Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/090337

Mining Vehicles Frequently Appearing Together from Massive Passing Records

Dongjin Yu[1,*], *Wensheng Dou*[1], *Wanqing Li*[1], *Suhang Zheng*[1] *and Jianhua Shao*[2,3]

[1] Hangzhou Dianzi University, Hangzhou, China
[2] Zhejiang Topcheer Information Technology Co., Ltd, Hangzhou, China
[3] Zhejiang Provincial Key Laboratory of Network Technology and Information Security, China

Received: 7 Aug. 2014, Revised: 8 Nov. 2014, Accepted: 9 Nov. 2014
Published online: 1 May 2015

Abstract: Vehicles Frequently Appearing Together, or VFATs, can be clues in solving criminal cases. Traditional sequence mining approaches help identify VFATs from passing-through records collected at monitoring sites. However, huge traffic data streams hinder fast identification of VFATs. In this paper, we present a multi-threaded approach to fast identification of VFATs based on multi-core processors, called Frequent Sequential Mining based on Multi-Cores (FSMMC). It parallels the execution of tasks, partitions large volumes of data, and obtains VFATs by merging local candidates discovered in different threads running on different processor cores. Through local parallel reduction, FSMMC eliminates the repetitive patterns and reduces computational effort. Moreover, it achieves workload balance by the dynamic distribution of tasks to a pool of threads where the thread that finishes first joins another running thread. Both theoretical analysis and case studies show that FSMMC takes full advantage of multi-core computing platforms and has higher speed-up when searching VFATs among massive passing through records, compared with other approaches without multi-threading.

Keywords: massive data mining, parallel, sequential patterns, multi-core, Vehicles Frequently Appearing Together

1 Introduction

When solving criminal cases, Vehicles Frequently Appearing Together, or VFATs, can sometimes be valuable clues. Collecting records on vehicles passing through from different monitoring sites and then searching for vehicles frequently appearing together has been proven to be an effective manner to find VFATs. However, such investigation always involves large traffic streams and therefore takes a long time. Moreover, VFATs have high mobility and can usually escape notice. How to quickly identify VFATs from massive traffic data streams therefore becomes a key issue.

In recent years, various methods of data mining have matured and been applied widely in various fields, including the discovery of motifs in DNA sequences, the analysis of web log and customer shopping sequences, and study of XML query access patterns [1]. Frequent pattern discovery or sequential mining, which was pioneered by the works of Agrawal et al. in the Apriori algorithm [2], could be used to find VFATs. The problem with frequent patterns, given a minimum support threshold *min_sup*, is in discovering all the item sets that occur at least *min_sup*times in the database. Here, vehicles frequently appearing together can be regarded as frequent patterns, i.e., they often appear somewhere as a whole.

High-performance computation utilities, such as multi-core and many-core servers, offer ideal mining platforms. The problem in finding VFATs is therefore how to fully exploit the parallelism, or harness the power of these multi-core processors. A number of works have focused on parallel formulations for finding frequent patterns on shared-memory computers and GPU nodes [3, 4]. Indeed, many parallel computing models already exist. For example, OpenMP is a well-known parallel framework supporting multi-platform shared-memory parallel programming in C/C++. Although OpenMP is simple to use because of its automatic data layout and decomposition by directives, it lacks reliable error handling and fine-grained mechanisms to control thread-processor mapping. In this paper, we present a novel parallel frequent sequential mining approach

* Corresponding author e-mail: yudj@hdu.edu.cn

employing multi-cores called FSMMC (Frequent Sequential Mining based on Multi-Cores) to search for VFATs. FSMMC takes threads as the "parallel unit" and can minimize memory bandwidth and maximize cache reuse. Both theoretical analysis and case studies indicate that it is an efficient multi-core implementation.

