Parallel Processing of Burst Detection in Large-Scale Document Streams and Its Performance Evaluation

Kaishi Hirahara, Keiichi Tamura, Hajime Kitakami, and Shingo Tamura

Abstract—**Online documents on the Internet are represented as a document stream because the documents have a temporal order. This has resulted in numerous studies on extracting a frequent phenomenon (involving keywords, users, locations etc.) known as a burst. Recently, with the growth of interest in social media, the number of documents created on the Internet has increased exponentially. Therefore, the speed-up of burst detection in a large-scale document stream is one of the most important challenges. In this paper, we propose a novel parallelization method for the parallel processing of Kleinberg's burst detection algorithm in a large-scale document stream. Specifically, we present a technique to combine the inter-task parallelization model with the intra-task parallelization model. This combination can achieve seamless dynamic load balancing and detect bursts in a large-scale document streams in memory.**

***Index Terms*—document stream; burst detection; parallel processing; dynamic load balancing; text mining;**

I. Introduction

TOPIC detection and tracking in documents on the Internet, as well as those on micro-blogs (e.g., Twitter), online news, and blogs, has been attracting researchers in the text mining domain ever since people began to exchange information through the Internet [1]. In particular, with the growth of interest in social media, documents on the Internet have begun including not only technical but also social topics. Therefore, extracting specific patterns from these documents is one of the most important challenges in social topic analysis, mark analysis, emergency management, and search engine performance improvement.

Documents on the Internet are represented to as a document stream because the documents have a temporal order [2]. Thus, numerous studies have been conducted on the extraction of a frequent phenomenon (involving keywords, users, locations etc.) known as a burst. Burst measurement is one of the simplest ways of detecting a sudden increase in the frequency of a certain phenomenon. Detecting bursts in a document stream assists topic detection and tracking, because increased attention of people on certain events and topics increases the frequencies of terms related to the events and topics in a document stream.

Kleinberg's burst detection algorithm [3] was proposed for detecting bursts in a document stream. Kleinberg defines a bursty term as one that increasingly occurs in a document stream. Some terms are highly bursty in the sense that the frequency of their occurrence rises when a particular event or topic attracts public attention. Kleinberg's burst detection algorithm aims to find certain time periods in which terms occur with a high frequency. When a term related to an attention-attracting event or topic becomes extremely bursty, the interarrival time between documents that include the term becomes smaller. Therefore, the time period when a term becomes extremely bursty can be detected using the interarrival time between the documents.

Recently, with growing worldwide interest in social media, the number of documents created on the Internet has increased exponentially. In this situation, burst detection presents three challenges. First, the computation time for detecting bursts in a document stream is increasing, because there are numerous terms in a large-scale document stream. Second, the size of the time-series data of a term is increasing. Kleinberg's burst detection algorithm needs memory space equal to the product of the number of occurrences of a term and the number of states. Thus, when the document stream is large, the algorithm is unable to perform efficiently using only the main memory of the computer. This causes serious performance degradation. Third, it is difficult for a simple parallelization model to balance loads efficiently because terms occur with widely varying frequencies.

In this paper, we propose a novel parallelization method for the parallel processing of Kleinberg's burst detection algorithm in a large-scale document stream. The main contributions of this study are as follows:

(1) To parallelize Kleinberg's burst detection algorithm, we define two types of parallelization models: the inter- and intra-task parallelization models. The processing of burst detection for one term in a document stream is defined as a task. The burst detection for each term using Kleinberg's burst detection algorithm can be performed independently. Thus, tasks can be performed concurrently. The inter-task parallelization model is defined as one in which tasks are performed simultaneously. Conversely, the time-series data of one term can be divided into several sub-time-series data called partitions. The intra-task parallelization model is defined as one in which burst detection for each partition using Kleinberg's burst detection algorithm is performed simultaneously.

(2) To balance work loads dynamically, we propose combining the inter- and intra-task parallelization models. If

Manuscript received Oct 26, 2012

K.Hirahara, K.Tamura, H.Kitakami and S.Tamura are with the Graduate School of Information Sciences, Hiroshima City University, 3-4-1, Ozuka-Higashi, Asa-Minami-Ku, Hiroshima 731-3194, Japan; corresponding e-mail: (ktamura@hiroshima-cu.ac.jp).

