance evaluation. Furthermore, this paper includes some more original findings as the result of the additional experiments on the approximation that controls the inter-node communications.
- Masada et al. [11] and Yan et al. [13] used GPGPU for the parallel inference of LDA. Their focus was on the limit of memory size of GPGPU, which is different from our motivation for this research.

3. MPI/OpenMP Hybrid Parallel Inference for LDA

Newman et al.'s approximate distributed inference for LDA (AD-LDA) [6] and its implementations [10] can save considerable memory and time. However, it requires global synchronization at each iteration; therefore, the entire processing time is dominated by the communication overhead with the increase in the number of processors, which limits the increase in inference speed.

We attempt to improve the speed of LDA inference while maintaining inference accuracy. In this section, we briefly review general computing models in parallel and distributed environments and describe our hybrid parallel inference method for LDA, including the approximation that controls the inter-node communications.

3.1 Parallel and Distributed Computing Models

We describe MPI, OpenMP, and MPI/OpenMP hybrid parallelization, which supply low-level services necessary for parallel computing.

3.1.1 MPI

MPI is a popular protocol for programming parallel computing. It uses message passing for communication between processors. MPI is typically suitable for distributed memory architecture, where each processor does not share the memory spaces. MPI can also be used for an SMP cluster; however, MPI implementation usually does not share memory between processors. We use MPI for inter-node message-passing communications in an SMP cluster, but not for intra-node parallelization.

MPI supports an API function "`MPI::COMM_WORLD.Allreduce(sendbuf, recvbuf, count, datatype, op)`", where `sendbuf` is the starting address of the sending buffer, `recvbuf` is the starting address of the receiving buffer, `count` is the number of elements of the sending buffer, `datatype` is the data type of elements of the sending buffer (e.g., `MPI_INT` for the 32-bit integer type), `op` is an operation (e.g., `MPI_SUM` for the summation operation), and `COMM_WORLD` is a default group consisting of all the processors participating in parallel computation. For example, we can use this function as "`MPI::COMM_WORLD.Allreduce(`$\mathbf{C}_{|n}^{word}$, $V \times K$, $\mathbf{C}^{word}$, `MPI_INT, MPI_SUM)`" for computing Eq. (2) that we will introduce in Sect. 3.2.1. Instead of "`COMM_WORLD`", we can also specify a group of processors that are selected to be used for communication.

3.1.2 OpenMP

OpenMP supports shared-memory parallel programming, especially on symmetric multi-processors, where multiple processors share a single memory space. Using OpenMP, loop iterations can be split up into multiple threads. We use it for shared-memory parallelization within each node of an SMP cluster.

OpemMP supports various directives. We used OMP's `parallel` directive to parallelize each Gibbs sweep using multiple threads. We also used `shared(data)` to make all the threads share the `data`. In this case, the multiple threads can read the `data` simultaneously; however, they should not simultaneously write to a specific memory location. We used `atomic` directive to prevent a specific memory location being updated by multiple threads simultaneously.

3.1.3 MPI/OpenMP Hybrid Parallelization

In terms of data communication, shared-memory parallelization is faster than message passing. The fewer processors that take part in message passing communication, the less time it takes. Therefore, the MPI/OpenMP hybrid programming model sometimes substantially increases inference speed on SMP clusters [14]. We discuss how to increase the inference speed of LDA using the hybrid MPI/OpenMP method while maintaining inference accuracy.

3.2 Inter-Node Parallelization

3.2.1 Distributing and Synchronization of Topic-Word Distributions

The inter-node parallel procedure can be used in the manner similar to AD-LDA [6], [10]. The AD-LDA method first divides and distributes D documents over N nodes, assigning $D_n = D/N$ documents to each node n. In other

[†]http://code.google.com/p/ompi-lda/

words, the method partitions word counts $\mathbf{W} = \{\mathbf{w}_d\}_{d=1}^{D}$ into $\{\mathbf{W}_{|1}, \cdots, \mathbf{W}_{|N}\}$ in units of documents, and corresponding topic assignments $\mathbf{Z} = \{\mathbf{z}_d\}_{d=1}^{D}$ into $\{\mathbf{Z}_{|1}, \cdots, \mathbf{Z}_{|N}\}$, where $\mathbf{W}_{|n}$ and $\mathbf{Z}_{|n}$ are allocated only on node n. Here, $\mathbf{w}_d$ and $\mathbf{z}_d$ consist of $w_{d,i}$ and $z_{d,i}$, respectively. Document-specific counts $\mathbf{C}^{doc}$ are likewise divided and distributed over nodes. However, every node maintains its own copies of the overall word-topic counts $\mathbf{C}^{word}$. We denote node-specific counts as $\mathbf{C}_{|n}^{doc}$. Moreover, $\mathbf{C}_{|n}^{word}$ is used to temporarily store word-topic counts accumulated from topic assignments to local documents on each node.

In each Gibbs sweep (each iteration in Gibbs sampling), $\mathbf{Z}_{|n}$ is updated on each node n by sampling every $z_{d,i|n} \in \mathbf{Z}_{|n}$ from the approximate posterior distribution, and each node updates $\mathbf{C}_{|n}^{doc}$ and $\mathbf{C}^{word}$ according to the new topic assignments. After each Gibbs sweep, each node samples and updates word-topic counts of its local documents $\mathbf{C}_{|n}^{word}$ and uses the Allreduce operation to reduce and broadcast the new $\mathbf{C}^{word}$ to all nodes.