The structure of the paper is as follows. In section 2, we briefly define the problem of extracting frequent patterns with multi-core processors, describe the approach in depth, and provide the necessary theoretical background. In section 3, we theoretically evaluate the performance of the approach; we thus prove that the method is precise, with lower calculation complexity and more feasibility. Section 4 presents the detailed experimental results, comparing the approach with different numbers of running threads on a multi-core processor. In section 5 we then review the current state of parallel pattern mining technology. Finally, section 6 concludes the paper and gives directions for future work.

2 The FSMMC Approach

2.1 Definitions

Definition 1 *The itemset of vehicle passing record in a given database D, is denoted as a quadruple, i.e.,* $I_{p,t,l,d} =< p,t,l,d >$ *, in which p represents the vehicle plate number, t represents the time of passing through monitoring sites, l represents the location of monitoring sites, and d represents the vehicle driving direction.*

Definition 2 *A sequence s is an ordered list of itemsets in a period of time* D_t*, denoted as* $S =< I_{p_1,t_1,l_1,d_1}, I_{p_2,t_2,l_2,d_2}, \ldots, I_{p_m,t_m,l_m,d_m} >$.

Definition 3 *The term motorcade sequence is used to represent the group of vehicles passing through the same monitoring sites in the same direction in the time of interval* Δ_t*, denoted as: ;*

$$S_m = \left\{ (p_1,\ldots,p_i,\ldots,p_j,\ldots,p_n) \;\middle|\; \begin{array}{c} < I_{p_1,t_1,l,d},\ldots,I_{p_i,t_i,l,d},\ldots,I_{p_j,t_j,l,d},\ldots,I_{p_n,t_n,l,d} > \in S \\ |t_i - t_j| \le \Delta_t, n \in \{1,\ldots,m\} \\ l \in \{l_1,\ldots,l_m\}, d \in \{d_1,\ldots,d_m\} \end{array} \right\} \quad (1)$$

The length of S_m*, denoted as* $|S_m|$*, is the number of the itemsets* S_m *holds.*

Definition 4 *Given a database D storing the vehicle passing records, the support of the sequence* S_m *, or the reliability as the suspect VFAT, is denoted as* $sup(S_m) = S_t * S_l / D_t$*, where* S_t *denotes the count of* S_m *occurring, and* s_l *denotes the number of passing monitoring sites* S_m *covers.*

Definition 5 *Given a minimum support threshold min_sup, if* $sup(S_m) \ge min\_sup$*,* S_m *is then called a frequent sequence* $S^{'}$*, i.e., VFATs. The collection of* $S^{'}$ *is denoted as* L_N *when* $|S^{'}| = N$.

Definition 6 *Given a database D storing the vehicle passing records, all* S_m *it holds are called candidate sequences, denoted as* C_N *when* $|S_m| = N$*. In other words,*$L_N = \{C_N | sup(C_N) \ge min\_sup\}$.

Definition 7 *In the multi-thread environment, the subsequence of* C_N *acquired on thread i is called* C_N^i*. There exists* $\sum_{i=1}^{n} C_N^i = C_N$*, where n is the total number of running threads. To distinguish different motorcade sequences from* C_N^i*,*$V_{C_N^i}^{S_m}$ *is used to represent the set of* C_N^i *when its sequential value is* S_m.

Definition 8 *Given a sequence database D storing the vehicle passing records, let* T_s *be the serial sequence mining time with a single-core processor, and let* $T(q)$ *be the parallel sequence mining time with q-core processors. The speed-up is then defined as* $S(q) = T_s / T(q)$.

2.2 The FSMMC Approach

The FSMMC approach is designed to be executed on a shared memory system. It partitions the workload into independent tasks, but assumes that the whole dataset is accessible to all threads. In this way, each thread runs independently through lock-free programming without the need for inter-thread communication.

In order to combine the properties of multi-core processors, the FSMMC approach can be further divided into three phases: 1) the global database is divided into several local datasets for each thread by means of the equidistant static projection method; 2) local motorcade sequential patterns are located in each thread by local parallel reduction; 3) local motorcade patterns are dynamically combined into the frequent sequential patterns. These phases are illustrated in Figure 1.