DOI: 10.5176/2251-3043_2.4.206

the frequency of term occurrence in a task is less than a threshold σ, the proposed parallelization model performs the task normally. Otherwise, the task is divided into several sub-tasks comprising partitions that constitute the entire time-series data of the task. Sub-tasks are performed simultaneously. By adjusting the size of the partition, we can perform large tasks in a computer's main memory.

(3) To evaluate the proposed model for parallel processing of Kleinberg's burst detection algorithm, we used an actual large-scale document stream composed of crawling tweets on Twitter. The number of tweets is 1,280,000 and they were collected from June to December in 2009. The experimental results show that the proposed parallelization model addresses the above three challenges faced by burst detection in a large-scale document stream.

The rest of this paper is organized as follows: Section 2 overviews related work. Section 3 defines a burst and describes Kleinberg's burst detection algorithm. Section 4 explains the problem definition and proposes our novel parallelization method. Section 5 presents the experimental results of performance evaluation experiments. Section 6 concludes this paper.

II. Related Work

With the widespread of use of the Internet, many techniques for topic detection and tracking have been proposed [1]. In particular, many studies have been conducted on topic detection and tracking in a document stream. This section overviews related work on burst detection and parallel processing of the Viterbi algorithm and dynamic programming, which is based on Kleinberg's burst detection algorithm.

To track and detect topics in a document stream that have public appeal, burstiness is the simplest but the most effective criterion. A number of studies have been conducted on burst detection algorithms [3], [4] [5], [6], [7], [8], [9], [10], [11]. Of these, Kleinberg's burst detection algorithm [3] has had the most significant impact on many studies. It is based on a queuing theory for bursty network traffic. The shorter the data arrival time interval, the higher is the degree of burst state and vice versa. The algorithm is explained in detail ahead in the paper. It is applicable to various document streams such as e-mails [3], blogs [4], [5], online publications [6], and social tags [11].

Kleinberg's burst detection algorithm is known as one of the most efficient algorithms for burst detection. However, with the rapid growth of social media sites, the number of online documents created on the Internet has been increasing exponentially. Thus, we meet a new challenge: How do we detect bursts in large-scale document streams? Kleinberg's algorithm is based on the Viterbi algorithm. Several studies have attempted to parallelize the Viterbi algorithm on the basis of hardware [12], [13], [14]. Hui et al. [12] proposed a method for parallelizing the Viterbi algorithm using a multi-microprocessor. Yeo et al. [13] performed parallelization on a certain hardware chip, while Wang et al. [14] used a field-programmable gate array (FGPA). These studies focused on real-time processing on hardware. However, our goal is to develop an efficient parallelization method for use on software and not hardware.

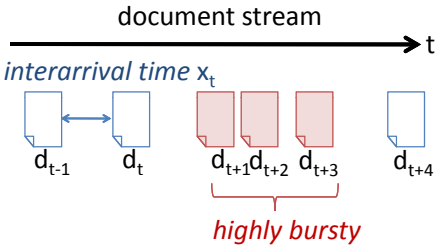


Fig. 1. Document stream, in which documents arrive in a temporal order.

The Viterbi algorithm involves dynamic programming. Many studies have been conducted on the parallel processing of dynamic programming [15], [16], [17], [18], [19], [20]. Recently, several parallelization methods for dynamic load balancing on multi-core processors were proposed [18], [19], [20]. These methods focus on memory or cache use efficiency. However, almost all studies on parallelization for dynamic programming address the processing of only one large task. To the best of our knowledge, little attention has been paid to the case where many dynamic programming tasks are executed. In this study, we propose a novel parallelization method for this case.

III. Burst Detection

In this section, we define a document stream and burst detection, and briefly explain Kleinberg's burst detection algorithm.

A. Document Stream

A document stream, which resembles a data stream, is defined as a sequence of documents that have a temporal order. Fig.1 shows an example of a document stream. In this figure, the documents arrive in a temporal order. The time interval x_t between document d_{t+1} and document d_t is called the interarrival time. Examples of a document stream include, but are not limited to, tweets on Twitter. Tweet i is represented as document d_i. The interarrival time x_i is defined as the time interval between the posting time of tweet $i+1$ and that of tweet i.