$\mathbf{C}_{|n}^{word}$ denotes the result of word-topic count matrices estimated from the data assigned to a node n. In particular, $\sum_n \sum_{v,k} C_{k,v|n}^{word} = W$, where W is the total number of words across all nodes. After node n samples and updates topic assignments $\mathbf{Z}_{|n}$, we modify both $\mathbf{C}_{|n}^{doc}$ and $\mathbf{C}_{|n}^{word}$. To merge the per-node counts, we perform a global update using a Reduce-Scatter operation:

$$\mathbf{C}^{word} \leftarrow \mathbf{C}^{word} + \sum_n (\mathbf{C}_{|n}^{word} - \mathbf{C}^{word}), \qquad \mathbf{C}_{|n}^{word} \leftarrow \mathbf{C}^{word} \tag{2}$$

3.2.2 Synchronization of Document-Topic Distributions

Following [6], document data are distributed over multiple nodes for increasing the speed of LDA inference. When document data are divided in units of documents, as in [6], we need to perform inter-node communication and synchronization only for $\mathbf{C}_{|n}^{word}$ in each Gibbs sweep using Eq. (2), and $\mathbf{C}_{|n}^{doc}$ is not necessary for the transfer across nodes or processors. Therefore, we can reduce the cost of communication.

On the other hand, when document data are divided into units of words, the number of words on each node is roughly equal; therefore, we can reduce the cost of synchronization of count matrices. However, we need to transfer $\mathbf{C}_{|n}^{doc}$ (or a part of it) across nodes. In this case, we have to perform inter-node communication and synchronization for both $\mathbf{C}_{|n}^{word}$ and $\mathbf{C}_{|n}^{doc}$ using Eqs. (2) and (3), respectively.

$$\mathbf{C}^{doc} \leftarrow \mathbf{C}^{doc} + \sum_p (\mathbf{C}_{|n}^{doc} - \mathbf{C}^{doc}), \qquad \mathbf{C}_{|n}^{doc} \leftarrow \mathbf{C}^{doc} \tag{3}$$

However, the cost for transferring document data $\mathbf{C}^{doc}$ may cancel out the cost reduction of the synchronization of count matrices; therefore, we use the method of distributing document data in units of documents.

3.3 Intra-Node Parallelization

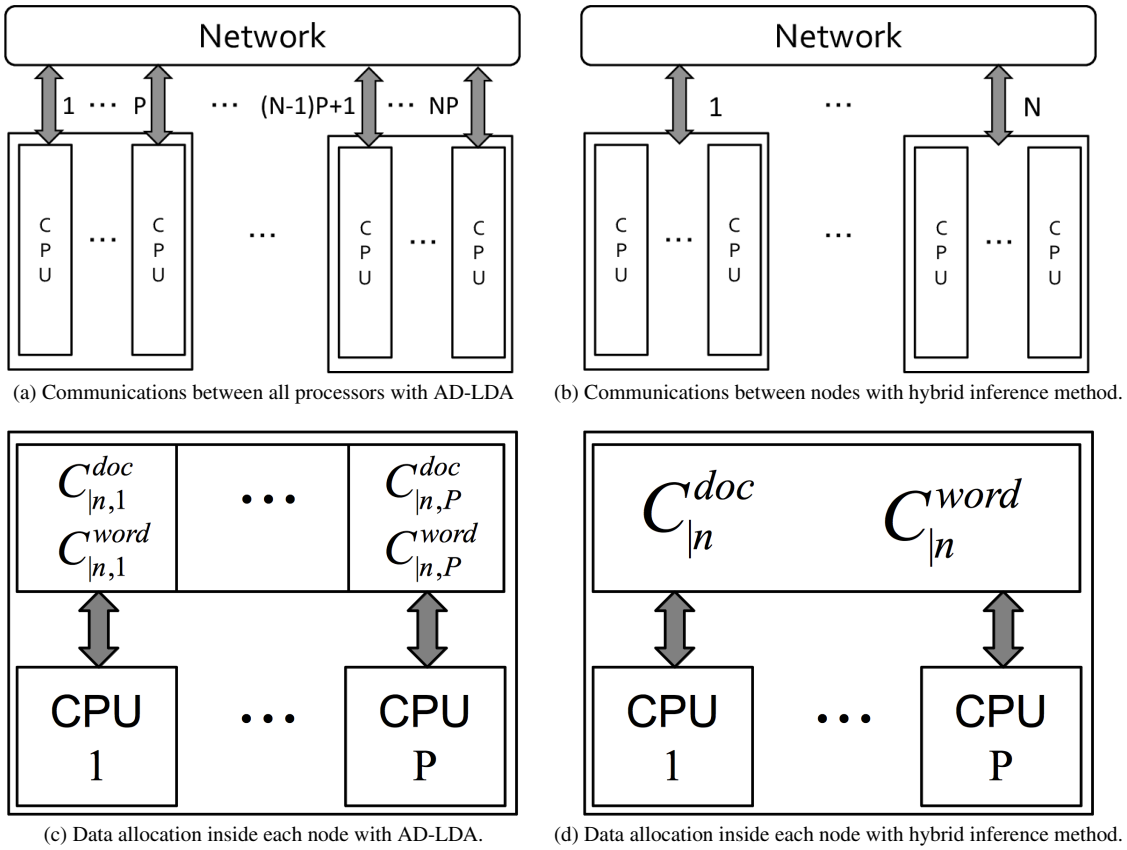
D_n documents allocated to node n are distributed over P processors inside the node, and $D_{np} = D_n/P$ documents are handled by each processor. We partition $\mathbf{W}_{|n}$ into $\{\mathbf{W}_{|n,1}, \cdots, \mathbf{W}_{|n,P}\}$ over each processor p within each node n; however, $\mathbf{Z}_{|n}$ is shared among all the processors within the node. Moreover, each processor handles the original $\mathbf{C}_{|n}^{word}$, not its copy. Each processor p samples topic assignment $z_{d,i|n} \in \mathbf{Z}_{|n}$ to update $\mathbf{Z}_{|n}$ and updates $\mathbf{C}_{|n}^{doc}$ and $\mathbf{C}_{|n}^{word}$ according to the new topic assignment.