1) The complete database D is partitioned to D_i and assigned to the thread $i(i = 1,2,\ldots,n)$ for loading. The global database D is divided into $D_1, D_2, \ldots, D_n$, and $D = \cup_{i=1}^{n} D_i$. If there are R records in D, then the records R_i for thread i are shown as (2). Here, M_i^j represents record j in the local database D_i and T_p represents record p in the global database D.

$$R_i = \left\{ M_i^j \;\middle|\; \begin{array}{c} M_i^j = T_p, p = \lfloor \frac{n}{R} \rfloor * (i-1) + j, \\ p \in \left[\lfloor \frac{n}{R} \rfloor * (i-1), \lfloor \frac{n}{R} \rfloor * i \right] \end{array} \right\} \quad (2)$$

2) For thread $i(i = 1,2,\ldots,n)$, the local database is scanned once to find all motorcade sequential patterns; where necessary, they are then reduced and stored in files. Because D_i is always too large to store in memory wholly, FSMMC needs further division to get

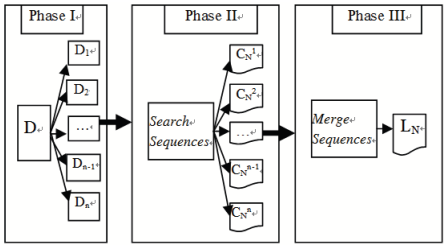

Fig. 1: The process of FSMMC approach.

smaller datasets $D_i^{'}$. Then, it spawns n threads, each scanning $D_i^{'}$ to get candidate sequences. Considering this step is one of the most costly steps, we use local parallel reduction to eliminate the repetitive patterns. It is very likely that one certain task has a lower computational cost than all the others. Therefore, FSMMC creates the thread pool within which each thread is assigned to one certain task of pattern searching. Those which finish searching first will join in with other threads. In other words, FSMMC allows each thread to process asynchronously, which can help to gain space and reduce running time efficiently.

3) Local motorcade patterns are combined in each storing file and final frequent sequences are derived. After being processed by each thread, the reduction objects need to be merged. First, FSMMC puts the tasks of combining files in a global task list after the files have been regularly marked, making sure each task has a number corresponding to its rank. Then, every thread selects a task from the list as their own assignment and independently eliminates infrequent motorcade items. Since all threads are independent of each other, only their calculation workloads required to be balanced in order to boost performance.

FSMMC repeatedly checks whether there is an idle core. If one exists, it selects a new task from the global task list and runs it. All frequent motorcade sequences will then be finally identified when the task list becomes empty.

3 Performance Evaluation

In this section, we evaluate the performance of FSMMC by checking its running time. Suppose the time we spend on the first phase, i.e., the phase where the global database D is divided into $D_1, D_2, \ldots, D_n$, is T_1. The time that the thread $i (i = 1, 2, \ldots, n)$ spends on database D_i to find the motorcade sequential pattern S_{m_j} is $Q_i^j (j = 1, 2, \ldots, m; i = 1, 2, \ldots, n)$. We use m to indicate the number of motorcade sequences on D_i and n as the total number of threads. The total time thread i spends on D_i to find all the local sequences can then be represented as:

$$T_i = \sum_{j=1}^{m} Q_i^j, i = 1, 2, \ldots, n \quad (3)$$

On the other hand, through the parallel computation on multi-cores, the time for generating all the motorcade sequential patterns is:

$$T_2 = Max(T_i) = Max\left(\sum_{j=1}^{m} Q_i^j\right), i = 1, 2, \ldots, n \quad (4)$$

However, the time for the traditional serial computational method to find the sequential patterns is equivalent to the sum time of each thread treated separately, as is:

$$T_2^{'} = \sum_{i=1}^{n} T_i = \sum_{i=1}^{n} \sum_{j=1}^{m} Q_i^j, i = 1, 2, \ldots, n \quad (5)$$