B. Burst

The number of documents that include particular terms related to a certain event or topic increases gradually as more and more people become interested in that event or topic and vice versa. Furthermore, as the number of documents that include a term related to a certain event or topic increases in a document stream, the interarrival time between these documents becomes smaller. A term is considered highly bursty during a period in which the interarrival time is shorter than usual.

C. Kleinberg's Burst Detection Algorithm

Kleinberg defined a model with an infinite-state automaton in which bursts are represented as state transitions. Suppose that there are m states in the infinite-state automaton. Each interarrival time is a probabilistic output that depends on the internal states of the infinite-state automaton. In the model, a state is associated with the degree of burstiness: a higher state indicates a higher degree of burstiness, and vise versa.

Let the sequence of interarrival times between document postings be $x = (x_1, x_2, \cdots, x_n)$. The problem is defined as finding the optimal state-transition sequence $s = (s_1, s_2, \cdots, s_n)$ that will minimize the cost function

$$C(s|x) = (\sum_{i=1}^{n-1} \tau(s_i, s_{i+1})) + (\sum_{i=1}^{n} -lnf_{s_i}(x_i)). \quad (1)$$

The function $\tau(s_i, s_{i+1})$ returns a state-transition cost from state i to state j. It is defined as

$$\tau(i,j) = \begin{cases} (j-i)\gamma, \ if \ j > i, \\ 0, otherwise, \end{cases} \quad (2)$$

where $\gamma(> 0)$ is a user-given parameter and n is the number of documents in the concerned document stream. Equation (2) indicates that moving to a higher state incurs a cost which moving to a lower state incurs no cost.

Function $f_k(x_i)$ is the exponential density function for the probability of outputting the interarrival time x_i in state k, and is defined as

$$f_k(x_i) = \lambda_k e^{-\lambda_k x_i}, \quad (3)$$

where λ_k is the arrival rate of documents associated with state k and is defined as

$$\lambda_k = \frac{n}{T}\beta^k, \quad (4)$$

where n is the number of documents, T is the entire time range, and $\beta(> 1.0)$ is a user-given parameter.

The Viterbi algorithm for hidden Markov models, which is a dynamic programming approach, is the most effective solution for determining an optimal state-transition sequence $s = (s_1, s_2, \cdots, s_n)$ to minimize Equation (1). First, we calculate the cost $C_j(i)$:

$$C_j(i) = -lnf_j(x_i) + min_l(C_l(i-1) + \tau(l,j)), \quad (5)$$

where $C_j(i)$ is the minimum cost of a state-transition sequence that ends with state j at the i-th time interval in the document stream. Equation (5) can be calculated using the previous $(i-1)$-th $C_l(i-1)(0 \le l \le m-1)$. Second, we find the minimum cost in $C_j(i)(0 \le j \le m-1)$. Suppose that the minimum cost in $C_j(n)(0 \le j \le m-1)$ is $C_{min}(n)$. Finally, we trace back with $C_{min}(n)$ as the starting point.

IV. Proposed Method

This section gives the problem definition, and presents two types of parallelization models: the inter- and intra parallelization models. Furthermore, we propose a novel parallelization model that combines these two models.

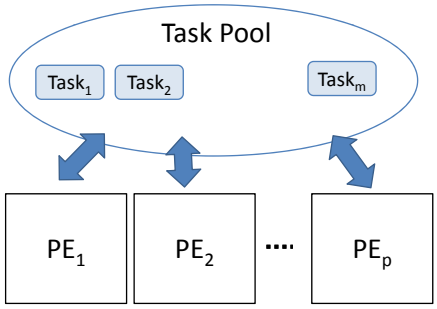


Fig. 2. A parallelization environment containing task pool and several PEs.

A. Problem Definition

Let a document d_i on a document stream $DS = \{d_1, d_2, \cdots, d_n\}$ be $d_i = <altime_i, text_i>$, where $text_i$ is the text data and $alvtime_i$ is the arrival time. In addition, we consider a set of all the terms appearing in text data, $T = \{term_1, term_2, ..., term_m\}$. The number of documents $term_i$ is denoted by $|term_i|$. Here, we define the inter-arrival time sequence of documents that include $term_i$ as

$$TALT_i = (talt_{i,1}, talt_{i,2}, \cdots, talt_{i,|term_i|}), talt_{i,j} \in ALT, \quad (6)$$

where $ALT = \{altime_1, altime_2, \cdots, altime_n\}$ is a set of all arrival times for all the documents.