As mentioned above, the intra-node parallel method using multiple processors is not an approximate inference method in the sense that we do not use copies of word-topic counts. However, the inference result is not the same as that of serial inference using a single processor since the intra-node parallel method divides the document data into $D_{np} = D_n/P$; thus, the sampling order is different from that of the serial method.

Using OpenMP, multiple threads can simultaneously write to and read from a memory location. However, if two (or more) threads write to a memory location to update $C_{|n}^{doc}$ and $C_{|n}^{word}$ at the same time, the result may be indeterminate. We used OpenMP's `atomic` directive to prevent a specific memory location being updated by multiple threads simultaneously, similar to "Exact Parallelization over N words" of [12].

Figure 2 (a) illustrates the communication pattern of AD-LDA, where the label of each vertical arrow indicates the number of each communication channel. When the AD-LDA method is conducted independently using all processors within and across nodes, the sequence number of processors that take part in inter-node communications is NP, where N and P denote the number of nodes and the number of processors of each node, respectively. To increase the inference speed, we parallelize the inference procedure inside each node using OpenMP. As shown in Fig. 2 (b), the number of processors that take part in inter-node communications is N in the MPI/OpenMP hybrid inference method for LDA, even if all the processors of all nodes are used. Algorithm 1 shows the procedure of n-th node in the hybrid inference method described above. The sixth line indicates that the block in curly brackets is computed by P processors. The eighth line means that all the processors share $z_{d,i}$, $C_{|n}^{word}$, and $C_{|n}^{doc}$. The expression "atomic" in the 10th, 12th, and 14th lines prohibit the processors from simultaneous access to the shared data, such as $z_{d,i}$.

This hybrid inference method uses copies of word-topic counts on each node, while AD-LDA independently using all processors across nodes requires copies of word-topic counts for each processor. The hybrid inference method can save considerable memory when applied to a large collection of documents, as shown in Figs. 2 (c) and 2 (d).



(a) Communications between all processors with AD-LDA

(b) Communications between nodes with hybrid inference method.

(c) Data allocation inside each node with AD-LDA.

(d) Data allocation inside each node with hybrid inference method.

Fig. 2 Inter-processor/node communications and intra-node data allocation in the cases of AD-LDA and hybrid inference method.

3.4 Selective Communication and Synchronization

To achieve further speed-up, we discuss an approximate synchronization in the MPI/OpenMP hybrid inference approach that we described in the previous subsections. In the hybrid inference, each node communicate with each other, via the Allreduce operation, at every sweep of Gibbs sampling in order to synchronize global word-topic counts. As an approximation, we performed the communications between the nodes where a large fraction of updates occur in $Z_{|n}$ $(n = 1, \cdots, N)$ at each Gibbs sweep.

To select the nodes to communicate, we use a binary vector $\mathbf{s} = \{s_n\}$ $(n = 1, \cdots, N)$, where N indicates the number of nodes. Before every Gibbs sweep, all the compoments of $\mathbf{s}$ are set to be zero. After every Gibbs sweep, n-th componet is set to be $s_n = 1$ if the update rate of $\mathbf{Z}_{|n}$ is larger than a threshold, or $s_n = 0$ otherwise. Here, the update rate of $\mathbf{Z}_{|n}$ is given by $\frac{\text{\# of updated elements in } \mathbf{Z}_{|n}}{\text{\# of elements in } \mathbf{Z}_{|n}}$. Then, all nodes communicate using the Allreduce operation to synchronize $\mathbf{s}$ via the Allreduce operation and make a group of the nodes where $s_n = 1$. At the inter-node synchronization step, the communication to synchronize $\mathbf{C}^{word}$ is performed between only the nodes within this group. Algorithm 2 shows the flow of the hybrid inference with the approximate synchronization, where $s\_send[N]$ and $s\_recv[N]$ indicate the binary vector $\mathbf{s} = \{s_n\}$ $(n = 1, \cdots, N)$ for sending and receiving, respectively, in MPI communications.