In the third phase (combining local patterns to obtain all the motorcades' frequent sequences), the time that thread i takes is:

$$T_i = \sum_{j=q}^{k} F_i^j + t_i, q = 1, 2, \ldots, k; i = 1, 2, \ldots, n \quad (6)$$

in which, F_i^j represents the processing time for file j, k means the total number of files for combining, and t_i is the system overhead for threads accessing the global task list, fetching new assignments and other system operations. Remarkably, $t_i \leq \sum_{j=q}^{k} F_i^j$. So, the time FSMMC spends on this phase by parallel processing on multi-cores is:

$$T_3 \approx Max(T_i) \approx Max\left(\sum_{j=q}^{k} F_i^j + t_i\right), \quad q = 1, 2, \ldots, k; i = 1, 2, \ldots, n \quad (7)$$

However, relative to parallel processing, the time for traditional serial sequential processing approximates to:

$$T_3^{'} \approx \sum_{i=1}^{n} T_i = \sum_{i=1}^{n} \sum_{j=q}^{k} (F_i^j + t_i), \quad q = 1, 2, \ldots, k; i = 1, 2, \ldots, n \quad (8)$$

In conclusion, the total time with FSMMC is:

$$T = T_1 + T_2 + T_3 = T_1 + Max\left(\sum_{j=1}^{m} Q_i^j\right) + Max\left(\sum_{j=q}^{k} F_i^j + t_i\right), q = 1, 2, \ldots, k; i = 1, 2, \ldots, n \quad (9)$$

The total running time with the traditional serial approach is:

$$T^{'} = T_1 + T_2^{'} + T_3^{'} = T_1 + \sum_{i=1}^{n} \sum_{j=1}^{m} Q_i^j + \sum_{i=1}^{n} \sum_{j=q}^{k} (F_i^j + t_i), q = 1, 2, \ldots, k; i = 1, 2, \ldots, n \quad (10)$$

Therefore, because $Max(\sum_{j=1}^{m} Q_i^j) \leq \sum_{i=1}^{n} \sum_{j=1}^{m} Q_i^j$ and $Max(\sum_{j=q}^{k} F_i^j + t_i) \leq \sum_{i=1}^{n} \sum_{j=q}^{k} (F_i^j + t_i)$, FSMMC approach can achieve higher performance on multi-core processors.

4 Case Studies

4.1 Case Environments

The FSMMC approach has been successfully used in fast identification of VFATs based on massive traffic data streams. In the experiment, VFATs are defined as N suspect motorcades, which pass through the monitoring sites with the support over *min_sup*.

In the testing phase, the attributes of vehicle passing records include plate number, time of passing by monitoring sites, location of monitoring sites and vehicle driving direction. We ran the test program on an Intel Core 2 processor with 2.40G Hz and 2GB RAM running Windows XP. The databases used contained about 3,000,000 records. The FSMMC approach was implemented with JDK 1.6.

4.2 Case Results

We ran the FSMMC approach in different scales of traffic streams by spawning varying numbers of threads, where each thread executed the same code for frequent sequence mining. The approach provided good extensibility by optionally changing the number of threads optionally. Input datasets of the same size were used and all the results were saved in a file on hard disks to be used later. The results generated are shown in Figure 2. When the *min_sup* is assigned 2.50, VFATs are the top 15 records. A more detailed analysis of the average running time used to search for VFATs is illustrated in Figure 3.

As shown in the Figure 3, the more sequences generated, the more calculation time for file reduction is required. However, as the number of threads increases, the increase becomes less, especially when the dataset has more than 1,000,000 records. Specific to a certain multi-core system, the approach can employ resources of existing multi-core processors through multithread programming technology, leading to better results on larger volumes of datasets.