The goal of this study is to parallelize the processing of burst detection that extracts all the state-transition sequences of all the m terms using Kleinberg's burst detection algorithm.

B. Parallelization Model

In this subsection, we describe the inter- and intra-task parallelization models. There are many parallelization environments for parallel processing (e.g., PC clusters, multi-core CPUs, SMPs, and GPUs). In this study, we focus only on environments in which the task pool model is executable (Fig.2). In the task pool model contains a task pool and a Processor Element (PE). The task pool stores tasks and a PE is a processing unit such as a CPU and a CPU-core in a multi-core CPU. Each PE gets a task out of the task pool and performs the task while the task pool is not empty.

1) Inter-Task Parallelization Model: The processing of burst detection for one term in a document stream is defined as a task. Thus, if there are m terms in a document stream, there are m tasks. In the inter-task parallelization model, tasks are performed simultaneously, because each task can be performed individually (Fig.3).

The procedure steps of the inter-task parallelization model are as follows:

(1) For each interarrival time sequence, $TALT_i$ is put into the task pool as task i.
(2) Each PE gets a task from the task pool. Suppose that the task is the k-th term. The PE obtains the state-transition sequence s, which is an output of performing Kleinberg's burst detection algorithm on the inter-arrival time sequence $TALT_k$. A pair of the term $term_k$ and the state-transition sequence s is put into the result pool.

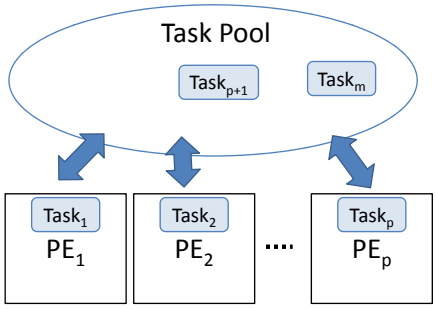


Fig. 3. Inter-task parallelization model.

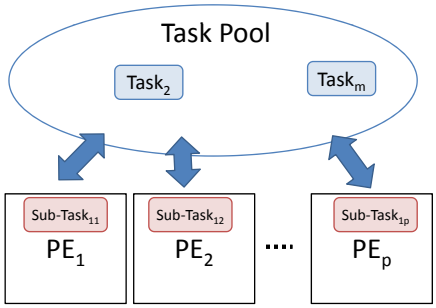


Fig. 4. Intra-task parallelization model.

(3) When a PE finishes a task, it gets its next task out of the task pool and performs the task while the task pool is not empty.

2) Intra-Task Parallelization: Each interarrival time sequence is divided into several sub-sequences called partitions. The intra-task parallelization model is defined as parallel processing in which burst detection is simultaneously for each partition using Kleinberg's burst detection algorithm (Fig.4).

First, the inter-arrival sequence $TALT_i$ is divided into p partitions. Let a partition $PTALT_i^l$ be

$$PTALT_i^l = (talt_{i,d\times l}, talt_{i,d\times l+1}, \cdots, talt_{i,d\times l+d-1}), \\ talt_{i,j} \in ALT, 1 \leq l \leq p,$$

where

$$d = \frac{|TALT_i|}{p}. \tag{7}$$

For each $PTALT_i$, the intra-task parallelization model obtains the sub-state-transition sequence, s^k, by performing Kleinberg's burst detection algorithm on $PTALT_i$, as follows:

$$s = s^1 \cup s^2 \cdots \cup s^p \tag{8}$$

Following are the procedure steps of the intra-task parallelization model:

(1) For each inter-arrival time sequence, $TALT_i$ is put into the task pool as task i.

(2) A task is obtained from the task pool. Suppose that the task is the k-th term's task. We divide $TALT_k$ into p (p is the number of PEs) partitions. The partition $PTALT_k^l$ is assigned to the l-th PE. Each PE that is assigned a partition performs Kleinberg's burst detection algorithm on that partition. The state-transition sequence s constitutes all the sub-state-transition sequences s_i after all the sub-tasks are finished. A pair of the term $term_k$ and the state-transition sequence s is put into the result pool.