This approximation can be applied to the MPI-based AD-LDA, by performing the communications between the individual processors where a large fraction of updates occur in $Z_{|m}$ $(m = 1, \cdots, NP)$ at each Gibbs sweep.

4. Experiments

To evaluate our MPI/OpenMP hybrid parallel inference method for LDA, we measured inference speed and test-set perplexity. We randomly divided the words that appear in each document into 10% for testing and 90% for training, such as in [18]. We estimated each document's topic multinomial parameters $\theta_{d,k}$ and each topic's word multinomial parameters $\phi_{k,w}$ using the training set, and computed test-set perplexity with the estimated $\theta_{d,k}$ and $\phi_{k,w}$. In Sect. 4.1, we discuss performance evaluation of the simple MPI/OpenMP hybrid inference method for the LDA model, which we described in Sects. 3.2 and 3.3. In Sect. 4.2, we further investigate the approximation that controls the inter-node communications, which we described in Sect. 3.4.

Algorithm 1 MPI/OpenMP hybrid parallel inference (with the full synchronization)

for each word $w_{d,i} \in W_{|n}$ **do**
  assign k to $z_{d,i}$ according to the uniform distribution $Uniform(1, \cdots, K)$
end for
MPI::COMM_WORLD.Allreduce($C^{word}_{|n}, C^{word}, V \times K$, MPI_INT, MPI_SUM)
for each iteration **do**
  `# pragma omp parallel num_threads(P)`
  {
  `# pragma omp for shared` ($z_{d,i}$, $C^{word}_{|n}$, $C^{doc}_{|n}$)
  for each word $w_{d,i} \in W_{|n}$ **do**
    `# pragma omp atomic`
    $C^{word}_{|n}--$, $C^{doc}_{|n}--$
    `# pragma omp atomic`
    sampling $z_{d,i}$ given $C^{word}_{|n}$, $C^{doc}_{|n}$
    `# pragma omp atomic`
    $C^{word}_{|n}++$, $C^{doc}_{|n}++$
  end for
  }
  MPI::COMM_WORLD.Allreduce($C^{word}_{|n}, C^{word}, V \times K$, MPI_INT, MPI_SUM)
end for

Algorithm 2 MPI/OpenMP hybrid parallel inference with the approximate synchronization

for each word $w_{d,i} \in W_{|n}$ **do**
  assign k to $z_{d,i}$ according to the uniform distribution $Uniform(1, \cdots, K)$
end for
MPI::COMM_WORLD.Allreduce($C^{word}_{|n}, C^{word}, V \times K$, MPI_INT, MPI_SUM)
for each iteration **do**
  $update\_count = 0$
  $s\_send[N] = \{0, \cdots, 0\}$
  $s\_recv[N] = \{0, \cdots, 0\}$
  `# pragma omp parallel num_threads(P)`
  {
  `# pragma omp for shared` ($z_{d,i}$, $C^{word}_{|n}$, $C^{doc}_{|n}$)
  for each word $w_{d,i} \in W_{|n}$ **do**
    `# pragma omp atomic`
    $C^{word}_{|n}--$, $C^{doc}_{|n}--$
    `# pragma omp atomic`
    sampling $z_{d,i}$ given $C^{word}_{|n}$, $C^{doc}_{|n}$
    if $z_{d,i} \neq z^{old}_{d,i}$ **then**
      $update\_count$++
    end if
    `# pragma omp atomic`
    $C^{word}_{|n}++$, $C^{doc}_{|n}++$
  end for
  }
  if $update\_count/(number\ of\ elements\ in\ W_{|n}) > threshold$ **then**
    $s\_send[m] = 1$
  else
    $s\_send[m] = 0$
  end if
  MPI::COMM_WORLD.Allreduce(s_send, s_recv, N, MPI_INT, MPI_SUM)
  make a new group `new_group` consisting of n-th node where $s\_send[n] = 1$
  MPI::new_group.Allreduce($C^{word}_{|n}, C^{word}, V \times K$, MPI_INT, MPI_SUM)
end for

4.1 Evaluation for Hybrid Inference

4.1.1 Settings

We used the hybrid inference method described in Sects. 3.2 and 3.3 with increasing degrees of parallelization, and compared it with two conventional inference methods: MPI-based AD-LDA [6] and the serial inference via the collapsed Gibbs sampling for LDA [8]. We used a 20-node SMP cluster comprising 8 processor cores for each node, although 2 nodes were unable to use in a part of experiments as mentioned below. We evaluated the hybrid inference method (1) increasing the number of nodes $N \in \{2, 5, 10, 15, 20\}$ and using 8 processor cores on each node (2) using 18 nodes and increasing the number of processor cores for inference $P \in \{1, 2, 4, 6\}$ for each node — we used 18 nodes in the experiment (2), due to powering down of two nodes. When $P = 1$ and $N = 18$, AD-LDA parallelizes over 18 nodes comprising one processor core for each node; therefore, AD-LDA is equivalent to the hybrid inference method in this case. For MPI implementation, we used MPICH2 in both implementations of AD-LDA and the hybrid inference method.

We set the Dirichlet hyperparameters to $\alpha = 50/K$, where K denotes the number of topics, and $\beta = 0.01$ [8], [9]. We ran the collapsed Gibbs sampling (both serial and parallel versions) for 500 iterations in all the experiments in this paper.