In order to verify the effectiveness of the FSMMC approach more intuitively, we analysed the speed-up of different threads on a four-core and a two-core processor (using the same datasets with 2,700,000 records). Figure 4 shows the average T(2) is about 897.4 seconds in a multithreading environment from one thread to five threads, whereas T(4) is about 261.2 seconds. Thus, the average S(4)/S(2) is approximately 3.44. Furthermore, as can be seen in Figure 4, a processor with more cores can obtain more stable results. Due to the dynamic task distribution mechanisms and local parallel reduction, the FSMMC approach reduces idle core time and the time required to combine sequences. It incorporates runtime performance characteristics and succeeds in using multi-core processor collaboration to optimize the performance of the parallel approach. This approach could therefore achieve good performance in identifying VFATs from massive traffic data streams.

CarNum1	CarNum2	St	Sl	Dt	sup(Sm)
BMA763V	G220PEG	19	8	5day	30.40
SQM888Y	E384VET	22	5	8day	13.75
G380RQY	BC877SB	8	3	2day	12.00
E676LOU	SLRU898	3	4	1day	12.00
G380RQY	BC877SB	9	6	6day	9.00
SL55AMG	BMW229V	6	3	2day	9.00
0006JDW	BC65LAD	20	2	5day	8.00
E384VET	MO65VET	4	2	1day	8.00
CHE150A	ARB150N	14	3	7day	6.00
LGB200V	LES986X	2	7	3day	4.67
WAX180X	HA515LK	9	4	9day	4.00
SLK320L	LES986X	15	2	8day	3.75
GT11POR	ANT16DA	12	2	8day	3.00
A911P0W	K155WRX	18	1	6day	3.00
SL55AMG	BMW229V	15	1	6day	2.50

Fig. 2: VFATs found in the case where $N = 2(vehicles)$, $\delta_t = 60(seconds)$ and *min_sup*=2.5.

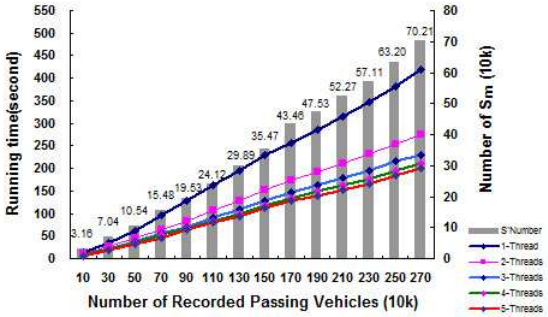

Fig. 3: Running time of searching for VFATs from different scales of passing vehicles by FSMMC on a four-core processor.

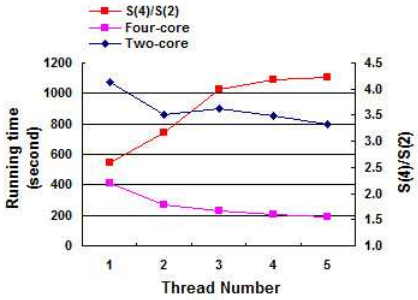

Fig. 4: Running time and speed-ups on multiple cores with different thread numbers.

5 Related Works

The efficient analysis of spatio-temporal data, generated by moving vehicles, is an essential requirement for intelligent transportation services. To our knowledge, such research currently focuses mainly on the methods of efficiently extracting long sharable frequent routes [5,6], or Swarms [7], but not deliberately trailing vehicles. In contrast to the ridesharing application, the identification of VFATs involves a huge amount of data and therefore demands more mining power.

Frequent pattern mining is a core field in data mining research. Since the first solution to the problem of frequent item-set mining was presented by Agrawal et al. [8], various specialized in-memory data structures have been proposed to improve mining efficiency [9]. It has been recognized that the set of all frequent item-sets is too large to be analysed and the information they contain is therefore redundant. To remedy this, numerous works have studied parallel frequent pattern mining on clusters to improve mining efficiency [10,11]. These works explore a spectrum of trade-offs between computation, communication, memory usage, synchronization, and the use of problem-specific information in parallel data mining. However, the experiments showed synchronization costs became quite large if the data distributions were skewed or the nodes were not equally capable.