(3) We get the next task out of the task pool and perform the task according to Steps (1) and (2) while the task pool is not empty.

C. Inter-Task with Intra-Task Parallelism

The inter-task parallelization model is the simplest parallelization model; however, there are two inherent issues: (1) It is difficult for this model to balance loads efficiently because the frequency of term occurrences differs very widely; (2) Kleinberg's algorithm cannot perform on memory in a large-scale document stream because it needs memory space equal to the product of the number of term occurrences and the number of states.

To address these two issues, we combine the inter-task parallelization model with the intra-task parallelization model. In our parallelization model, each PE performs a task individually in the same manner as in the inter-task parallelization model; however, in our model, a PE divides the task into sub-tasks if $|TALT_i| \geq \sigma$. The task is divided into $div = |TALT_i|/range$ partitions. A sub-task is one that finds the optimal state-transition sequence on a partition. We call a regular task an "ordinary-task" and call this sub-task a "divided-task."

Below are the procedure steps of the proposed parallelization model are:

(1) For each inter-arrival time sequence, $TALT_i$ is put into the task pool as ordinary-task i.

(2) Each PE gets a task from the task pool.

a) If the task is an ordinary-task and $|TALT_k| < \sigma$, the PE gets the state-transition sequence s, which is the output of performing the Kleinberg's burst detection algorithm for the inter-arrival time sequence $TALT_k$. A pair of the term $term_k$ and the state-transition sequence s is put into the result pool.

b) If the task is an ordinary-task, but $|TALT_k| \geq \sigma$, $TALT_k$ is divided into div ($div = (|TALT_i|)/range$) partitions. For each partition, the partition $PTALT_k^l$ is put into the task pool as a divided-task.

c) If the task is a divided-task, the PE performs Kleinberg's burst detection algorithm on it.

(3) When a PE has finished the task, it obtains the next task out of the task pool and performs the task according to Step (1) and Step (2), while the task pool is not empty.

V. Performance Evaluation

To evaluate the proposed parallelization model, we performed three experiments. This section presents the experimental results.

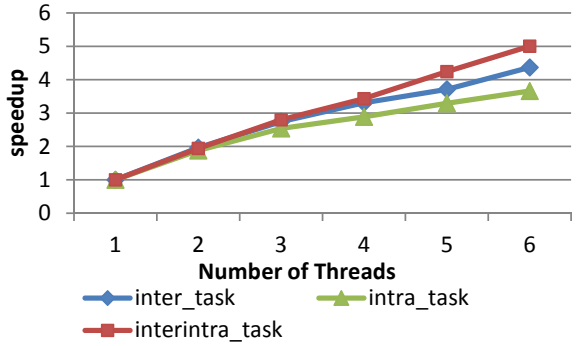


Fig. 5. Speed-up ratio (Task-100).

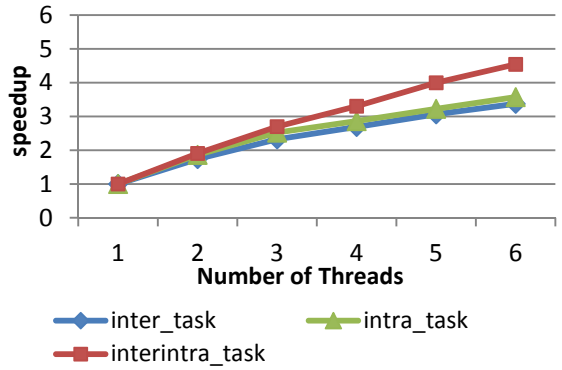


Fig. 6. Speed-up ratio (Task-1000).

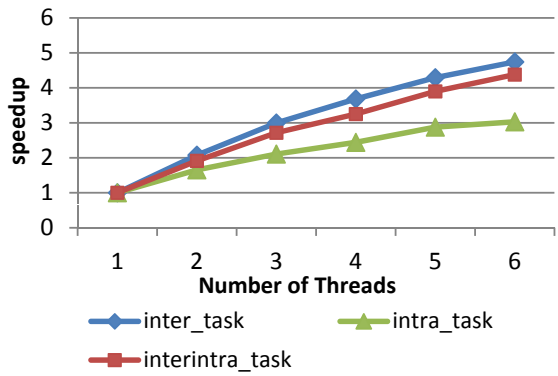


Fig. 7. Speed-up ratio (Task-10000).