4.1.2 Data Sets and Evaluation Metrics

We used The New York Times (NYTimes) news article data and PubMed data[†]. These data sets are summarized in Table 2, where D is the number of documents, V is the vocabulary size, and W is the approximate total number of words.

Table 2 Summary of datasets.

dataset	D	V	W (approx.)
NYTimes	300,000	102,660	100,000,000
PubMed	8,200,000	141,043	730,000,000

We measured inference speed and test-set perplexity. The test-set perplexity is given by:

$$perplexity(\mathbf{W}^{test}) = \exp\left(-\frac{\sum_d \log P(\mathbf{w}_d^{test})}{\sum_d |\mathbf{w}_d^{test}|}\right), \quad (4)$$

where

$$P(\mathbf{w}_d^{test}) = \prod_i \sum_k \theta_{d,k}\phi_{k,w_{d,i}^{test}}, \quad (5)$$

$$\theta_{d,k} = \frac{C^{doc}_{d,k} + \alpha}{\sum_{k'} C^{doc}_{d,k'} + K\alpha}, \quad (6)$$

[†]http://archive.ics.uci.edu/ml/datasets/Bag+of+Words

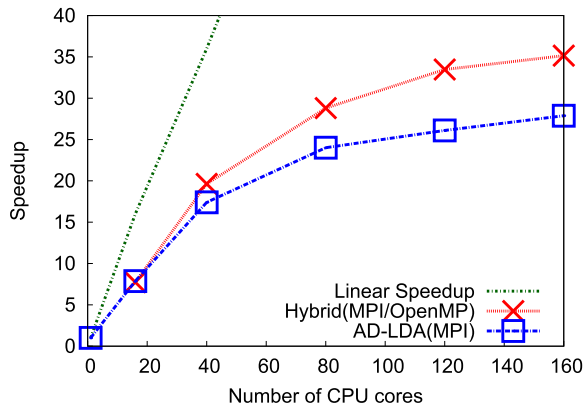
$$\phi_{k,v} = \frac{C^{word}_{k,v} + \beta}{\sum_{v'} C^{word}_{k,v'} + V\beta} \quad . \tag{7}$$

Here, $|\mathbf{w}^{test}_d|$ in Eq. (4) indicates the number of test-set words in document d.

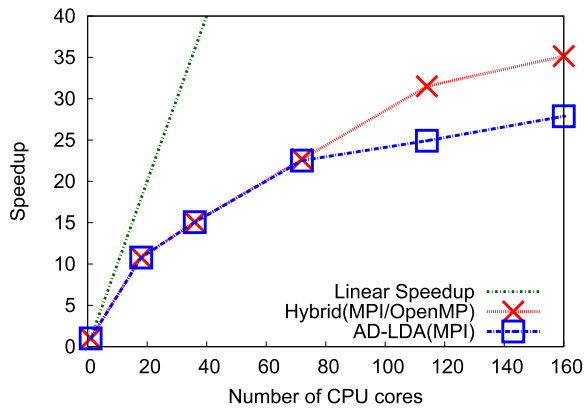
4.1.3 Results

Figure 3 (a) compares the increase in inference speed of AD-LDA and the MPI/OpenMP hybrid inference method. To evaluate this increase in speed, we used the conventional serial inference method of LDA as the baseline. the hybrid inference method increased inference speed as the degree of parallelization increased, even when the parallelization degree was 160.

For reference, Fig. 3 (b) compares the two inference methods when we fixed the number of CPU cores on each node used for inference. The hybrid inference method still performed better than AD-LDA. The more parallelization within each node, the faster the hybrid inference method performed. Not surprisingly, the overall performance of the hybrid method in this case was not as good as when we fixed the number of CPU cores on each node used for inference, as shown in Fig. 3 (a). This is because the number of processor cores that are involved in inter-node communications with the hybrid inference method is not very different from that of AD-LDA, particularly in low level parallelization.



(a) Increase in inference speed on NYTimes dataset for number of processor cores (when we used eight CPU cores on each node and increased number of nodes).



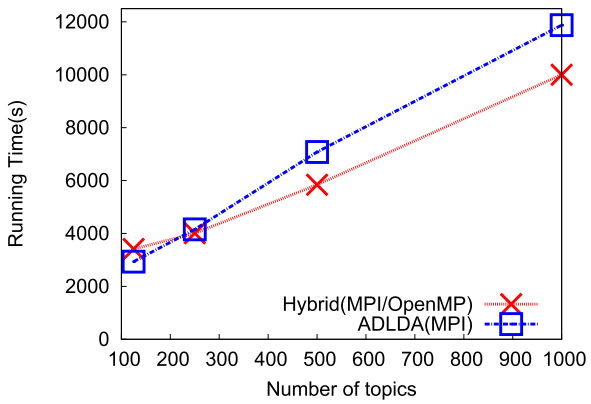
(b) Increase in inference speed on NYTimes dataset for number of processor cores (when we used 18 nodes and increased number of CPU cores we used on each node).

Fig. 3 Increase in inference speed on NYTimes dataset for number of processor cores.