Considering multi-core systems with lower inter-processor communication costs and limited off-chip bandwidth, parallel frequent pattern mining on multi-core processors was pioneered by Buehrer et al. [12,13]. Based on the serial algorithm gSpan [14] and the similar study by Worlein et al. [15], Buehrer et al. proposed a parallel frequent graph mining algorithm with excellent scale-up properties. Their contribution comprises an efficient way to decompose work and to explore the search space in a depth-first way. They also proposed a way to exploit temporal locality of the cache. However, this method needs excessive memory consumption due to its static embedding techniques. Lucchese et al. proposed similar strategies for mining closed frequent item-sets, which contain optimizations for improving cache usage when creating conditional databases (called projections in their paper) [16]. Tatikonda et al. studied the approaches on parallel frequent tree mining [17]. Their algorithm could scale up very well with the number of cores, leading to a quasi-linear speed-up in a lot of real-world databases. However, it costs too much time for memory accesses.

The past few years have also witnessed the emergence of several novel approaches other than the multi-core ones for the implementation and deployment of large-scale data mining. MapReduce, which has been popularized by Google, is a scalable and fault-tolerant data processing model that enables to process a massive volume of data in parallel with many low-end computing nodes. We in [18] introduce a parallel implementation of BIDE algorithm on MapReduce, called BIDE-MR. The experiments on an Apache Hadoop cluster show that BIDE-MR attains good parallelization. However, the approach presented in this paper is easier to be implemented since it effectively utilizes the multi-core structure of the single node.

6 Conclusions

This paper presents a novel approach to the fast identification of VFATs from massive traffic data streams on multi-core processors. To harness the power of the multi-core processors, we use a dynamic task distribution mechanism to balance the workloads of different threads. A thread-steal happens when a task is not comparable with the cumulative cost of the other tasks. Both theoretical analysis and case studies show that the approach takes good advantage of multi-core computing platforms and has higher performance and speed-up, compared with other approaches without multi-threading.

It is notable that sequential pattern mining requires iterative scans of the sequence dataset with numerous data comparisons and analyses. In other words, it is memory intensive. Therefore, optimizations of massive storage access are always needed. Other problems, such as how to increase the certainty of thread scheduling and how to limit the search space to further improve accuracy, still need to be studied.

Acknowledgements

The work is supported by Natural Science Foundation (No.61472112), Natural Science Foundation of Zhejiang (No.LY12F02003), the Key Science and Technology Project of Zhejiang (No. 2012C11026-3, No. 2008C11099-1) and the open project of Zhejiang Provincial Key Laboratory of Network Technology and Information Security. The authors would also like to thank anonymous reviewers who made valuable suggestions to improve the quality of the paper.

References

[1] Agrawal, R., Srikant, R., Mining sequential patterns. In: Proc. of ICDE, Taipei, Taiwan, Mar., 1995.

[2] Agrawal, R., Srikant, R., Fast algorithms for mining association rules. In: Proc. of VLDB, 1994, pp. 487-499.

[3] Jin, R., Yang, G., Agrawal, G., Shared memory parallelization of data mining algorithms: Techniques, programming interface, and performance. IEEE Trans. on Knowl. and Data Eng., vol. 17, no. 1, 2005, pp. 71-89.

[4] Fang, W., Lu, M., Xiao, X., He, B., Luo, Q., Frequent itemset mining on graphics processors. In DaMoN 09: Proc. of the 5th International Workshop on Data Management on New Hardware, New York, NY, USA, ACM , 2009, pp. 34-42.