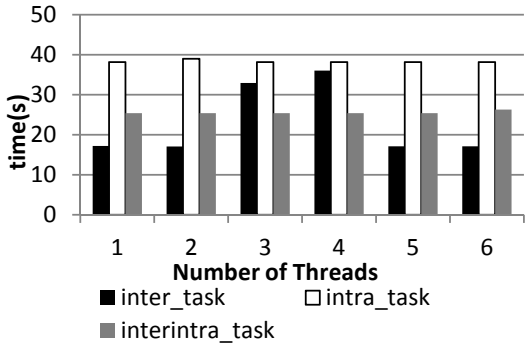


Fig. 8. Processing time (Task-1000).

A. Experimental Setup

We implemented the proposed parallelization model on a multi-core CPU using the master worker model with multi threads. In our implementation, therefore, a thread on a CPU core represents a PE. We used a PC that having a multi-core CPU (CPU: Phenom II X6 1090T Six-Core/3.2G/L3, Memory: 4GB RAM).

Our performance evaluation constituted three experiments. Experiment 1 was performed to evaluate the speed-up ratios of the inter-task, intra-task, and the proposed parallelization model. Experiment 2 showed the error in the results of intra-task parallelization. Experiment 3 compared the inter-task parallelization model with the proposed one in terms of the perspective of adaptive ability.

In the experiments, we used an actual large-scale document stream composed of crawling tweets on Twitter. The number of tweets was 1,280,000; they were collected from June to December in 2009. We extracted 10,000 terms from the documents. We created three types of tasks: task-100, task-1000, and task-10000, which consisted of 100, 1,000, and 10,000 tasks and detected the bursts of 100, 1,000, 10,000 terms, respectively.

B. Experiment 1

In Experiment 1, we compared the speed-up ratios of the inter-task, the intra-task, and the proposed parallelization model. Figs.5, 6, and 7 show the results of task-100, task-1000, and task-10000, respectively. The vertical axis represents the speed-up ratios, while the horizontal axis shows the number of threads. In these figures, "inter_task," "inter_task," and "interintra_task" denote the inter-task parallelization model, the intra-task parallelization model, and the proposed parallelization model. The length of the divided range in the interintra-task parallelization model is 1,000. The gamma value is 1.1, and the beta value is 0.05.

There is no difference between the performance of the inter-task and the proposed parallelization model for task-100, because load unbalancing does not occur in either of the cases. However, the intra-task parallelization model performs poorly in comparison with both these models. The intra-task parallelization model is necessary to synchronize each other thread. In the case of task-1000, the proposed parallelization model outperforms inter-task parallelization. Fig.8 shows the processing time of each thread. Load unbalancing occurs in the inter-task parallelization model, which leads to a decline in the performance of the inter-task parallelization model. On the other hand, in the case of task-10000, the inter-task parallelization model outperforms the other parallelization models. task-10000 contains many small-sized tasks; therefore, it does not benefit from dividing the task.

To evaluate the length of the divided range in the proposed parallelization model, we changed its size to 1,000, 5,000, and 10,000 terms. Fig.9 shows the speed-up ratio of each divided range size. The results show that for the small divided tasks of 1,000 and 5,000, the performance of the proposed model is good. This indicates that by reducing the size of the divided task, overload can be reduced.

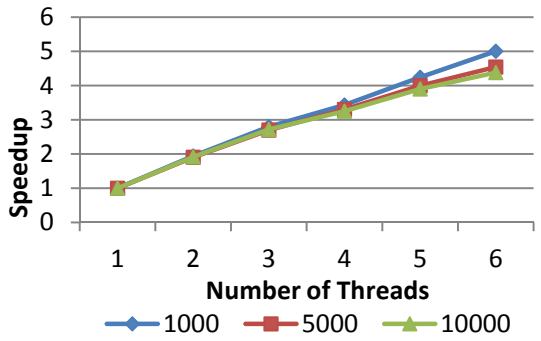


Fig. 9. Speed-up ratio (Task-1000).