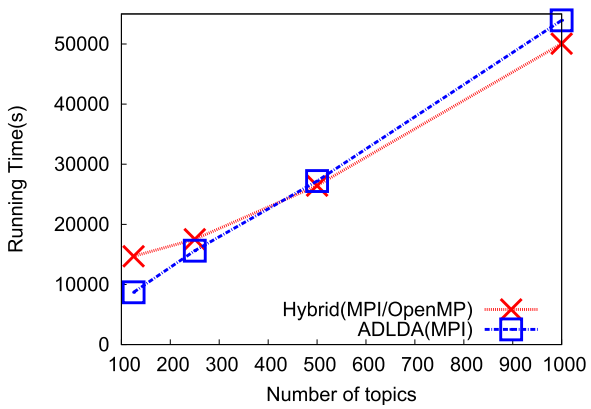
We present further experimental results in the following paragraphs. Tables 3 (a) and 3 (b) list the detailed results when we fixed the number of CPU cores on each node used for inference. We can see in the tables that the hybrid inference method reduced communication cost, compared with AD-LDA. The hybrid inference method increased the inference speed as the degree of parallelization increased; however, AD-LDA slowed the inference speed in high parallelization. This performance gap is caused by the fact that the hybrid inference method can minimize the communication cost by combining inter-node communications via message passing and intra-node communications via loop directives, compared to AD-LDA where all inter-processor communications are implemented via message passing.

For reference, Tables 3 (c) and 3 (d) list the detailed results when we changed the number of CPU cores on each node with a fixed number of nodes. The hybrid inference method maintained constant communication time, as shown in Table 3 (d).

Figures 4 (a) and 4 (b) show the running time for the number of topics when we used 18 nodes and eight CPU



(a) Running time for number of topics (NYTimes).



(b) Running time for number of topics (PubMed).

Fig. 4 Running time for number of topics.

Table 3 Increases in inference speed.

(a) Increase in inference speed of AD-LDA on NYTimes dataset (when we used eight CPU cores on each node and increased number of nodes).

number of cores	computation time (sec)	communication time (sec)	synchronization time (sec)	total time (sec)	speedup ratio
1	217798	0.00	0.00	217798	1.00
16	26571	856	240	27667	7.872
40	9910	2247	354	12511	17.41
80	6126	2681	265	9072	24.01
120	4917	3178	249	8344	26.10
160	4287	3298	227	7812	27.87

(b) Increase in inference speed of hybrid inference on NYTimes dataset (when we used eight CPU cores on each node and increased number of nodes).

number of cores	computation time (sec)	communication time (sec)	synchronization time (sec)	total time (sec)	speedup ratio
1	217798	0.00	0.00	217798	1.00
16	27662	301	168	28131	7.742
40	10304	607	181	11092	19.63
80	6441	1007	113	7561	28.80
120	4751	1620	135	6506	33.48
160	3419	2333	90	5842	37.28

(c) Increase in inference speed of AD-LDA on NYTimes dataset (when we used 18 nodes and increased number of CPU cores used on each node).

number of cores	computation time (sec)	communication time (sec)	synchronization time (sec)	total time (sec)	speedup ratio
1	217798	0.00	0.00	217798	1.00
18	17265	2322	597	20184	10.79
36	11376	2644	439	14459	15.06
72	6391	2892	389	9672	22.52
108	5354	3221	166	8741	24.92
160	4287	3298	227	7812	27.88

(d) Increase in inference speed of hybrid inference on NYTimes dataset (when we used 18 nodes and increased number of CPU cores used on each node).

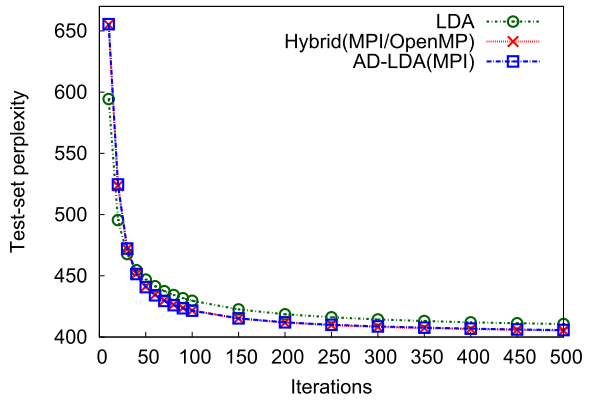
number of cores	computation time (sec)	communication time (sec)	synchronization time (sec)	total time (sec)	speedup ratio
1	217798	0.00	0.00	217798	1.00
18	17265	2322	597	20184	10.79
36	11615	2408	433	14456	19.63
72	6775	2480	349	9604	28.80
108	4224	2495	194	6913	33.48
160	3419	2333	90	5842	37.28

cores on each node. When the number of topics was small, the hybrid inference method was slower than AD-LDA because the communication cost is a small percentage of the total running time and the collision of topic assignments is frequent. However, the more topics, the better the hybrid method performs. By comparing the results in Figs. 4 (a) and 4 (b), the difference in the two methods is smaller with the PubMed dataset. This is because the number of words in the PubMed dataset is much larger than that of NYTimes, whereas the vocabulary size of these two datasets is not very different.