[5] Gidfalvi, G., Pedersen, T.B., Mining long, sharable patterns in trajectories of moving objects. GeoInformatica, vol. 13, no. 1, 2009, pp. 27-55.
[6] Xue, G., Li, Z., Zhu, H., Liu, Y., Traffic-known urban vehicular route prediction based on partial mobility patterns. In: Proc. of the International Conference on Parallel and Distributed Systems - ICPADS, 2009, pp. 369-375.
[7] Li, Z., Ding, B., Han, J., Kays, R., Swarm: Mining Relaxed Temporal Moving Object Clusters. In: Proc. of the VLDB Endowment, vol. 3, no. 1, 2010, pp. 723-734.
[8] Agrawal, R., Imilienski, T., Swami, A., Mining association rules between sets of items in large databases. In: Proc. of SIGMOD, 1993, pp. 207-216.
[9] Goethals, B., Survey on frequent pattern mining. In: http://~citeseer.ist.psu.edu/goethals03survey.html, 2003
[10] Agrawal, R., Shafer, J. C., Parallel mining of association rules. IEEE Trans. Knowl. Data Eng., vol. 8, no. 6, 1996, pp. 962-969.
[11] Zaki, M. J., Parthasarathy, S., Ogihara, M., Li, W., Parallel algorithms for discovery of association rules. Data Min. Knowl. Discov., vol. 1, no. 4, 1997, pp. 343-373.
[12] Buehrer, G., Parthasarathy, S., Chen, Y. K., Adaptive parallel graph mining for CMP architectures. In: Proc. of ICDM, 2006, pp. 97-106.
[13] Buehrer, G., Parthasarathy, S., Kim, D., Towards data mining on emerging architectures. In: Proc. of 9th SIAM Workshop on High Performance and Distributed Mining. Bethesda, USA, 2006.
[14] Yan, X., Han, J., gSpan: Graph-based substructure pattern mining. In: ICDM, 2002, p. 721.
[15] Worlein, M., Meinl, T., Fischer, I., Philippsen, M., A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston. In: Proc. of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Porto, Portugal, 2005, pp. 392-403.
[16] Lucchese, C., Orlando, S., Perego, R., Parallel mining of frequent closed patterns: Harnessing modern computer architectures. In: Proc. of ICDM, 2007, pp. 242-251.
[17] Tatikonda, S., Parthasarathy, S., Mining Tree-Structured Data on Multicore Systems. In: Proc. of VLDB, 2009, pp. 694-705.
[18] Yu, D., Wu, W., Zheng, S., Zhu, Z., BIDE-based parallel mining of frequent closed sequences with MapReduce, LNCS 7440, 2012, pp.177-186.

Dongjin Yu is currently a professor at Hangzhou Dianzi University and a visiting scholar of University of California, Santa Barbara. He received his BS and MS in Computer Applications from Zhejiang University in China, and PhD in Management from Zhejiang Gongshang University in China. His current research efforts include intelligent information processing, program comprehension and service computing. He is especially interested in the novel approaches to constructing large enterprise information systems effectively and efficiently by emerging advanced information technologies. He is the director of Institute of Cloud and Big Data and vice director of Institute of Intelligent and Software Technology of Hangzhou Dianzi University. He is a member of ACM and IEEE, and a senior member of China Computer Federation (CCF). He is also a member of Technical Committee of Software Engineering CCF (TCSE CCF) and a member of Technical Committee of Service Computing CCF (TCSC CCF).

Wensheng Dou is currently a postgraduate at Hangzhou Dianzi University, China. He has participated in some government-funded projects related with data management. His current research interests mainly include data mining and big data processing.

Wanqing Li received his PhD degree in mechanics of solid from Lanzhou University in China in 2007 and works as an Associate Professor in Hangzhou Dianzi University. His present interests are numerical parallel computing and data mining.

Suhang Zheng received her master degree in computer science from Hangzhou Dianzi University in China. She has published a number of high-quality papers related with data mining. She now works for Alibaba.com.

Jianhua Shao received his bachelor's degree in Mathematics from Fudan University in China. His primary research area includes networking computing and system integration.