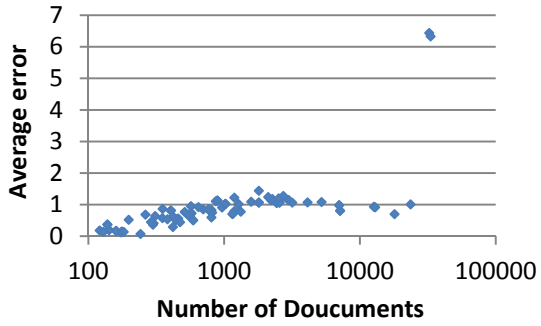


Fig. 10. Average error.

C. Experiment 2

In Experiment 2, we investigate the precision of the intra-task parallelization model. Errors occur in this model, because it divides an inter-arrival time sequence into partitions and detects bursts in each partition individually. In this experiment, we measured the error between a state transition sequence obtained in the proposed model and an original state transition sequence that is obtained using Kleinberg's burst detection algorithm. An average error is defined as

$$averageerror = \frac{\sum_{i=1}^{|s|} |s_i - s_k^r|}{|s|} \quad (9)$$

Fig.10 shows all the average errors. The vertical axis of the figure represents the average error while the horizontal axis denotes the number of documents that include a certain term. Overall, almost all the average errors are less than 1.5. This indicates there is no problem to assessment of accuracy.

D. Experiment 3

In Experiment 3, we created six virtual tasks in which the length of the inter-arrival time sequence is 4.6 million. Not more than one task can be performed on the memory in a PC. because the size of the required memory is more than that of the PC. We compare the proposed, inter-task, and intra-task parallelization models.

Table 1 shows the results of the experiment. In the case of more than two threads, the inter-task parallelization model needs more processing time. In the inter-task parallelization model, two tasks are performed at the same time. Therefore, the parallelization model uses considerably more memory than the main memory. This causes OS thrashing. On the other hand, the processing times of the proposed parallelization model and intra-task parallelization do not increase. The proposed parallelization model performs tasks on memory because it utilizes the intra-parallelization model.

TABLE I
RESULT OF EXPERIMENT 3.

Thread	*Inter-Task Parallelization Model(s)*	*Intra-Task Parallelization Model(s)*	*Proposed Parallelization Model(s)*
2	4299.58465	369.043796	294.083362
3	29897.8395	263.967795	202.185169
6	-	183.964157	120.792438

VI. CONCLUSION

In this paper, we proposed a novel parallelization model for burst detection in a large-scale document stream. The proposed method combines the inter-task and intra-rask parallelization models. This combination provides seamless dynamic load balancing, and detects burst in a large-scale document stream on memory. The experimental results showed the efficiently of the proposed method. In our future work, we intend to investigate the trade-off relationship between speed-up and accuracy. In addition, we will develop a real time algorithm for burst detection in a large-scale document stream using parallelization.

ACKNOWLEDGEMENT

This work was supported in part by a Grant-in-Aid for Young Research (B) (No. 23700124) from the Ministry of Education, Culture, Sports, Science and Technology, Japan and a Grant-in-Aid for Scientific Research (C) (2) (No. 20500137) from the Japanese Society for the Promotion of Science, Japan.

REFERENCES

[1] J. Allan, R. Papka, and V. Lavrenko, "On-line new event detection and tracking," in *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pp. 37–45, 1998.

[2] J. Kleinberg, "Temporal dynamics of on-line information streams," in *IN DATA STREAM MANAGEMENT: PROCESSING HIGH-SPEED DATA*, Springer, 2006.

[3] J. Kleinberg, "Bursty and hierarchical structure in streams," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pp. 91–101, 2002.

[4] Y. Zhu and D. Shasha, "Efficient elastic burst detection in data streams," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 336–345, 2003.

[5] R. Kumar, J. Novak, P. Raghavan, and A. Tomkins, "On the bursty evolution of blogspace," in *Proceedings of the 12th international conference on World Wide Web*, pp. 568–576, 2003.

[6] K. K. Mane and K. Börner, "Mapping topics and topic bursts in pnas," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101 Suppl 1, pp. 5287–5290, 2004.