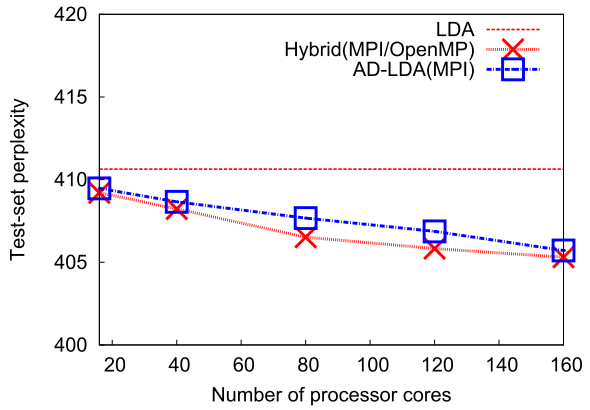
Figures 5 (a) and 5 (b) show test-set perplexity for the serial inference, AD-LDA, and the hybrid inference method, where Fig. 5 (a) shows test-set perplexity over 500 iterations, while Fig. 5 (b) shows test-set perplexity at the 500th iteration point for number of processor cores. The test-set perplexities were comparable; however, the perplexity of AD-LDA and the hybrid inference method was slightly better than that of the serial inference. Figure 5 (b) shows the test-set perplexity at the 500th iteration, varying the number of processor cores used. The test-set perplexity changed only slightly even when the number of processor cores was increased. The test-set perplexity with the hybrid inference method was even better than that of the standard LDA in all cases.

4.2 Evaluation for Approximate Synchronizaton

To demonstrate the effect of the approximate synchronization described in Sect. 3.4, we ran the collapsed Gibbs sampling with the MPI/OpenMP hybrid approach, changing the update-rate threshold of topic assignments in all the documents allocated to each node. We also applied the approximate synchronization to MPI-based AD-LDA, for compari-



(a) Test-set perplexity over 500 iterations on NYTimes dataset (when we fixed number of CPU cores on each node used for inference).



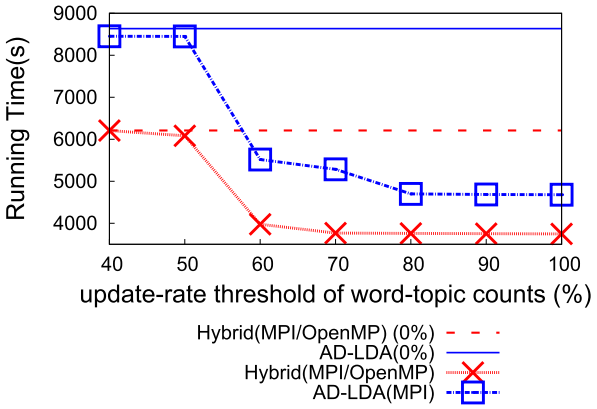
(b) Test-set perplexity at the 500th iteration for number of processor cores on NYTimes dataset.

Fig. 5 Test-set perplexity on NYTimes dataset.

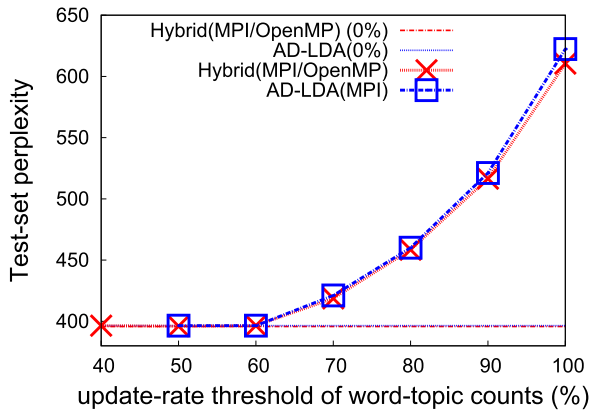
son.

For the experiments, we used 18 nodes comprising 8 processor cores for each node, in the SMP cluster used in the experiments described in the previous subsection. We ran the collapsed Gibbs sampling for 500 iterations, assuming the Dirichlet hyperparameters to be $\alpha = 50/K$ and $\beta = 0.01$, as in Sect. 4.1.1.

Figure 6 (a) shows the total running time for the collapsed Gibbs sampling at the 500th iteration point. Here, the horizontal axis indicates the update-rate threshold of topic assignments. As shown in this figure, the larger the threshold is, the less nodes or processors communicate with each other. Therefore, the total running time can be reduced as the threshold gets larger. Figure 6 (b) demonstrates test-set perplexity for different update-rate threshold of topic assignments. When the threshold is too large, the global word-topic counts C^{word} are rarely synchronized, resulting in a larger test-set perplexity that indicates a lower inference accuracy. Considering these results, when we set the threshold to be 60% in the experiments, we observed that the approximate synchronization increased inference speed while maintaining inference accuracy, compared with either of the hybrid inference method without the approximate synchronization or MPI-based AD-LDA.



(a) Running time at the 500th iteration point for different update-rate threshold of word-topic counts on NYTimes dataset. Two horizontal lines indicate the cases when all nodes (or processors) communicate at each iteration.



(b) Test-set perplexity at the 500th iteration point for different update-rate threshold of word-topic counts on NYTimes dataset. Two horizontal lines indicate the cases when all nodes (or processors) communicate at each iteration.

Fig. 6 Running time and test-set perplexity for different update-rate threshold of word-topic counts on NYTimes dataset.

5. Conclusions

We investigated parallel distributed methods for the sake of efficient collapsed Gibbs sampling inference for LDA, a well-accepted probabilistic topic model that incurs high computational cost for the inference with large-scale data. We developed an MPI/OpenMP hybrid parallel inference method for LDA, assuming an SMP cluster, which is widely used. We demonstrated through experiments with various settings that the MPI/OpenMP hybrid parallel method can substantially improve the inference speed by reducing the communication cost while maintaining inference accuracy, compared with using only an MPI or a single processor. We then carried out experiments on the hybrid parallel method with approximate synchronization, and demonstrated that it can further increase inference speed while keeping inference accuracy, compared with using the hybrid parallel method without the approximate synchronization.

Experiments with more complex topic models are in

progress. Algorithmic improvements [15] and pipeline processing or other scheduling techniques [16], [17] are for future work.

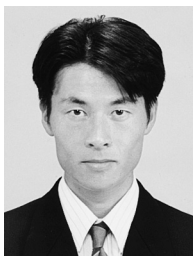
Acknowledgements

We thank Tomio Kamada, Eric P. Xing, and the anonymous reviewers for valuable discussions and comments. This work was supported in part by the Grant-in-Aid for Scientific Research (#23300039) from JSPS, Japan.

References

[1] D.M. Blei, A.Y. Ng, and M.I. Jordan, "Latent Dirichlet allocation," J. Machine Learning Research, vol.3, pp.993–1022, 2003.
[2] D.M. Blei and M.I. Jordan, "Modeling annotated data," Proc. 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval, Toronto, Canada, pp.127–134, 2003.
[3] F.F. Li and P. Perona, "A Bayesian hierarchical model for learning natural scene categories," Proc. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Diego, California, USA, pp.524–531, 2005.
[4] H. Zhang, C.L. Giles, H.C. Foley, and J. Yen, "Probabilistic community discovery using hierarchical latent Gaussian mixture model," Proc. 22nd national conference on Artificial intelligence, Vancouver, Canada, pp.663–668, 2007.
[5] E.M. Airoldi, D.M. Blei, S.E. Fienberg, and E.P. Xing, "Mixed membership stochastic block models," J. Machine Learning Research, vol.9, pp.1981–2014, 2008.
[6] D. Newman, A. Asuncion, P. Smyth, and M. Welling, "Distributed inference for latent Dirichlet allocation," Advances in Neural Information Processing Systems 20, pp.1081–1088, MIT Press, 2007.
[7] A. Asuncion, P. Smyth, and M. Welling, "Asynchronous distributed learning of topic models," Advances in Neural Information Processing Systems 21, pp.81–88, MIT Press, 2008.
[8] T.L. Griffiths and M. Steyvers, "Finding scientific topics," Proceedings of the National Academy of Sciences of the United States of America, vol.101, pp.5228–5235, 2004.
[9] M. Steyvers and T.L. Griffiths, "Probabilistic topic models," Handbook of Latent Semantic Analysis, 2007.
[10] Y. Wang, H. Bai, M. Stanton, W.Y. Chen, and E.Y. Chang, "PLDA: Parallel latent Dirichlet allocation for large-scale applications," Algorithmic Aspects in Information and Management, Lecture Notes in Computer Science, vol.5564, pp.301–314, Springer, 2009.
[11] T. Masada, T. Hamada, Y. Shibata, and K. Oguri, "Accelerating collapsed variational Bayesian inference for latent Dirichlet allocation with Nvidia Cuda compatible devices," Next-Generation Applied Intelligence, Lecture Notes in Computer Science, vol.5579, pp.491–500, Springer, 2009.
[12] D. Newman, P. Smyth, and M. Steyvers, "Scalable Parallel Topic Models," Journal of Intelligence Community Research and Development, 2006.
[13] F. Yan, N. Xu, and Y. Qi, "Parallel inference for latent Dirichlet allocation on graphics processing units," Advances in Neural Information Processing Systems 22, pp.2134–2142, MIT Press, 2009.
[14] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes," Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, Weimar, Germany, pp.427–436, 2009.
[15] L. Yao, D. Mimno, and A. McCallum, "Efficient methods for topic model inference on streaming document collections," Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, pp.937–946, 2009.
[16] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," Proc. VLDB Endowment, vol.3, no.1-2, pp.703–710, 2010.
[17] Z. Liu, Y. Zhang, E.Y. Chang, and M. Sun, "PLDA+: Parallel latent Dirichlet allocation with data placement and pipeline processing," ACM Transactions on Intelligent Systems and Technology, vol.2, no.3, pp.26:1–26:18, 2011.
[18] Y.W. Teh, D. Newman, and M. Welling, "A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation," Advances in Neural Information Processing Systems 19, pp.1353–1360, MIT Press, 2006.

Shotaro Tora received the B.E. degree in Computer Science and Systems Engineering from Kobe University, Japan in 2011. He is currently pursuing a masters degree at the Graduate School of System Informatics, Kobe University, Japan.



Koji Eguchi is an Associate Professor at the Graduate School of System Informatics, Kobe University, Japan. His research interests include information retrieval and statistical machine learning.