[7] G. P. C. Fung, J. X. Yu, P. S. Yu, and H. Lu, "Parameter free bursty events detection in text streams," in *Proceedings of the 31st international conference on Very large data bases*, pp. 181–192, 2005.

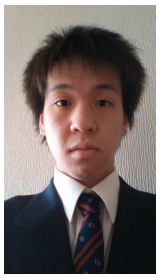
[8] X. Wang, C. Zhai, X. Hu, and R. Sproat, "Mining correlated bursty topic patterns from coordinated text streams," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 784–793, 2007.

[9] Q. He, K. Chang, E.-P. Lim, and J. Zhang, "Bursty feature representation for clustering text streams," in *Proceedings of the Seventh SIAM International Conference on Data Mining*, 2007.
[10] D. He and D. S. Parker, "Topic dynamics: an alternative model of bursts in streams of topics," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 443–452, 2010.
[11] J. Yao, B. Cui, Y. Huang, and X. Jin, "Temporal and social context based burst detection from folksonomies," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, 2010.
[12] Z. Hui, Y. Xiaokang, S. Toru, and K. Iwane, "Parallel viterbi decoding implementation by multi-microprocessors," *IEICE transactions on communications*, vol. 76, no. 6, pp. 658–666, 1993.
[13] E. Yeo, S. Ausburger, W. R. Davis, and B. Nikolic, "Implementation of high throughput soft output viterbi decoders," in *Proceedings of IEEE Workshop on Signal Processing Systems*, pp. 146–151, 2002.
[14] L. Wang and Z.-y. Li, "Design and implementation of a parallel processing viterbi decoder using fpga," *Memory*, pp. 77–80, 2010.
[15] P. Edmonds, E. Chu, and A. George, "Dynamic programming on a shared-memory multiprocessor," *Parallel Computing*, vol. 19, no. 1, pp. 9–22, 1993.
[16] Z. Galil and K. Park, "Parallel algorithms for dynamic programming recurrences with more than o(1) dependency," *Parallel Computing*, vol. 21, no. 2, pp. 213–222, 1994.
[17] D. G. Morales, F. Almeida, C. Rodríguez, J. L. Roda, I. Coloma, and A. Delgado, "Parallel dynamic programming and automata theory," *Parallel Computing*, vol. 26, no. 1, pp. 113–134, 2000.
[18] G. Tan, N. Sun, and G. R. Gao, "A parallel dynamic programming algorithm on a multi-core architecture," in *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '07, pp. 135–144, 2007.
[19] R. A. Chowdhury and V. Ramachandran, "Cache-efficient dynamic programming algorithms for multicores," in *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, SPAA '08, pp. 207–216, 2008.
[20] G. Tan, N. Sun, and G. R. Gao, "Improving performance of dynamic programming via parallelism and locality on multicore architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, pp. 261–274, 2009.

Kaishi Hirahara is a student at the Department of Intelligent Systems, Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan. His research interests include parallel computing.

Keiichi Tamura received his B.Eng., M.Eng., and Ph.D. degrees in Information Science from Kyushu University, Fukuoka, Japan, in 1998, 2000, and 2005, respectively. He is presently Associate Professor at the Department of Intelligent Systems, Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan. He has been Treasurer of IEEE SMC Hiroshima Chapter since 2012. His research interests include parallel computing, data engineering, data mining, high performance computing and evolutionary computation.

Hajime Kitakami received his M.Eng. from Tohoku University in 1976 and Ph.D. in engineering from Kyushu University in 1992. He has been a Professor in the Department of Intelligent Systems, Graduate School of Information Sciences, Hiroshima City University in Japan since 1994. His paper was recorded as the 25th Anniversary Best Paper Award of Information Processing Society of Japan (IPSJ) in 1985. He received Paper Award from Japanese Society for Engineering Education (JSEE) in 2003. His research interests' include database, data mining, distributed parallel processing, and bioinformatics. He has been an editorial board member for Transactions on Mathematical Modeling and its Applications (TOM), Journal of the Information Processing Society of Japan (IPSJ) since 2006. Also, he has been an editorial board member for Journal of the Database Society of Japan (DBSJ) since 2008.



Shingo Tamura is a student at the Department of Intelligent Systems, Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan. His research interests include document stream.